Received: 18 July 2020 | Revised: 21 March 2021 | Accepted: 22 March 2021

DOI: 10.1002/nla.2384

RESEARCH ARTICLE

WILEY

The Sherman–Morrison–Woodbury formula for generalized linear matrix equations and applications

Yue Hao[1] | Valeria Simoncini[2]

[1]School of Mathematics and Statistics, Lanzhou University, Lanzhou, PR China

[2]Dipartimento di Matematica and AM^2, Alma Mater Studiorum - Università di Bologna and IMATI-CNR, Bologna, Italia

Correspondence
Valeria Simoncini, Dipartimento di Matematica and AM2, Alma Mater Studiorum - Università di Bologna and IMATI-CNR, Bologna, Italia.
Email: valeria.simoncini@unibo.it

Funding information
Indam-GNCS, Grant AlmaIdea 2017-2020 - Università di Bologna, Grant/Award Number: 2017-2020; China Scholarship Council, Grant/Award Number: Contract No. 201906180033; National Natural Science Foundation of China, Grant/Award Number: 11471150

Abstract
We discuss the use of a matrix-oriented approach for numerically solving the dense matrix equation $AX+XA^T+M_1XN_1+\ldots+M_\ell XN_\ell=F$, with $\ell\geq 1$, and M_i,N_i, $i=1,\ldots,\ell$ of low rank. The approach relies on the Sherman–Morrison–Woodbury formula formally defined in the vectorized form of the problem, but applied in the matrix setting. This allows one to solve medium size dense problems with computational costs and memory requirements dramatically lower than with a Kronecker formulation. Application problems leading to medium size equations of this form are illustrated and the performance of the matrix-oriented method is reported. The application of the procedure as the core step in the solution of the large-scale problem is also shown. In addition, a new explicit method for linear tensor equations is proposed, that uses the discussed matrix equation procedure as a key building block.

KEYWORDS
linear matrix equations, linear tensor equations, Schur decomposition, Sherman–Morrison–Woodbury formula, Sylvester equation

1 | INTRODUCTION

We are interested in solving dense linear matrix equations in the form

$$AX+XA^T+MXM^T=F, \tag{1}$$

where $A,F\in\mathbb{R}^{n\times n}$ and M of rank $s\ll n$. We assume that A has no eigenvalues λ such that $\lambda=-\overline{\lambda}$, and that X can be uniquely determined.

Equation (1) is a simplified version of the following more general linear equation (see, e.g., References 1, [2, ch. 12]),

$$AX+XA^T+M_1XN_1+\ldots+M_\ell XN_\ell=F, \tag{2}$$

with $M_i,N_i\in\mathbb{R}^{n\times n}$, $i=1,\ldots,\ell$ not necessarily symmetric, having rank $s_{M_i},s_{N_i}\ll n$, respectively. These matrix equations are characterized by the presence of a Lyapunov operator $X\mapsto AX+XA^T$ and of extra terms $\sum_{i=1}^{\ell}M_iXN_i$, which make the closed form of the solution hard to formulate solely in terms of the given coefficient matrices. In particular, even

This version is dated February 26, 2021.

in the simplest case (1) of three terms, that is $\ell = 1$, no closed-form based on the Schur decomposition exists, which is instead the case for $\ell = 0$. Solvability conditions associated with the relative roles of the two operators $X \mapsto AX + XA^T$ and $X \mapsto \sum_{i=1}^{\ell} M_i X N_i$ and geometry properties of the solutions are studied, for example, in Reference [3, sec.3.4].

This type of equation classically arises in Control, for instance as the Gramian equations of bilinear dynamical systems; see, for example, Reference 4. The problem has been recently attacked in the large-scale case, which provides additional challenges.[3,5–7] Multiterm equations in the form (2) may also arise in discretizing elliptic equations using a matrix-oriented approach,[8,9] or in parametric and time-dependent problems.[10,11] In all these settings, the low-rank framework depends on the problem hypotheses.

For large-scale matrix equations with sparse data, a small number of methods is available in the literature, and they are based on the efficient matrix-oriented application of standard vector iterative methods; these are, for example, References 5–7,12,13, alternating least squares,[14] subspace projection [5, sec. 5.2],.[15] In general, these methods all rely on the possibility of determining a good low-rank approximation to the solution, and this problem is analyzed, for instance, in Reference 5. Methods that implicitly exploit the Kronecker form of the problem have also been analyzed,[16–18] and these are close to the method we are going to discuss.

We aim at analyzing a strategy for solving *dense or banded small and medium size* problems of type (1), and generalizing it to (2); we do not make any rank assumption on the sought after numerical solution. As we will see, the approach can then be used as a building block for large-scale projection-type methods, or for solving linear *tensor* equations.

The problem in (1) can be stated in a more familiar form using the Kronecker version of the problem. Given two matrices $A = (A_{i,j})$ and $B = (B_{i,j})$, the Kronecker product is defined in block form as

$$A \otimes B = \begin{bmatrix} A_{1,1}B & \dots & A_{1,n}B \\ \vdots & \ddots & \vdots \\ A_{n,1}B & \dots & A_{n,n}B \end{bmatrix}.$$

This matrix operator satisfies

$$\mathrm{vec}(AXB) = (B^T \otimes A)\mathrm{vec}(X), \tag{3}$$

where vec(X) stacks the columns of X one below the other. Using this property in (1) with $\mathcal{A} = A \otimes I + I \otimes A$, $f = \mathrm{vec}(F)$, and $M = UV^T$ so that we can define $\mathcal{U} = U \otimes U$, $\mathcal{V} = V \otimes V$, the matrix equation can be written in vectorized form as $(\mathcal{A} + \mathcal{U}\mathcal{V}^T)x = f$, with $x = \mathrm{vec}(X)$ and $\mathrm{rank}(\mathcal{U}) = s^2 = \mathrm{rank}(\mathcal{V})$. Note that $\mathcal{U} = [\mathrm{u}_1, \dots, \mathrm{u}_{s^2}]$ with $\mathrm{u}_t = u_k \otimes u_i$ where $t = (k-1)s + i$. If s^2 is still much lower than n, the vector x can be effectively determined by means of the Sherman–Morrison–Woodbury (SMW) formula[19–21] as follows

$$x = (\mathcal{A} + \mathcal{U}\mathcal{V}^T)^{-1}f = \mathcal{A}^{-1}f - \mathcal{A}^{-1}\mathcal{U}(I + \mathcal{V}^T\mathcal{A}^{-1}\mathcal{U})^{-1}\mathcal{V}^T\mathcal{A}^{-1}f. \tag{4}$$

Hager[22] described many properties of this formula and illustrated a variety of applications where the low-rank update naturally arises, such as statistics, matrix partitioning, networks, linear programming, discretization of partial differential equations (PDEs), physics. In particular, this formula is a building block of many numerical linear algebra methods, such as preconditioning and quasi-Newton methods, see, for example, References 23–25, [26, sec.6.1]. It has been classically used in bordering, partitioning, tearing, and modifications of sparse linear systems,[27] [28,29, sec.11.6], among others. In general, the formula is quite appealing every time the modification $\mathcal{U}\mathcal{V}^T$ destroys some computationally convenient structure of $\mathcal{A}$, which can still be exploited in the SMW formula. Classical notes of caution regarding the use of the SMW formula, however, state that the updating method (4) cannot be expected to be numerically stable in all cases. In particular, problems will arise when the initial problem is more ill-conditioned than the modified one. In other words, $\kappa(\mathcal{A})$ could be significantly larger than $\kappa(\mathcal{A} + \mathcal{U}\mathcal{V}^T)$, hence possible worse accuracy is known to be expected in some cases when using the SMW formula. In the following, we will assume that the method is *stable*, that is, all the steps associated with the application of the formula are stable.[30] Indeed, in our treatment we are mainly interested in devising an implementation of the SMW formula that takes into account the matrix equation structure, assuming the whole computation is stable.

The use of the Kronecker form within the SMW formula has a serious computational drawback, which makes the formula hard to apply. The matrix $\mathcal{A}$ has very large dimensions even for moderate n, since $\mathcal{A}$ is $n^2 \times n^2$. In particular, if A is not sparse, the matrix $\mathcal{A}$ is going to be rather full, and thus impossible to store online. Fortunately, the solution X

can be directly obtained by only relying on the original matrices A, U, V, and F in (1) by maintaining and exploiting their structure as much as possible. The SMW formula is still employed but at the matrix level, with great advantages both in terms of memory consumption and computational costs. A possibly first description of a general procedure that avoids the Kronecker product in the symmetric case can be found in Reference 31. In there, however, there is no explicit derivation of the SMW formula associated with a possibly low-rank structure as in our setting. Nonetheless, it is acknowledged that the overall cost can be significantly lower than that of the vectorized form as long as the rank of U, V is moderate. In Reference 32 a generalization to systems of matrix equations stemming from Control applications is given. These procedures target small size problems, for which dense computations can be carried out. A later work by Damm[16] explicitly focuses on the symmetric version of (2), that is $N_i = M_i^T$, in a way that the use of the SMW formula becomes apparent. The implementation differs from ours, and we provide a computational cost comparison showing the benefits of a full matrix-oriented implementation. Finally, more recently, the authors of Massei et al.[17] have developed the same implementation that we are going to discuss, however, their focus is on (very) large problems (1) with the highly structured symmetric matrix A, and the role of s is not analyzed (in fact, $s = 1$ is used); the computational cost of the overall method is not discussed. A similar procedure is employed in Reference 18 for preconditioning purposes, without a specialized analysis, and in Reference 33 for a related problem.

We aim to provide a detailed analysis of the matrix-oriented implementation of the SMW formula for solving dense or structured (e.g., banded A) equations of type (2), with particular attention to the role of the ranks s_i and the number ℓ of terms. We also discuss how this implementation can be used as a building block for methods to solving more complex problems.

This methodology can be extended in a straightforward manner to the case when the Lyapunov operator $X \mapsto AX + XA^T$ is replaced by the *Sylvester* operator $X \mapsto AX + XB$ with no major change in the algorithm. On the other hand, dealing with the Lyapunov operator allows us to lighten the presentation, by using a single matrix A. We refrain from repeating our same arguments for the Sylvester operator, and we directly use this variant whenever needed.

The synopsis of the article is as follows. In Section 2 we discuss the SMW formula in matrix form for solving the small and medium size linear matrix equation, and then extend it to the general problem in Section 3. Considerations on stability are briefly discussed in Section 4. In Section 5, some numerical experiments are given to illustrate the performance of our strategy on generally random data, while in Section 6 we illustrate the applicability of this strategy to moderate size matrices stemming from the discretization of certain PDEs. More general numerical linear algebra problems where this methodology can be effectively employed are discussed in subsequent sections. In particular, in Section 7 we consider using this approach for solving the reduced problem associated with a Galerkin projection method for the large-scale multiterm linear equation. Moreover, in Section 8 we illustrate the applicability of our scheme to a new solution strategy for linear tensor equations. Finally, a brief conclusion is given in Section 9.

2 | THE SMW FORMULA IN MATRIX FORM

The standard SMW formula (4) in vectorized form is applied using the following sequence of computations,

Algorithm 0.

```
1. Solve $\mathcal{A}w = f$
2. Solve $\mathcal{A}\mathrm{p}_j = \mathrm{u}_j$ where $\mathcal{U} = [\mathrm{u}_1, \ldots, \mathrm{u}_{s^2}]$ to give $\mathcal{P} = [\mathrm{p}_1, \ldots, \mathrm{p}_{s^2}]$;
3. Compute $H = I + \mathcal{V}^T\mathcal{P} \in \mathbb{R}^{s^2\times s^2}$
4. Solve $Hg = \mathcal{V}^T w$
5. Compute $x = w - \mathcal{P}g$.
```

We next illustrate that all computations associated with n^2-long vectors can be transformed into matrix-matrix operations of size n throughout the whole procedure. The first step satisfies

$$w = \mathcal{A}^{-1}f \quad \Leftrightarrow \quad AW + WA^T = F, \tag{5}$$

that is, a Lyapunov equation of size $n \times n$ can be solved to obtain W such that $w = \mathrm{vec}(W)$. A Schur decomposition-based method such as the Bartels–Stewart algorithm can be used to this end if A is dense and of moderate size.[34]

Regarding step 2, for each $\mathrm{u}_j, j = 1, \ldots, s^2$, noticing that $\mathrm{u}_j = \mathrm{vec}(u_i u_k^T)$ for k, i such that $j = (k-1)s + i$, we have

$$\mathrm{p}_j = \mathcal{A}^{-1}\mathrm{u}_j \quad \Leftrightarrow \quad AP_j + P_jA^T = u_iu_k^T, \quad \mathrm{p}_j = \mathrm{vec}(P_j). \tag{6}$$

Therefore, once again a sequence of small dense Lyapunov equations can be solved.

The third step determines $H = I + \mathcal{V}^T\mathcal{A}^{-1}\mathcal{U} = I + \mathcal{V}^T\mathcal{P} = I + \mathcal{V}^T[\mathrm{p}_1, \ldots, \mathrm{p}_{s^2}]$, which can equivalently be computed by using

$$\mathrm{v}_j^T\mathcal{A}^{-1}\mathrm{u}_t = v_i^TP_tv_k, \quad j = (k-1)s + i.$$

Analogously, the solution of the $s^2 \times s^2$ linear system $Hg = \mathcal{V}^Tw$ requires computing the right-hand side. This can be obtained in terms of the original data and of (5) as follows

$$\mathcal{V}^T\mathcal{A}^{-1}f = \begin{bmatrix} v_1^TWv_1 \\ v_2^TWv_1 \\ \vdots \\ v_s^TWv_s \end{bmatrix}. \tag{7}$$

The final step can equivalently be obtained as $X = W - \sum_{j=1}^{s^2} P_j(g)_j$, although the vectorized version $x = w - \mathcal{P}g$ may give better efficiency.

The complete procedure is summarized in Algorithm 1.

Algorithm 1. Sherman–Morrison–Woodbury formula for (1)

```
1: INPUT: $A, F \in \mathbb{R}^{n\times n}$, $U, V \in \mathbb{R}^{n\times s}$
2: OUTPUT: Numerical solution $X \in \mathbb{R}^{n\times n}$ to $AX + XA^T + UV^TX(UV^T)^T = F$
3: Compute Schur decomposition of $A$: $A = QRQ^*$
4: Change of basis for $F$, $U$ and $V$ wrto $Q$
5: Solve $AW + WA^T = F$ for $W$ using $Q, R$
6: For $k, i = 1, \ldots, s$ solve $AP_j + P_jA^T = u_iu_k^T, j = (k-1)s + i$ for $\mathrm{p}_j = \mathrm{vec}(P_j)$, using $Q, R$
7: For $k, i = 1, \ldots, s$ compute the component $(d)_j = v_i^TWv_k$ and
       $$H_{j,t} = e_j^Te_t + \mathrm{v}_j^T\mathcal{A}^{-1}\mathrm{u}_t = e_j^Te_t + v_i^TP_tv_k,$$
   with $j = (k-1)s + i$, $t = 1, \ldots, s^2$, and $e_j$ the $j$th column of the identity matrix.
8: Solve $Hg = d$ for $g$
9: Compute $X = W - \sum_{j=1}^{s^2} P_j(g)_j$
```

We explicitly observe that the computations within each cycle in steps 6 and 7 are completely independent, and can be performed in parallel. In step 6, the procedure requires the solution of s^2 Lyapunov equations, together with the solution of a Lyapunov equation in step 5. This computation can be significantly accelerated by first performing a single (real) Schur decomposition of the matrix A: the right-hand side is written in terms of the Schur orthogonal basis, and only backward substitutions with quasi-triangular matrices need to be carried out to solve the Lyapunov/Sylvester equation. This step is particularly efficient whenever A is normal, since in this case the Schur decomposition leads to a diagonal matrix, so that all remaining computations can be performed elementwise. Indeed, the matrix equations to be solved in the orthogonal basis have the form

$$RY + YR^* = G. \tag{8}$$

Matrix R is diagonal for A normal, and upper quasi-triangular when A is nonnormal, so different solvers are adopted. For R diagonal, it holds

TABLE 1 Computational cost of Algorithm 1

Step	Computational cost
Schur decomposition of A	$25n^3$
Computation H	$s^2(2n^3+n^2)+s^4(2n^2+n-1)+4n^2s+s^2-2ns$
Computation d	$6n^3+2s^2n^2+s^2n-2n^2-s^2$
Computation g	$\frac{2}{3}s^6$
Computation X	$(2s^2-2)n^2+4n^3$

$$Y = G \oslash (r\mathbf{1}^T + \mathbf{1}r^T), \quad r = [R_{11}, \dots, R_{nn}]^T, \tag{9}$$

where $a \oslash b$ denotes element-by-element division, and $\mathbf{1}$ is the vector of all ones. When R is upper quasi-triangular, a recursive triangular solver can be used; see, for example, Reference 35 and its references[*].

The main computational costs of Algorithm 1 are summarized in Table 1, giving a total of

$$(35 + 2s^2)n^3 + (2s^4 + 5s^2 + 4s - 4)n^2 + (s^4 + s^2 - 2s)n + \frac{2}{3}s^6 - s^4 \tag{10}$$

floating-point operations for determining the final solution. The cost for computing the Schur decomposition of A can be significantly alleviated if A has further structure. For instance, this cost becomes $\mathcal{O}(n^2)$ if A is symmetric and tridiagonal. We numerically experimented with different matrix structures to show the impact of this computation on the overall procedure. Moreover, additional sparsity of some of the quantities, such as in U_i, V_i may be exploited during the computation. These are clearly problem-dependent issues and are not taken into account in the general (worst case) cost description. In the actual implementation, the Lyapunov equation solution for all u_i, u_k is split into different stages, and the matrix H is generated during the solution of these Lyapunov equations in the eigenvector basis, so that the actual costs are spread through the algorithm. In Appendix A we report a typical implementation, pointing to the most expensive steps.

The algorithm leading cost is associated with the solution of s^2 Lyapunov equations with the same matrix A, and different right-hand sides. For $s = \mathcal{O}(1)$ this cost is largely dominant, and it may be convenient to simplify the algorithm and just use the vectorized form (Algorithm 0 above), limiting the exploitation of the matrix structure in step 2 to the solution of the s^2 Lyapunov equations instead of the large system with $\mathcal{A}$. Indeed, taking into account the underlying structure of $\mathcal{V}$ appears to be irrelevant for $s = \mathcal{O}(1)$. We refer to this simplified implementation as a hybrid version. For this reason, in our experimental analysis we consider larger values of s, for which a fully matrix-oriented version becomes relevant. The same occurs when more than one low-rank term appears in (2), that is $\ell > 1$, in which case the dimensions of $\mathcal{U}$, $\mathcal{V}$, and thus that of H, quickly increase, as it will be discussed in the following section.

Remark 1. Though we mainly focus on solving (2) for dense and moderate size problems, it is of interest to linger over the large-scale case. For large A, the low rank of the right-hand side terms $u_iu_k^T$ allows one to employ powerful recently developed iterative methods.[15] Since $s^2 + 1$ such equations need to be solved, however, this cost is affordable as long as $s = \mathcal{O}(1)$, and in this case the hybrid approach could be used. In Section 7, we explore an ad hoc large-scale approach for (2) based on the projection onto a single approximation space, that uses the SMW formula as a core solver for an occurring small size problem.

We conclude this cost analysis by reporting on the comparison with a similar implementation proposed by Damm,[16] whose algorithm is reported in Appendix B, together with the computational costs of the main floating-point operations. For a fair comparison, we only considered the operations marked with $\star$, since the remaining ones handle nongeneric situations. The leading total cost of the implementation proposed by Damm is given by

$$(45 + 10s^2)n^3 + 2(s^4 + s^2 - 4)n^2 + \frac{2}{3}s^6 - s^4 - s^2,$$

which differs from ours mainly in the coefficient of the n^3 term. Figure 1 shows typical cost curves of the two approaches as s (left) and n (right) vary. The factors in n^3 are dominant for s small, thus being in favor of our implementation, whereas

[*]In our Matlab implementation we did not exploit this more efficient version, we simply used the `lyap` function with triangular coefficient matrices.

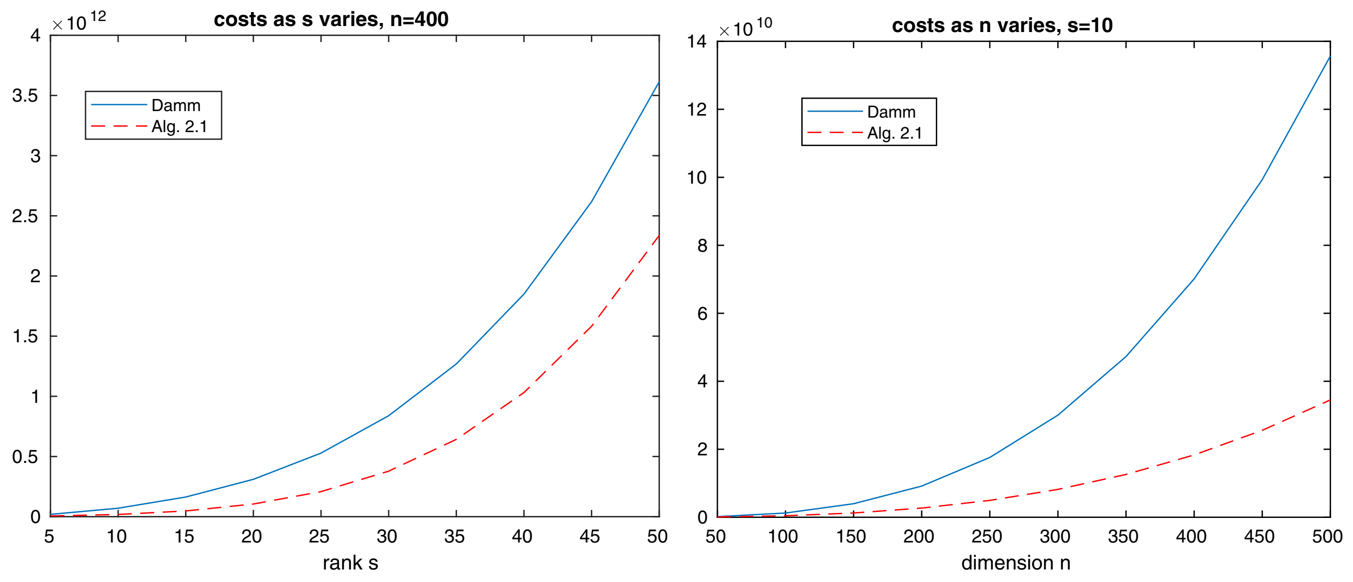

FIGURE 1 Floating point computations for the implementation in algorithm B1 and in Algorithm 1 as s and n vary. Left: $n = 400$. Right: $s = 10$

for large values of s the costs for the two implementations become more comparable, also due to the presence of similar factors in high powers of s.

3 | THE SMW FORMULA FOR THE GENERAL CASE

The strategy in Section 2 can be extended to the following nonsymmetric problem with more than three linear terms in X,

$$AX + XA^T + U_1V_1^TX(U_2V_2^T)^T + U_3V_3^TX(U_4V_4^T)^T = F, \tag{11}$$

where $U_i = [u_1^{(i)}, \dots, u_{s_i}^{(i)}]$, $V_i = [v_1^{(i)}, \dots, v_{s_i}^{(i)}] \in \mathbb{R}^{n\times s_i}$, of rank s_i, for $i = 1, \dots, 4$; we notice the change of notation for the rank, to adhere to the adopted notation for the low-rank matrices $U_iV_i^T$. The considered procedure is also applicable to the case of even more terms, and it is effective as long as the total rank of the U_is and the V_is remains moderate with respect to n. To pass to the Kronecker form of the problem, we define

$$\mathcal{U} = [U_2 \otimes U_1, U_4 \otimes U_3] \quad \text{and} \quad \mathcal{V} = [V_2 \otimes V_1, V_4 \otimes V_3], \tag{12}$$

so that (11) can be written as

$$(\mathcal{A} + \mathcal{U}\mathcal{V}^T)x = f, \tag{13}$$

where $\mathcal{A}$ and f are defined as in Section 2, with $\mathcal{U} = [\mathrm{u}_1, \dots, \mathrm{u}_{s_1s_2+s_3s_4}]$ and $\mathcal{V} = [\mathrm{v}_1, \dots, \mathrm{v}_{s_1s_2+s_3s_4}]$. Thus, the problem (13) goes back to the problem (4). The matrix-oriented strategy proceeds as follows.

1. Solve (5) to get W;
2. For $i = 1, \dots, s_1$ and $k = 1, \dots, s_2$, solve $AP_j + P_jA^T = u_i^{(1)}(u_k^{(2)})^T$, $j = (k-1)s_1 + i$ for $\mathrm{p}_j = \mathrm{vec}(P_j)$;
3. For $i = 1, \dots, s_3$ and $k = 1, \dots, s_4$, solve $AP_j + P_jA^T = u_i^{(3)}(u_k^{(4)})^T$, $j = (k-1)s_3 + i + s_1s_2$ for $\mathrm{p}_j = \mathrm{vec}(P_j)$;
4. For $i = 1, \dots, s_1$ and $k = 1, \dots, s_2$, compute the entries $(d)_j = (v_i^{(1)})^TWv_k^{(2)}$ and

$$H_{j,t} = e_j^Te_t + \mathrm{v}_j^T\mathcal{A}^{-1}\mathrm{u}_t = e_j^Te_t + (v_i^{(1)})^TP_tv_k^{(2)},$$

for $j = (k-1)s_1 + i$, $t = 1, \dots, s_1s_2 + s_3s_4$;

5. For $i = 1, \ldots, s_3$ and $k = 1, \ldots, s_4$, compute the entries $(d)_j = (v_i^{(3)})^T W v_k^{(4)}$ and

$$H_{j,t} = e_j^T e_t + \mathrm{v}_j^T \mathcal{A}^{-1} \mathrm{u}_t = e_j^T e_t + (v_i^{(3)})^T P_t v_k^{(4)},$$

   for $j = (k-1)s_3 + i + s_1 s_2$, $t = 1, \ldots, s_1 s_2 + s_3 s_4$;
6. Solve $Hg = d$ for g;
7. Compute $X = W - \sum_{j=1}^{s_1 s_2 + s_3 s_4} P_j (g)_j$.

A Matlab[36] implementation of this algorithm is reported in Appendix A for completeness[†].

4 | CONSIDERATIONS ON STABILITY

It is well known that using the SMW formula in the vectorized case may have disastrous stability effects depending on the choices of $\mathcal{U}, \mathcal{V}$;[22] Yip[30] gives suggestions on how these two low-rank matrices could be selected to lower these effects, in case these matrices can be tuned.

The stability considerations in Reference 30 focus on the conditioning of the matrix $I + \mathcal{V}^T \mathcal{P}$, which in turn also depends on the stability properties of the equation associated with $\mathcal{P}$. Our computation differs from the vectorized approach for the use of Sylvester/Lyapunov matrix equations instead of algebraic (vector) systems to determine $\mathcal{P}$. Higham [37, chapter 15] gives a thorough account of the accuracy and stability properties associated with the solution of linear matrix equations. Let $AX + XB = C$ be the matrix equation to be solved, and assume that a perturbation on the data occurs. Let

$$\epsilon = \max \left\{ \frac{\|\Delta A\|_{\mathcal{F}}}{\|A\|_{\mathcal{F}}}, \frac{\|\Delta B\|_{\mathcal{F}}}{\|B\|_{\mathcal{F}}}, \frac{\|\Delta C\|_{\mathcal{F}}}{\|C\|_{\mathcal{F}}} \right\},$$

where $\|\cdot\|_{\mathcal{F}}$ denotes the Frobenius norm. In Reference [37, section 15.3] it is shown that the solution is perturbed by ΔX satisfying

$$\frac{\|\Delta X\|_{\mathcal{F}}}{\|X\|_{\mathcal{F}}} \leq \sqrt{3}\Psi\epsilon,$$

and the bound is sharp, as first-order bound in ϵ; here, in our notation,

$$\Psi = \|\mathcal{A}^{-1}[\|A\|(X^T \otimes I), \|B\|(I \otimes X), -\|C\| I_{mn}]\|_2 / \|X\|_{\mathcal{F}},$$

and $\mathcal{A} = I \otimes A + B^T \otimes I$. The quantity Ψ plays the role of the condition number for the Sylvester equation.

A weaker bound, which actually corresponds to applying standard perturbation theory to the Kronecker form of the problem, is given by Higham [37, section 15.3].

$$\frac{\|\Delta X\|_{\mathcal{F}}}{\|X\|_{\mathcal{F}}} \leq \sqrt{3}\Phi\epsilon, \quad \Phi = \|\mathcal{A}^{-1}\|_2 \frac{(\|A\|_{\mathcal{F}} + \|B\|_{\mathcal{F}})\|X\|_{\mathcal{F}} + \|C\|_{\mathcal{F}}}{\|X\|_{\mathcal{F}}}.$$

Hence, comparing Ψ and Φ corresponds to measuring the relative solution sensitivity with respect to the input data, associated with using either the vector or the matrix approaches for the problem with $\mathcal{A}$. It was shown in Reference [37, section 15.3] that Φ and Ψ "can differ by an arbitrary factor." That is, although from their definition we have $\Psi \leq \Phi$ and they have in general similar magnitude, it may happen that $\Psi \ll \Phi$. Thus, the analysis seems to favor the matrix equation approach. An alternative analysis could be performed that makes use of the sep function.[35,37] These results provide insights into the different behavior of the two formulations under perturbations in the data. It would be interesting to analyze whether similar differences carry over in finite precision arithmetic computations. A thorough analysis remains an open problem.

[†]The general Matlab code will be made available by the authors on their webpages.

We next linger over the effect of computational perturbations on the final solution accuracy. Let $x_f = \mathcal{A}^{-1}f + e_f$ be the numerical solution obtained with the chosen method to solve $\mathcal{A}x = f$, and analogously, $x_{\mathcal{U}} = \mathcal{A}^{-1}\mathcal{U} + E_{\mathcal{U}}$. Then the numerical solution to $(\mathcal{A} + \mathcal{U}\mathcal{V}^T)x = f$ can be written as

$$\tilde{x} = \mathcal{A}^{-1}f + e_f - (\mathcal{A}^{-1}\mathcal{U} + E_{\mathcal{U}})(I + \mathcal{V}^T(\mathcal{A}^{-1}\mathcal{U} + E_{\mathcal{U}}))^{-1}\mathcal{V}^T(\mathcal{A}^{-1}f + e_f),$$

where we assume that sums and products among matrices and vectors in the formula are exact. Note that both $\|e_f\|$ and $\|E_{\mathcal{U}}\|$ are related to the accuracy of the method used to solve the Sylvester equation with A.

Let $H_e = I + \mathcal{V}^T(\mathcal{A}^{-1}\mathcal{U} + E_{\mathcal{U}})$ and $H = I + \mathcal{V}^T\mathcal{A}^{-1}\mathcal{U}$. Then, assuming for simplicity that the system with H or H_e is solved exactly, we can write

$$\begin{aligned} x - \tilde{x} &= -\mathcal{A}^{-1}\mathcal{U}(H^{-1} - H_e^{-1})\mathcal{V}^T\mathcal{A}^{-1}f - e_f + E_{\mathcal{U}}H_e^{-1}\mathcal{V}^T\mathcal{A}^{-1}f \\ &\quad + \mathcal{A}^{-1}\mathcal{U}H_e^{-1}\mathcal{V}^T e_f + E_{\mathcal{U}}H_e^{-1}\mathcal{V}^T e_f \\ &= -\mathcal{A}^{-1}\mathcal{U}H^{-1}\mathcal{V}^T E_{\mathcal{U}}H_e^{-1}\mathcal{V}^T\mathcal{A}^{-1}f + E_{\mathcal{U}}H_e^{-1}\mathcal{V}^T\mathcal{A}^{-1}f \\ &\quad + \mathcal{A}^{-1}\mathcal{U}H_e^{-1}\mathcal{V}^T e_f + E_{\mathcal{U}}H_e^{-1}\mathcal{V}^T e_f - e_f \end{aligned}$$

Hence,

$$\|x - \tilde{x}\| \le \|(I - \mathcal{A}^{-1}\mathcal{U}H_e^{-1}\mathcal{V}^T)e_f\| + \|(I - \mathcal{A}^{-1}\mathcal{U}H^{-1}\mathcal{V}^T)E_{\mathcal{U}}H_e^{-1}\mathcal{V}^T\mathcal{A}^{-1}f\| + \mathcal{O}(\boldsymbol{u}^2),$$

where we assumed that $\|e_f\| \le c_1\boldsymbol{u}$, $\|E_{\mathcal{U}}\| \le c_2\boldsymbol{u}$, in which $\boldsymbol{u}$ is the machine precision and the constants c_1, c_2 depend on the data. The two matrices $(I - \mathcal{A}^{-1}\mathcal{U}H_e^{-1}\mathcal{V}^T)$ and $(I - \mathcal{A}^{-1}\mathcal{U}H^{-1}\mathcal{V}^T)$ in the bound above are related to the quality of the data, and the conditioning of the obtained H_e and H, respectively. Some calculations would recover a bound similar to that by Yip[30] for $(I - \mathcal{A}^{-1}\mathcal{U}H_e^{-1}\mathcal{V}^T)$ and $(I - \mathcal{A}^{-1}\mathcal{U}H^{-1}\mathcal{V}^T)$.

5 | NUMERICAL EXPERIMENTS WITH DENSE OR STRUCTURED DATA

In this section, we illustrate the performance of Algorithm 1 when using dense and banded random data. This random setting provides a frame of reference for the computational costs associated with the methods in the worst scenario, that of dense data. All considered methods can also appropriately exploit sparsity whenever available, in different ways, and this will be explored in later sections. In all displayed tables we report the CPU time, and the relative error and residual norms

$$\text{Err} := \frac{\|X - X_\star\|_F}{\|X_\star\|_F}, \qquad \text{Res} := \frac{\|F - \mathcal{L}(X)\|_F}{\|F\|_F}, \tag{14}$$

where $X_\star$ is our "true" solution, computed as a random matrix with uniform elements in the interval $(0, 1)$, so that the right-hand side is determined explicitly: for instance, for (1) it is given as $F := \mathcal{L}(X_\star)$ with $\mathcal{L}(X) := AX + XA^T + UV^TX(UV^T)^T$. Computational comparisons are principally performed among the matrix-oriented and vectorized forms of the SMW formula. For the sake of completeness and as a reference target, in all our tables we also report on the performance of a direct method for explicitly solving $(\mathcal{A} + \mathcal{U}\mathcal{V}^T)x = f$ (backslash in Matlab); we notice that this is a built-in (compiled) code, which is expected to be (much) faster than an interpreted code implemented within Matlab. In spite of this, our results show that the matrix-oriented method is able to outperform the direct solver in practically all cases, and in some instances with orders of magnitude better CPU times, both in the dense and sparse cases.

Example 1. *Symmetric and dense matrix A for various s's.* We consider the problem

$$AX + XA + U_1V_1^TXV_1U_1^T + U_3V_3^TXV_3U_3^T = F$$

with the symmetric matrix $A = A_0 + A_0^T$ where A_0 is full and it has random entries uniformly distributed in (0,1); U_1, V_1 and U_3, V_3 are also random matrices from the same distribution. Due to the symmetry of A, the eigendecomposition of A was computed once at the beginning and the Lyapunov equations solved as described in Section 2. The computational

TABLE 2 Example 1

		Direct			Matrix form			Vectorized form		
n	s_1/s_3	**CPU**	**Res**	**Err**	**CPU**	**Res**	**Err**	**CPU**	**Res**	**Err**
40	3/5	0.08	1.2e-15	3.7e-12	0.02	1.2e-14	2.1e-11	0.23	1.2e-13	3.5e-11
	4/6	0.08	2.2e-15	5.2e-12	0.02	6.5e-15	1.8e-11	0.29	1.0e-12	4.8e-11
	5/7	0.10	1.1e-15	5.7e-11	0.02	1.1e-14	1.2e-10	0.37	4.9e-12	2.2e-10
80	3/5	2.01	3.4e-15	3.3e-11	0.02	5.7e-15	4.6e-11	6.14	8.3e-13	9.4e-10
	4/6	2.05	2.3e-15	2.8e-10	0.02	3.3e-15	1.4e-10	8.19	3.9e-12	6.6e-10
	5/7	2.00	2.7e-15	7.2e-11	0.03	5.7e-14	2.0e-10	10.6	1.5e-12	1.8e-09
160	3/5	85.6	9.2e-15	1.5e-10	0.04	2.4e-14	3.9e-10	168	1.6e-13	9.6e-09
	4/6	86.4	9.0e-15	1.4e-09	0.05	7.4e-14	5.3e-09	211	3.1e-11	7.9e-08
	5/7	86.9	6.4e-15	3.0e-10	0.08	8.6e-14	2.3e-09	257	2.7e-12	2.0e-07

Note: Symmetric and dense matrix A, U_1, V_1 and U_3, V_3 for various s_1, s_3.

TABLE 3 Example 2

		Direct			Matrix form			Vectorized form		
U_1, U_3	***n***	**CPU**	**Res**	**Err**	**CPU**	**Res**	**Err**	**CPU**	**Res**	**Err**
nonorth	40	0.08	9.1e-16	3.8e-12	0.02	4.8e-14	1.9e-11	0.04	1.5e-12	1.1e-10
	80	1.94	1.1e-15	5.8e-10	0.02	3.0e-14	2.3e-10	0.22	1.7e-11	2.6e-09
	160	85.1	5.9e-15	9.2e-09	0.04	2.5e-13	1.9e-08	1.24	1.5e-10	7.4e-08
	320	–	–	–	0.08	1.4e-12	5.0e-08	6.64	8.3e-11	4.8e-06
orth	40	0.08	3.2e-15	8.9e-14	0.02	1.8e-15	1.1e-14	0.04	2.6e-16	9.9e-15
	80	1.88	6.0e-15	3.2e-13	0.02	1.7e-15	2.1e-14	0.18	1.8e-16	2.2e-14
	160	84.8	1.7e-14	2.7e-12	0.03	1.7e-15	1.6e-13	1.32	2.5e-15	3.2e-12
	320	–	–	–	0.07	2.0e-15	3.9e-13	6.81	1.6e-15	1.3e-12

Note: Symmetric and pentadiagonal matrix A; U_1, V_1 and U_3, V_3 with different geometric properties. Here, $s_1 = 3$ and $s_3 = 5$. "–" stands for excessive computational time.

results for increasing n and s_1, s_3 are reported in Table 2, and show the great advantages of the matrix setting, gaining several orders of magnitude in CPU time, while maintaining at least the same accuracy as with the vectorized approach.

Example 2. *Symmetric and banded matrix A.* For the same matrix equation as in Example 1, we first consider A a random and symmetric pentadiagonal matrix, and U_1, V_1 and U_3, V_3 random matrices with $s_1 = 3$, $s_3 = 5$ columns, respectively.

The method performance is reported in Table 3. We also slightly modified the problem, by imposing that the U_is and V_is have orthonormal columns. This latter setting showed better accuracy of the obtained solution, as reported in the bottom results in Table 3; this is in full agreement with the results in Reference 30.

Example 3. *Random, nonsymmetric and dense matrix A.* Let A, U_1, V_1 and U_3, V_3 be random matrices for the same problem as in Example 1. The numerical results are listed in Table 4. As we expected, the vectorized form pays the price of a fully populated *and nonsymmetric* matrix. The nonsymmetry also affects the performance of the matrix method, since the full Schur form needs to be used, and Lyapunov equations with triangular matrices solved in all instances. The total costs remain very modest, though.

Example 4. *Dense A and four term equations.* We consider the equation

$$AX + XA^T + U_1V_1^TX(U_2V_2^T)^T + U_3V_3^TX(U_4V_4^T)^T = F,$$

TABLE 4 Example 3

		Direct			Matrix form			Vectorized form		
n	s_1/s_3	CPU	Res	Err	CPU	Res	Err	CPU	Res	Err
40	3/5	0.09	9.9e-16	1.8e-11	0.08	1.4e-13	3.9e-09	0.24	9.5e-12	4.3e-09
	4/6	0.08	1.4e-15	8.8e-12	0.09	3.9e-14	9.7e-11	0.33	9.5e-13	7.9e-10
	5/7	0.09	5.1e-16	8.9e-11	0.12	3.5e-14	5.9e-10	0.35	4.1e-12	1.2e-09
80	3/5	2.05	2.8e-15	3.2e-10	0.15	2.5e-13	2.6e-09	4.98	1.1e-12	1.0e-08
	4/6	2.16	3.0e-15	4.2e-10	0.18	6.9e-13	9.3e-10	6.56	1.9e-12	1.5e-08
	5/7	2.02	3.8e-15	7.1e-10	0.22	1.7e-13	8.5e-09	7.48	1.5e-11	1.9e-08
160	3/5	83.3	8.8e-15	1.2e-09	0.48	2.6e-12	1.3e-08	205	3.0e-12	1.0e-07
	4/6	84.3	7.8e-15	1.8e-09	0.69	1.2e-12	1.5e-08	199	2.0e-12	2.5e-07
	5/7	100.4	1.2e-14	2.1e-09	0.84	1.0e-11	4.1e-08	278	3.9e-11	4.2e-07

Note: Numerical results for nonsymmetric and dense matrix A, U_1, V_1 and U_3, V_3 for various s_1, s_3 columns, respectively.

TABLE 5 Example 4

		Direct			Matrix form			Vectorized form		
n	s_i	CPU	Res	Err	CPU	Res	Err	CPU	Res	Err
40	2/3/4/5	0.09	1.6e-15	1.7e-11	0.08	9.6e-14	1.6e-10	0.20	3.5e-13	7.2e-10
	4/5/6/7	0.09	1.2e-15	9.6e-12	0.10	9.8e-14	5.7e-11	0.34	1.2e-12	2.1e-10
	6/7/8/9	0.09	1.6e-15	1.9e-11	0.14	3.0e-13	2.1e-10	0.58	4.0e-12	9.2e-10
80	2/3/4/5	1.94	4.2e-15	1.0e-10	0.12	1.6e-13	1.1e-09	4.41	6.2e-12	3.1e-09
	4/5/6/7	1.93	2.1e-15	9.7e-10	0.19	2.7e-13	1.7e-09	7.10	3.9e-12	1.8e-08
	6/7/8/9	1.94	4.1e-15	6.5e-10	0.32	2.5e-13	1.8e-08	10.6	8.8e-11	7.5e-09
160	2/3/4/5	76.4	9.3e-15	7.9e-10	0.40	5.7e-13	4.5e-09	248	1.6e-12	3.1e-08
	4/5/6/7	74.6	6.1e-15	1.3e-09	0.80	1.2e-12	1.3e-08	268	1.8e-11	4.7e-08
	6/7/8/9	77.3	8.6e-15	2.3e-08	1.38	1.5e-12	1.1e-07	266	3.0e-11	8.3e-07

Note: Numerical results for a random nonsymmetric problem.

with U_i, V_i random matrices with s_i columns, and A a random matrix (all uniformly distributed in $(0, 1)$). The results are reported in Table 5. The results are consistent with our previous findings, also when several terms occur.

6 | APPLICATION TO DISCRETIZED PDES

In this section, we report on the computational solution of the small-scale problem (2) in the context of matrix-oriented discretization of PDEs, and in problems naturally described in terms of multiterm matrix equations including low-rank coefficient matrices. In all cases, the coefficient matrices are sparse, and all methods have been implemented so as to make the best use of sparsity. Once again, the solution with the (sparse) direct method is reported as a reference, taking into account that CPU time comparisons are biased by the different implementation settings (see the related discussion in Section 5).

Our first example was discussed in Reference [5, sec.6.2] in the equation solving context, and then employed for comparison purposes in Reference 7 in the large-scale case.

Example 5. A nonlinear RC circuit. The data stems from the model reduction of a nonlinear system, representing a scalable RC ladder with k resistors, with an exponential dependence on the voltage-current, giving rise to the problem

TABLE 6 Example 5

		Direct		Matrix form		Vectorized form	
n	**Rank of U_1**	**CPU**	**Res**	**CPU**	**Res**	**CPU**	**Res**
110	1	0.82	1.3e-16	0.03	1.4e-15	1.52	2.9e-16
	3	1.06	1.7e-16	0.07	1.8e-15	1.77	5.4e-16
	5	0.97	3.0e-16	0.11	1.8e-15	2.17	2.9e-16
	10	0.92	1.4e-16	0.35	1.3e-15	3.81	7.3e-16
156	2	4.49	2.4e-16	0.08	1.5e-15	7.70	4.5e-16
	4	7.45	8.0e-17	0.15	1.5e-15	8.23	1.6e-16
	6	4.54	1.1e-16	0.30	1.5e-15	10.1	2.5e-16
	12	6.05	2.6e-16	1.04	1.5e-15	18.5	5.1e-16
240	2	28.1	5.9e-17	0.15	2.3e-15	62.9	1.7e-16
	5	32.8	8.4e-17	0.75	2.3e-15	72.1	1.0e-16
	7	31.1	2.1e-16	1.48	2.4e-15	82.1	5.7e-16
	15	32.2	2.4e-16	7.05	2.2e-15	159.0	1.2e-15

Note: CPU time and residual norms as the problem dimension and the rank of U_1 vary.

of type (1) of dimension $n=k(k+1)$ with $N_1 = M_1^T$ and $\ell=1$. The matrix $M_1 = U_1V_1^T$ has rank $s=k$. To explore how the performance depends on the rank, in our experiments we considered portions of these matrices by taking the first r_1 columns of U_1, V_1, with r_1 up to k. Our numerical results are listed in Table 6 and illustrate the effectiveness of the matrix-oriented approach with respect to the vector settings, at a comparable residual accuracy.

The next examples stem from the discretization of two-dimensional convection-diffusion equations on rectangular domains, with homogeneous Dirichlet boundary conditions. The matrix-oriented discretization of this class of PDEs was treated in Reference 9 for separable coefficient functions. Here, we show that the matrix setting described in (2) can be obtained, for instance, when some of the coefficient functions have a small nonzero support in the given domain, both in the separable and nonseparable cases. To the best of our knowledge, the reported treatment in the nonseparable case is new.

Example 6. Separable coefficients. We consider the linear PDE

$$-\Delta u + \omega(x,y)u_y = 1, \quad (x,y) \in \Omega, \tag{15}$$

with $\Omega = (0,1) \times (0,1)$ and

$$\omega(x,y) = \begin{cases} (x+1)(2y+1) & (x,y) \in \Omega_1, \\ 0 & \text{otherwise}, \end{cases} \tag{16}$$

where Ω_1 is a small square centered at $(x,y) = (\frac{1}{2}, \frac{1}{2})$ and $2r_s + 1$ grid points on each side. Finite difference discretization of this problem[9] leads to the linear system

$$AX + XA + M_1X(BN_0) = F,$$

where $A = (n+1)^2\text{tridiag}(-1,2,-1) \in \mathbb{R}^{n\times n}$, $B = \frac{(n+1)}{2}\text{tridiag}(1,0,-1) \in \mathbb{R}^{n\times n}$, while M_1 and N_0 are diagonal matrices containing the nonzero values of the functions $\phi(x) = (x+1), \psi(y) = (2y+1)$, respectively, for $(x,y) \in \Omega_1$, and zero elsewhere, where the grid nodes are ordered lexicographically.[9] Due to the small function support, the only nonzero entries are those corresponding to the ith entries, with $(\frac{n}{2} - r_s \le i \le \frac{n}{2} + r_s)$. The parameter r_s allows us to control the rank of the matrices $M_1, N_1 = BN_0$ in our numerical experiments. Note that the matrices U_1, V_1 in $M_1 = U_1V_1^T$ can be determined at no computational cost, since they correspond to the nonzero columns and scaled rows of the nonzero diagonal elements

TABLE 7 Example 6

		Direct		Matrix form		Vectorized form	
n	**Rank of U_i**	**CPU**	**Res**	**CPU**	**Res**	**CPU**	**Res**
100	5	0.03	1.7e-13	0.02	1.5e-12	0.05	2.7e-13
	7	0.03	1.7e-13	0.02	1.5e-12	0.05	2.7e-13
	9	0.03	1.7e-13	0.02	1.4e-12	0.07	2.7e-13
200	9	0.12	6.6e-13	0.05	7.3e-12	0.23	1.1e-12
	11	0.12	6.7e-13	0.08	7.4e-12	0.34	1.1e-12
	13	0.13	6.6e-13	0.09	7.4e-12	0.43	1.1e-12
300	9	0.30	1.5e-12	0.09	1.3e-11	0.51	2.4e-12
	17	0.30	1.4e-12	0.30	1.3e-11	1.74	2.5e-12
	25	0.30	1.5e-12	0.61	1.3e-11	3.81	2.4e-12

Note: CPU time and residual norms as the problem dimension and the rank of M_1, N_1 vary.

of M_1. The numerical results are reported in Table 7, and show that the matrix-oriented version of the SMW formula is very attractive, when compared with the vectorized one. In spite of the problem high sparsity, the reference (compiled) direct method is not superior to the matrix-oriented method except for the largest dimension and rank.

In the following we consider the case where the coefficient $\omega(x, y)$ is not a separable function. For the sake of the description, we work with the linear PDE $-\Delta u + \omega(x, y)u = f$. The finite difference discretization of the zero-order term $\omega(x, y)u(x, y)$ on a rectangular grid leads to an addend in the matrix equation of the form $C \circ U$, where $\circ$ denotes the elementwise (Hadamard) product. Here, $(C)_{i,j} = \omega(x_i, y_j)$ where (x_i, y_j) are the grid nodes, hence the entries of C are nonzero only at the grid points included in the support of ω. In the vectorized case we would obtain $\text{vec}(C \circ U) = \text{diag}(c)\text{vec}(U)$, where the vector c would contain all values of $\omega(x, y)$ at the nodes. The discretization of the given PDE yields the matrix equation

$$AX + XA^T + C \circ X = F. \tag{17}$$

The inclusion of the uncommon Hadamard product term makes the solution of the matrix equation more challenging; see, for example, Reference 8. Nonetheless, the high sparsity of C allows us to efficiently employ the matrix-oriented SMW formula. Matrix C can be written as $C = GH^T$ with G, H of rank $\ell \leq n$, such that G, H maintain the sparsity of C. For instance, G can collect the nonzero columns of C, while H is a selection of the identity matrix columns corresponding to the nonzero columns of C. Let $G = [g_1, \dots, g_\ell]$ and $H = [h_1, \dots, h_\ell]$, for $\ell \leq n$. Using the following property of the Hadamard product

$$(GH^T) \circ X = \sum_{i=1}^{\ell} \text{diag}(g_i) X \text{diag}(h_i), \tag{18}$$

we can rewrite Equation (17) as

$$AX + XA^T + \sum_{i=1}^{\ell} \text{diag}(g_i) X \text{diag}(h_i) = F.$$

With this definition of G and H, the number of nonzero columns in C gives the number ℓ of summation terms. A different selection strategy for G, H can be considered if C is known to have rank lower than the number of its nonzero columns. From now on we can proceed as it was done in the case of a separable function case, except that $\ell \geq 1$. We observe that in our setting G, H are very sparse, so that $\text{diag}(g_i)$ and $\text{diag}(h_i)$ are diagonal matrices of very low rank. Therefore, setting $\text{diag}(g_i) = U_{L,i} V_{L,i}^T$ and $\text{diag}(h_i) = U_{R,i} V_{R,i}^T$ we have

$$AX + XA^T + \sum_{i=1}^{\ell} U_{L,i} V_{L,i}^T X (U_{R,i} V_{R,i}^T)^T = F. \tag{19}$$

TABLE 8 Example 7 for $\omega_1(x, y)$

		Direct			Matrix form				Vectorized form		
n	**s**	**CPU**	**Res**	**Err**	**dim(*H*)**	**CPU**	**Res**	**Err**	**CPU**	**Res**	**Err**
100	4	0.08	2.1e-16	4.3e-15	16	0.03	2.7e-15	1.7e-13	0.04	3.9e-16	3.3e-14
	6	0.08	2.1e-16	3.1e-15	36	0.04	2.8e-15	1.8e-13	0.05	4.1e-16	3.5e-14
	8	0.08	2.1e-16	4.0e-15	64	0.06	2.9e-15	1.9e-13	0.07	4.1e-16	3.1e-14
	10	0.08	2.1e-16	6.2e-15	100	0.07	2.7e-15	1.9e-13	0.09	4.0e-16	3.1e-14
200	4	0.32	2.1e-16	7.1e-15	16	0.04	3.6e-15	1.3e-12	0.16	4.7e-16	6.9e-13
	6	0.31	2.1e-16	1.2e-14	36	0.06	3.6e-15	1.3e-12	0.20	4.7e-16	7.0e-13
	8	0.30	2.1e-16	3.9e-15	64	0.09	3.6e-15	1.3e-12	0.28	4.7e-16	7.0e-13
	10	0.31	2.1e-16	6.7e-15	100	0.12	3.6e-15	1.3e-12	0.38	4.8e-16	6.9e-13
300	4	0.79	2.0e-16	9.6e-15	16	0.08	3.7e-14	1.5e-12	0.39	4.0e-16	9.4e-14
	6	0.80	2.1e-16	2.2e-14	36	0.12	3.7e-15	1.5e-12	0.50	4.1e-16	1.3e-13
	8	0.77	2.1e-16	1.4e-14	64	0.17	3.8e-15	1.5e-12	0.59	4.1e-16	1.0e-13
	10	0.83	2.1e-16	8.9e-15	100	0.24	3.7e-15	1.4e-12	0.87	4.1e-16	9.3e-14
400	4	1.79	2.1e-16	1.1e-14	16	0.13	4.4e-15	2.8e-12	0.78	4.4e-16	2.3e-12
	6	1.80	2.1e-16	2.8e-14	36	0.21	4.4e-15	2.6e-12	1.00	4.4e-16	2.3e-12
	8	1.68	2.1e-16	1.2e-14	64	0.30	4.4e-15	2.7e-12	1.29	4.4e-16	2.3e-12
	10	1.69	2.1e-16	1.7e-14	100	0.45	4.4e-15	2.6e-12	1.89	4.4e-16	2.3e-12

Note: CPU time, error and residual norms as the problem dimension and the rank vary. dim(H) reports the final rank of the matrix H used in Algorithm 1, step 8.

The new notation $U_{*,i}, V_{*,i}$ is only used here to emphasize the description generality.

In the following we will employ the derivation above to discretize a linear PDE with a convective term, hence the extra factor B will appear, as in Example 6.

Example 7. We consider the same Equation (15) as in Example 6, where this time the derivatives were discretized with a fourth-order finite difference formula, leading to a pentadiagonal structure in A and B, and we employed a different function $\omega(x, y)$. Let Ω_h be a uniform discretization of $\Omega = (0, 1) \times (0, 1)$, with nodes (x_i, y_j), $i, j = 1, \dots, n$, and $h = \frac{1}{n+1}$. Let

$$\Omega_1 = \left[\left(\frac{3n}{4}+1\right)h, \left(\frac{3n}{4}+s\right)h\right] \times \left[\left(\frac{n}{2}+1\right)h, \left(\frac{n}{2}+s\right)h\right], \quad \Omega_2 = [h, sh] \times \left[\left(\frac{n}{4}+1\right)h, \left(\frac{n}{4}+s\right)h\right].$$

We consider two distinct cases for $\omega(x, y)$:

$$\omega_1(x,y) = \begin{cases} (x - \frac{n}{4}h)^2 + y^2 + 2 & (x,y) \in \Omega_1, \\ 0 & \text{otherwise}, \end{cases} \qquad \omega_2(x,y) = \begin{cases} (x - \frac{n}{4}h)^2 + y^2 + 2 & (x,y) \in \Omega_1, \\ x^2 + (y - \frac{n}{4}h)^2 + 2 & (x,y) \in \Omega_2, \\ 0 & \text{otherwise}. \end{cases}$$

Repeating the described derivation in the presence of the Hadamard term, we obtain the following linear matrix equation,

$$AX + XA^T + \sum_{i=1}^{\ell} U_{L,i} V_{L,i}^T X B (U_{R,i} V_{R,i}^T)^T = F. \tag{20}$$

We note that ω_2 yields a larger value of ℓ than ω_1, and thus a larger final rank in the SMW formula, as reported in our tables. The numerical results are listed in Table 8 (for ω_1) and in Table 9 (for ω_2). All results consistently report the good performance of the matrix-oriented version of the SMW formula with respect to the vectorized one, with comparable error and residual norms. The approach also compares rather well with the compiled sparse direct method.

TABLE 9 Example 7 for $\omega_2(x,y)$

		Direct			Matrix form				Vectorized form		
n	**s**	**CPU**	**Res**	**Err**	**dim(*H*)**	**CPU**	**Res**	**Err**	**CPU**	**Res**	**Err**
100	4	0.08	2.1e-16	4.5e-15	32	0.03	2.8e-15	1.8e-13	0.05	4.0e-16	3.2e-14
	6	0.08	2.1e-16	3.4e-15	72	0.06	2.8e-15	1.7e-13	0.08	4.1e-16	3.7e-14
	8	0.08	2.1e-16	2.7e-15	128	0.08	2.7e-15	1.8e-13	0.10	4.0e-16	3.4e-14
	10	0.08	2.1e-16	3.7e-15	200	0.12	2.7e-15	1.8e-13	0.16	4.0e-16	3.7e-14
200	4	0.32	2.1e-16	8.7e-15	32	0.06	3.6e-15	1.3e-12	0.19	4.7e-16	6.9e-13
	6	0.32	2.1e-16	8.5e-15	72	0.10	3.6e-15	1.2e-12	0.30	4.8e-16	6.9e-13
	8	0.32	2.1e-16	1.0e-14	128	0.16	3.6e-15	1.3e-12	0.45	4.7e-16	6.9e-13
	10	0.32	2.1e-16	5.1e-15	200	0.25	3.6e-15	1.3e-12	0.67	4.8e-16	7.0e-13
300	4	0.80	2.0e-16	6.9e-15	32	0.10	3.7e-15	1.4e-12	0.45	4.0e-16	1.1e-13
	6	0.80	2.1e-16	1.4e-14	72	0.17	3.7e-15	1.5e-12	0.73	4.1e-16	1.0e-13
	8	0.81	2.1e-16	1.1e-14	128	0.29	3.7e-15	1.5e-12	1.01	4.1e-16	9.4e-14
	10	0.83	2.1e-16	9.8e-15	200	0.44	3.7e-15	1.6e-12	1.56	4.1e-16	1.0e-13
400	4	1.80	2.1e-16	1.9e-14	32	0.21	4.4e-15	2.8e-12	0.94	4.4e-16	2.2e-12
	6	1.78	2.1e-16	1.1e-14	72	0.34	4.4e-15	2.7e-12	1.45	4.4e-16	2.2e-12
	8	1.80	2.1e-16	1.2e-14	128	0.61	4.4e-15	2.6e-12	2.22	4.4e-16	2.3e-12
	10	1.86	2.1e-16	1.8e-14	200	0.84	4.4e-15	2.7e-12	3.20	4.4e-16	2.3e-12

Note: CPU time, error and residual norms as the problem dimension and the rank vary. dim(H) reports the final rank of the matrix H used in Algorithm 1, step 8.

7 | APPLICATION TO LARGE-SCALE EQUATIONS

In many applications Equation (1) has large dimensions, typically A and M are sparse and large, while F is very low rank. In this setting, the SMW formula may be prohibitively expensive to be used, both in the vectorized and matrix forms, unless $\ell, s_i = \mathcal{O}(1)$, as performed in References 17,18. Indeed, the matrix-oriented formula, or its hybrid version (see Remark 1), require solving at least as many Lyapunov equations as the number of columns of $\mathcal{U}$ in Algorithm 0. So, for instance, if $\ell = 1$ and $s_1 = 15$ (see Example 5), a total of $s^2 = 225$ large-scale Lyapunov equations with nonsymmetric right-hand side need to be solved to compute $H = I + \mathcal{V}^T \mathcal{A}^{-1} \mathcal{U}$. Significantly larger dimensions of H may arise for $\ell > 1$.

In this challenging context, the matrix-oriented approach may still be appealing as a core method within a projection strategy to solve (1), whenever the rank of M is modest, say up to a few tens.

The Galerkin method is a general approximation methodology widely used in several analytical and numerical procedures such as in the variational characterization and discretization of PDEs, in algebraic linear systems and eigenvalue problems, and more recently in linear matrix equations; see Reference 15 for a recent account concerning this last context. In our setting the strategy proceeds as follows. For the sake of the presentation[‡]; see, for example, Reference 15. Assume that F is symmetric and low rank, that is $F = F_1 F_1^T$. Given an approximation space $\mathbb{V} \subset \mathbb{R}^n$ of dimension m_k and a matrix $V_k \in \mathbb{R}^{n \times m_k}$ whose orthonormal columns span $\mathbb{V}$, we are interested in an approximation to X as $X \approx X_k = V_k Y_k V_k^T$ where Y_k is to be determined. The matrix X_k has rank at most m_k and under certain hypotheses on the data, it may be a good approximation to X; see, for example, Reference 5. To determine Y_k the following matrix orthogonality (Galerkin) condition is imposed to the residual matrix $S_k := AX_k + X_k A^T + MX_k M^T - F$,

$$V_k^T S_k V_k = 0,$$

that is, S_k is orthogonal to the approximation space, in the matrix inner product. Substituting S_k in this constraint equation and recalling that $V_k^T V_k = I$ we get

[‡]In the nonsymmetric case, a left and a right projection space may be required, as in the standard Sylvester equation. In the case F is not low rank, low-rank approximations can be considered

$$(V_k^T A V_k)Y_k + Y_k(V_k^T A^T V_k) + (V_k^T M V_k)Y_k(V_k^T M^T V_k) - V_k^T F V_k = 0. \tag{21}$$

The matrices $A_k = (V_k^T A V_k), M_k = (V_k^T M V_k)$ and $F_k = V_k^T F V_k$ have dimensions m_k, so that the new equation is of the same type as the original one, but it has much smaller dimension, as long as $m_k \ll n$. For $s < m_k$, where s is the rank of M, the matrix M_k can again be written as the product of two low-rank matrices[§], and our method can be applied to determine the sought after reduced solution Y_k. If the approximation is not good enough, the approximation space can be enlarged and a new reduced solution Y_k can be obtained. For the existence of Y_k the reduced matrix equation must be solvable. This can be obtained by requiring that the eigenvalues of the coefficient matrix in the Kronecker form of (21) are all contained in one half of the complex plane. This last condition is satisfied if, for instance, the field of values of the coefficient matrix in the Kronecker form of the original problem, that is of the matrix $\mathcal{A} + \mathcal{U}\mathcal{V}^T$, is contained in one half of the complex plane.[15]

To complete the algorithm the approximation space must be selected. Given the expansion property outlined above, a natural choice is to choose a nested space, that is $\mathbb{V}_k \subseteq \mathbb{V}_{k+1}$ where k indicates the iteration, so that the residual matrix is enforced to lay into a smaller and smaller space as k grows. So far the procedure is very general and can be applied to a variety of linear and quadratic matrix equations.[15] However, to the best of our knowledge no Galerkin method has been designed to directly attack the specific problem (1) of large dimension. To make the Galerkin methodology practical, the space selection plays a crucial role.

Let $M = U_1 \hat{V}_1^T$. We propose to consider the column orthogonalization of $V_0 = [F_1, U_1]$ as initial basis. If U_1 is thin, then V_0 has indeed few columns. Letting $V_0 = V_1 R$ be the reduced QR factorization of V_0, the columns of V_1 span $\mathbb{V}_1$. The space is expanded with the two vectors

$$\{(A - \sigma_k I)^{-1} v, (A + M)^{-1} v\}; \tag{22}$$

This choice is in agreement with similar selections for multiterm linear equations in the literature.[10,11] Here, after k iterations, v represents the kth vector in the already generated basis, which spans a space of dimension not smaller than k. Due again to the SMW formula, the vector $(A + M)^{-1}v$ is a linear combination of $A^{-1}v$ and the columns of $A^{-1}U_1$ (computed once for all at the beginning of the algorithm), hence after a number of iterations corresponding to the number of columns of U_1, this second vector in (22) will no longer be needed to expand the space, and only the vector $(A - \sigma_k I)^{-1}v$ will be added. Nonetheless, we found that adding this term in the first few iterations was crucial for the overall convergence. We stress that assuming the spectrum of A lay on one half of the complex plane, the shifts σ_k are computed so as to be on the other half of the complex plane, so as to ensure that $A - \sigma_k I$ is nonsingular. We refer the reader to Reference 15 for a general discussion on Galerkin type procedures for linear matrix equations, of which this is a typical example, and on the shift selection. As opposed to other Galerkin projection methods for multiterm equations, here the space is mainly based on the coefficient matrix A, since the terms involving M are dealt with explicitly.

The stopping criterion is based on the relative residual norm and it is given by

$$\frac{\|S_k\|_{\mathcal{F}}}{\|F\|_{\mathcal{F}}} < \text{tol}.$$

In our experiments we used tol $= 10^{-6}$. The residual matrix should not be explicitly computed as it would require too many memory allocations. We thus adopted a low-rank representation of the residual S_k as it is commonly done in this setting

$$S_k := \begin{bmatrix} AV_k & V_k & MV_k & F_1 \end{bmatrix} \begin{bmatrix} 0 & Y & 0 & 0 \\ Y & 0 & 0 & 0 \\ 0 & 0 & Y & 0 \\ 0 & 0 & 0 & -I \end{bmatrix} \begin{bmatrix} AV_k & V_k & MV_k & F_1 \end{bmatrix}^T =: GPG^T. \tag{23}$$

Let

$$G = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} \hat{R}_1 \\ 0 \end{bmatrix} = Q_1 \hat{R}_1$$

[§]A rank reduction is performed in practice, if M_k is found to be low rank, so as to exploit the matrix-oriented inner solver as soon as possible.

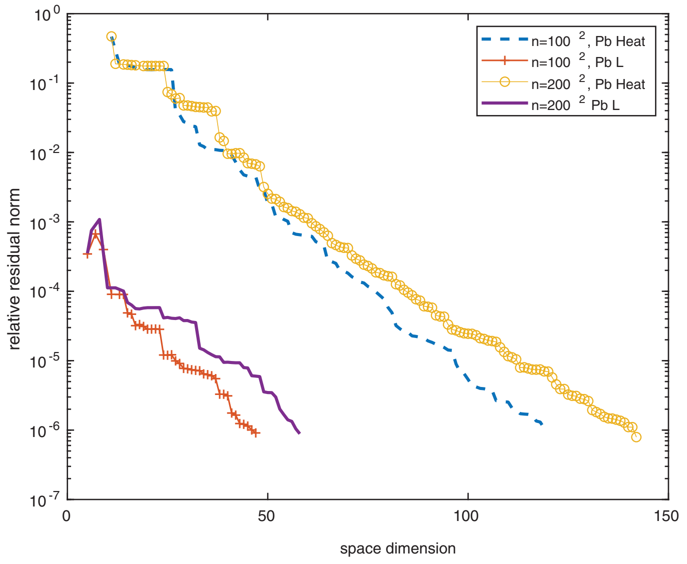

FIGURE 2 Convergence history of the Galerkin procedure for Example 8 ($\alpha = 10$, Pb L) and Example 9 (Pb Heat)

be the QR decomposition of G, then the norm of the residual S_k can be simplified as

$$\|S_k\|_F = \|GPG^T\|_F = \|Q_1\hat{R}_1P\hat{R}_1^TQ_1^T\|_F = \|\hat{R}_1P\hat{R}_1^T\|_F. \tag{24}$$

The QR decomposition is not computed from scratch at each iteration, but updated as new basis columns are included.

Remark 2. Although the derivation of this section is focused on the symmetric case, the procedure can be easily generalized to the form (2) with matrices of large dimensions. The reduced problem to be solved will use the general approach described in Section 3.

In the following, we report on two sets of numerical experiments we have performed. We consider symmetric matrices, for which Algorithm 1 has shown its greatest effectiveness. We also compare with a state-of-the-art method, GLEK in Reference 7, which is appropriate for any sparse M, not necessarily of very low rank. Being based on a matrix splitting condition, it may have convergence problems on data that do not satisfy a certain matrix norm requirement; see Reference [7, th. 3.1]. GLEK was shown to be competitive with respect to other methods for the same class of problems in Reference 7.

Example 8. We consider an academic example ¶ where A corresponds to the finite difference discretization of the operator $\mathcal{L}(u) = +(\exp(-xy)u_x)_x + (\exp(xy)u_y)_y$ and $M_1 = -U_1U_1^T$ in $(0,1)\times(0,1)$, with $U_1 = \alpha[U_{11};0]$, and $U_{11} = [\mathbf{1}\otimes I_4]$, $\mathbf{1}$ is the vector of all ones of length 10, so that U_1 has four columns having a sequence of ones at different locations, and it has orthogonal columns. Here, F_1 is a vector with random entries uniformly distributed in $(0,1)$. The convergence history of the Galerkin approach as the subspace dimension grows is reported in Figure 2 for $n = 100^2$ and $n = 200^2$. We notice that the convergence is barely sensitive to the problem dimension, as it has been observed in other application problems for rational Krylov subspaces. More details on the performance of the method are reported in Table 10, together with a comparison with the method GLEK in Reference 7, as α grows. For the large values of α the Lyapunov operator becomes less dominant, and the splitting-based method GLEK has convergence problems. The projection method converges for all values of α, though its computational costs are higher# than GLEK, whenever the latter converges. Note also that the performance of the new solver improves with α: for the larger values of α this seems to be due to the fact that the more dominant part is the portion of the problem that is solved explicitly by including U_1 in the space.

Example 9. Heat equation from Reference 16. We consider the heat equation on the unit square, so that A is the discretization of the Laplace operator $L(\boldsymbol{x}) = \Delta\boldsymbol{x}$ in $\Omega = (0,1)\times(0,1)$. Robin conditions $\boldsymbol{n}\cdot\nabla(\boldsymbol{x}) = \frac{1}{2}d\cdot(\boldsymbol{x}-1)$ are used on a small portion of the left domain boundary (corresponding to 10 nodes, so that U_1 has 10 columns), while Dirichlet conditions $x=0$ are used on the rest of the boundary, yielding a single matrix $N_1 (m=1)$. The parameter d represents the

¶The numerical experiments for this example were ran on a different computer (a DELL Latitude with Core i7) than for the other examples.
#In a performance-oriented implementation, computational costs can be lowered by computing the reduced problem solution and checking convergence only periodically (say every 5 iterations).

TABLE 10 Example 8

		GALERKIN			GLEK	
n	α	# iter	space dim	CPU time	space dim	CPU time
10,000	1	70	78	4.64	79	0.85
40,000	1	105	109	31.46	86	3.30
10,000	2.5	61	69	3.34	103	2.53
40,000	2.5	73	81	22.79	93	4.87
10,000	5	48	56	2.56	–	–
40,000	5	57	65	16.07	–	–
10,000	10	37	45	1.79	–	–
40,000	10	49	57	15.50	–	–

Note: Numerical comparisons of the Galerkin projection and GLEK.

TABLE 11 Example 9

			GALERKIN			GLEK		
n	d	rank(U_1)	# iter	space dim	CPU time	# iter	space dim	CPU time
2500	0.5	6	66	78	1.39	6	95	0.61
	0.8	6	79	91	1.42	11	106	0.95
	1.0	6	76	88	1.25	37	116	4.11
10,000	0.5	10	106	120	6.95	6	128	1.41
	0.8	10	107	124	7.30	14	154	4.86
	1.0	10	154	171	14.04	–	–	–
40,000	0.5	10	130	142	29.75	6	159	6.39
	0.8	10	141	154	34.80	16	211	30.68
	1.0	10	171	185	49.55	–	–	–

Note: Numerical comparisons of the Galerkin projection and GLEK.

control input and is problem dependent. This example was also reported in Reference [7, example 5.1], with a larger rank U_1. The numerical results are listed in Table 11 for $d \in \{0.5, 0.8, 1\}$. As long as the Lyapunov operator remains dominant, the method GLEK is superior to the Galerkin approach in terms of CPU time, while memory requirements are slightly higher than with the projection method. As soon as the term involving the Robin boundary conditions becomes dominant, problems for the splitting-based method arise, like in the previous example.

We conclude with a comment on this example, illustrating our Remark 1. The solution of each single Lyapunov equation $AP + PA^T = u_i u_j^T$ for $n = 40{,}000$ and $d = 0.5$ requires about 3.3 s (2.6 s) to reach a relative residual norm less than 10^{-7} in only 15 iterations (26 iterations), using the adaptive rational Krylov subspace method[38] (using KPIK with a prefactorized A[39]). If one were to use Algorithm 1 directly on the problem, the total time to approximate each of the 100 columns of $\mathcal{AU}$ would be largely over 200 s. A more advanced implementation might be able to reduce these large timings and make the direct use of the SMW matrix-oriented approach directly applicable to the large-scale problem for $\ell > 1$.

8 | APPLICATION TO LINEAR TENSOR EQUATIONS

Consider the tensor equation in the X tensor variable

$$(M_1 \otimes A_1 \otimes \boldsymbol{H} + A_2 \otimes M_2 \otimes \boldsymbol{H} + H_3 \otimes M_3 \otimes A_3)\mathrm{vec}(X) = b_3 \otimes b_2 \otimes b_1 \tag{25}$$

with $\boldsymbol{H}, H_3, M_3$ nonsingular, and A_1, M_1 low rank. Here, the matrix $\boldsymbol{H}$ is in bold to emphasize that it is the same matrix appearing in two different terms.

This tensor equation is representative of a large class of problems that can be described by means of tensors. This is the case for instance for discretized three-dimensional PDEs when a tensor basis is used for the discretization; see, for example, Reference [9, section 5] and references therein. More generally, tensor equations have become a key ingredient in the numerical treatment of mathematical models dealing with uncertainty quantification and of parameter-dependent model order reduction methodologies; the literature is quite vast, nonetheless we refer, for example, to References 40–47 and their references.

The analysis of tensors and the development of numerical methods have recently generated a large amount of literature. Different tensor representations have been analyzed, as discussed for instance in Reference 48; we also refer the reader to the literature survey in Reference 49. In most cases, the authors have been interested in numerical methods dealing with the presence of many summands and many Kronecker products in a sparse context, leading necessarily to iterative approaches. Nonetheless, available explicit methods deriving the solution in some closed form with low memory requirements are very scarce even for few summands. Here, we aim to contribute to filling this gap; we refer to Reference 50 for another contribution towards this aim for a tensor equation with different properties.

The following result yields a new procedure for computing the solution tensor X to Equation (25). To the best of our knowledge, this is the first method that explicitly determines the solution in closed form, without using the Kronecker form of the problem.

We recall that a tensor $X \in \mathbb{R}^{n_1\times n_2\times n_3}$ can be written using the mode-1 unfolding as (see, e.g., Reference 48).

$$X_{(1)} = [X_1, X_2, \dots, X_{n_3}], \quad X_j \in \mathbb{R}^{n_1\times n_2}, j = 1, \dots, n_3;$$

The proof of this result is closely related to a proof that has been recently given for a slightly different class of three-mode tensors in Reference 50.

Proposition 1. *Let $(\boldsymbol{H}^{-1}A_3)^T = QRQ^*$ be the Schur decomposition of the given $n_1\times n_1$ matrix. Then the solution can be obtained as $X_{(1)} = ([\hat{z}_1, \dots, \hat{z}_{n_1}]Q^*)^T$ where $\hat{z}_j = \mathrm{vec}(\hat{Z}_j)$ are such that for $j=1$, the matrix $\hat{Z}_1$ solves*

$$M_3^{-1}A_1\hat{Z}_1(H_3^{-1}M_1)^T + \hat{Z}_1R_{1,1} + M_3^{-1}M_2\hat{Z}_1(H_3^{-1}A_2)^T = M_3^{-1}b_2\gamma_1(H_3^{-1}b_3)^T,$$

while for $j=2, \dots, n_1$, the $n_2\times n_3$ matrix $\hat{Z}_j$ solves

$$M_3^{-1}A_1\hat{Z}_j(H_3^{-1}M_1)^T + \hat{Z}_jR_{j,j} + M_3^{-1}M_2\hat{Z}_j(H_3^{-1}A_2)^T = M_3^{-1}b_2\gamma_j(H_3^{-1}b_3)^T - W_{j-1},$$

where $W_{j-1} = \mathrm{mat}([\hat{z}_1, \dots, \hat{z}_{j-1}]R_{1:j-1,j})$ and $[\gamma_1, \dots, \gamma_{n_1}] = b_1^T\boldsymbol{H}^{-T}Q$.

Proof. For the unfolded tensor we have

$$\begin{aligned}
\boldsymbol{H}X_{(1)}(A_2\otimes M_2 + M_1\otimes A_1)^T + A_3X_{(1)}(H_3\otimes M_3)^T &= b_1(b_3\otimes b_2)^T\\
X_{(1)}(A_2\otimes M_2 + M_1\otimes A_1)^T(H_3\otimes M_3)^{-T} + \boldsymbol{H}^{-1}A_3X_{(1)} &= \boldsymbol{H}^{-1}b_1(b_3\otimes b_2)^T(H_3\otimes M_3)^{-T}\\
X_{(1)}(A_2^TH_3^{-T}\otimes M_2^TM_3^{-T} + M_1^TH_3^{-T}\otimes A_1^TM_3^{-T}) + \boldsymbol{H}^{-1}A_3X_{(1)} &= \boldsymbol{H}^{-1}b_1(b_3^TH_3^{-T}\otimes b_2^TM_3^{-T})
\end{aligned}$$

where we have used the properties of Kronecker product inverses and transpositions. Let us transpose both sides of the last equation and set $Y = (X_{(1)})^T$. Then

$$(H_3^{-1}A_2\otimes M_3^{-1}M_2 + H_3^{-1}M_1\otimes M_3^{-1}A_1)Y + Y(\boldsymbol{H}^{-1}A_3)^T = (H_3^{-1}b_3\otimes M_3^{-1}b_2)b_1^T\boldsymbol{H}^{-T}.$$

Using $(\boldsymbol{H}^{-1}A_3)^T = QRQ^*$ and multiplying the equation by Q from the right, we can write

$$(H_3^{-1}A_2\otimes M_3^{-1}M_2 + H_3^{-1}M_1\otimes M_3^{-1}A_1)YQ + YQR = (H_3^{-1}b_3\otimes M_3^{-1}b_2)b_1^T\boldsymbol{H}^{-T}Q.$$

Let $b_1^T\boldsymbol{H}^{-T}Q =: [\gamma_1, \dots, \gamma_{n_1}]$ and $YQ =: [\hat{z}_1, \dots, \hat{z}_{n_1}]$. Thanks to the upper triangular form of R, for the first column $\hat{z}_1$ it holds

$$(H_3^{-1}A_2\otimes M_3^{-1}M_2 + H_3^{-1}M_1\otimes M_3^{-1}A_1)\hat{z}_1 + \hat{z}_1R_{1,1} = (H_3^{-1}b_3\otimes M_3^{-1}b_2)\gamma_1. \quad (26)$$

For the subsequent columns $j = 2, \dots, n_1$, taking into account once again the triangular form of R, we set $w_{j-1} = [\hat{z}_1, \dots, \hat{z}_{j-1}]R_{1:j-1,j}$ so that

$$(H_3^{-1}A_2 \otimes M_3^{-1}M_2 + H_3^{-1}M_1 \otimes M_3^{-1}A_1)\hat{z}_j + \hat{z}_j R_{j,j} = (H_3^{-1}b_3 \otimes M_3^{-1}b_2)\gamma_j - w_{j-1}. \tag{27}$$

Each column can be obtained in sequence by further unmaking the Kronecker product as follows. Let us reshape each $\hat{z}_j$ so that $\hat{z}_j = \mathrm{vec}(\hat{Z}_j)$. Using (26) for $j = 1$, we can write

$$M_3^{-1}A_1\hat{Z}_1(H_3^{-1}M_1)^T + \hat{Z}_1 R_{1,1} + M_3^{-1}M_2\hat{Z}_1(H_3^{-1}A_2)^T = M_3^{-1}b_2\gamma_1(H_3^{-1}b_3)^T,$$

giving the first matrix equation. Analogously, for $j = 2, \dots, n_1$ and letting $W_{j-1} = \mathrm{mat}([\hat{z}_1, \dots, \hat{z}_{j-1}]R_{1:j-1,j})$, from (27) we obtain

$$M_3^{-1}A_1\hat{Z}_j(H_3^{-1}M_1)^T + \hat{Z}_j R_{j,j} + M_3^{-1}M_2\hat{Z}_j(H_3^{-1}A_2)^T = M_3^{-1}b_2\gamma_j(H_3^{-1}b_3)^T - W_{j-1},$$

with once again the desired form. ∎

The two matrix equations in $\hat{Z}_j$ in the proposition statement already have a form that can exploit our setting since A_1 and M_1 have low rank. If either M_2 or A_2 is nonsingular, then the equation can be put into the "canonical form" in (2) with $\ell = 1$ as a Sylvester (rather than Lyapunov) operator plus a low-rank term. Indeed, for the sake of the description let us take the first of the two equations[‖] in Proposition 1,

$$M_3^{-1}A_1\hat{Z}_1(H_3^{-1}M_1)^T + \hat{Z}_1 R_{1,1} + M_3^{-1}M_2\hat{Z}_1(H_3^{-1}A_2)^T = M_3^{-1}b_2\gamma_1(H_3^{-1}b_3)^T,$$

and assume that $M_3^{-1}M_2$ is invertible. Multiplying from the left by the inverse of this matrix we obtain

$$M_2^{-1}A_1\hat{Z}_1(H_3^{-1}M_1)^T + R_{1,1}(M_3^{-1}M_2)^{-1}\hat{Z}_1 + \hat{Z}_1(H_3^{-1}A_2)^T = M_2^{-1}b_2\gamma_1(H_3^{-1}b_3)^T,$$

which, after reordering the terms, is in the form

$$\boldsymbol{A}\hat{Z}_1 + \hat{Z}_1\boldsymbol{B} + U_1V_1^T\hat{Z}_1(U_2V_2^T)^T = F.$$

Here, $U_1V_1^T$ is the low-rank form of $M_2^{-1}A_1$, obtained for instance by an SVD; in the same way, $U_2V_2^T$ is the low-rank form of $H_3^{-1}M_1$. The matrices $\boldsymbol{A}, \boldsymbol{B}$ are readily defined.

The complete procedure is summarized in Algorithm 2.

Algorithm 2. Sherman–Morrison–Woodbury formula for tensor equations

```
1: INPUT: $\boldsymbol{H}, A_3 \in \mathbb{R}^{n_1\times n_1}, A_1, M_2, M_3 \in \mathbb{R}^{n_2\times n_2}, A_2, M_1, H_3 \in \mathbb{R}^{n_3\times n_3}, b_1 \in \mathbb{R}^{n_1}, b_2 \in \mathbb{R}^{n_2}, b_3 \in \mathbb{R}^{n_3}$
2: OUTPUT: Solution matrix $X_{(1)}$
3: Do Schur decomposition $(\boldsymbol{H}^{-1}A_3)^T := QRQ^*$
4: Compute $b_1^T\boldsymbol{H}^{-T}Q := [\gamma_1, \dots, \gamma_{n_1}]$
5: Solve $R_{1,1}M_2^{-1}M_3Z_1 + Z_1(H_3^{-1}A_2)^T + M_2^{-1}A_1Z_1(H_3^{-1}M_1)^T = M_2^{-1}b_2\gamma_1(H_3^{-1}b_3)^T$ for $Z_1$ by Algorithm 1
6: For $j = 2, \dots, n_1$ solve $R_{j,j}M_2^{-1}M_3Z_j + Z_j(H_3^{-1}A_2)^T + M_2^{-1}A_1Z_j(H_3^{-1}M_1)^T = M_2^{-1}b_2\gamma_j(H_3^{-1}b_3)^T - \sum_{i=1}^{j-1} R_{i,j}M_2^{-1}M_3Z_i$ for $Z_j$
   by Algorithm 1
7: Compute $X_{(1)} = \left([\mathrm{vec}(Z_1), \dots, \mathrm{vec}(Z_{n_1})]Q^*\right)^T$
```

Example 10. We consider the problem (25) with $n_1 = n_2 = n_3 = n$, and $A_1 = K_1K_2^T, M_1 = K_3K_4^T$ with K_1, K_2 and K_3, K_4 random matrices with s_1 and s_2 columns, respectively. All other matrices are random matrices (all with elements uniformly

[‖]We stress that equations of the form $A_1XB_1 + A_2XB_2 + \alpha X = F$ with $\alpha \neq 0$ are not solvable by the Schur decomposition-based approaches employed for $\alpha = 0$, hence our approach becomes very relevant.

TABLE 12 Example 10

		Vectorized form		Matrix Form	
n	s_1/s_2	CPU time	Res	CPU time	Res
20	2/3	2.41	3.485319e-11	0.03	1.102097e-12
	3/5	2.46	6.372322e-11	0.06	3.401809e-12
40	2/3	–	–	0.13	2.037035e-11
	3/5	–	–	0.25	5.789128e-11
80	2/3	–	–	0.89	7.387216e-10
	3/5	–	–	1.83	4.095868e-10
160	2/3	–	–	5.87	1.016724e-09
	3/5	–	–	11.94	5.948960e-10

Note: CPU time and residual norms for solving the tensor equation (25) as dimension and ranks vary.

distributed in $(0,1)$). The numerical results for increasing n and s_i, $i=1,2$ are reported in Table 12, where

$$\text{Res} = \frac{\|((A_2 \otimes M_2 + M_1 \otimes A_1) \otimes \boldsymbol{H} + (H_3 \otimes M_3) \otimes A_3)x - (b_3 \otimes b_2) \otimes b_1\|_2}{\|(b_3 \otimes b_2) \otimes b_1\|_2}.$$

The cost and residual when solving with Matlab backslash are also reported. Due to the high density of the problem, the direct method could only be used for the smallest dimensional problem. On the other hand, the new approach can solve the problem in a few seconds of CPU time, even in the largest considered case.

Under certain hypotheses on the data, the case of the tensor equation with more than three terms can be treated similarly. As an example, we report the result for the equation with four terms.

Corollary 1. *Consider the tensor equation*

$$(M_1 \otimes A_1 \otimes \boldsymbol{H} + A_2 \otimes M_2 \otimes \boldsymbol{H} + H_3 \otimes M_3 \otimes A_3 + A_4 \otimes M_4 \otimes \boldsymbol{H})\text{vec}(X) = b_3 \otimes b_2 \otimes b_1 \tag{28}$$

and let $X_{(1)} \in \mathbb{R}^{n_1 \times n_2 n_3}$ be its mode-1 unfolding solution. Let $(\boldsymbol{H}^{-1}A_3)^T = QRQ^*$ be the Schur decomposition of the given matrix. Then $X_{(1)} = ([\hat{z}_1, \dots, \hat{z}_{n_1}]Q^*)^T$, where $\hat{z}_j = \text{vec}(\hat{Z}_j)$ and $\hat{Z}_j$ solves the four term equations:

(i) *For $j=1$,*

$$A_1\tilde{Z}_1M_1^T + M_4\tilde{Z}_1A_4^T + M_2\tilde{Z}_1A_2^T + M_3R_{1,1}\tilde{Z}_1H_3^T = b_2\gamma_1b_3^T;$$

(ii) *For $j=2, \dots, n_1$,*

$$A_1\tilde{Z}_jM_1^T + M_4\tilde{Z}_jA_4^T + M_2\tilde{Z}_jA_2^T + M_3R_{j,j}\tilde{Z}_jH_3^T = b_2\gamma_jb_3^T - \tilde{W}_{j-1},$$

where $\tilde{W}_{j-1} = M_3\text{mat}([\tilde{z}_1, \dots, \tilde{z}_{j-1}]R_{1:j-1,j})H_3^T$ and $[\gamma_1, \dots, \gamma_{n_1}] = b_1^T\boldsymbol{H}^{-T}Q$.

Proof. The proof proceeds as that of the previous proposition, with the extra term $A_4 \otimes M_4 \otimes \boldsymbol{H}$ carried over. ■

Each of the two matrix equations in Corollary 1 contains four terms. However, two of these terms can be collected together under certain hypotheses on the data. For instance, if for instance $A_4 = H_3$, then the first matrix equation becomes

$$A_1\tilde{Z}_1M_1^T + (M_4 + M_3R_{1,1})\tilde{Z}_1A_4^T + M_2\tilde{Z}_1A_2^T = b_2\gamma_1b_3^T; \tag{29}$$

Our methodology still applies to this generalized Sylvester equation once the canonical form can be recovered; this is the case, for instance, for nonsingular M_2 and A_4. In addition, if A_1, M_1 are low rank, then the SMW formula can be employed.

TABLE 13 Example 11

		Direct method		Matrix form	
n	**Rank(U_i)**	**CPU**	**Res**	**CPU**	**Res**
31	3	0.2	4.6e-14	0.02	6.5e-14
47	4	1.5	8.5e-14	0.04	1.8e-13
63	5	7.9	1.9e-13	0.11	3.3e-13
79	6	34.8	2.6e-13	0.25	6.0e-13
95	7	172.0	3.6e-13	0.76	8.3e-13

Note: CPU time and residual norms for solving the tensor equation (31) as dimension and rank vary.

Example 11. *Three-dimensional linear PDEs.* We consider the PDE

$$-\Delta u + \omega(x,y,z)u = 1, \quad (x,y,z) \in \Omega, \tag{30}$$

with homogeneous Dirichlet boundary conditions and $\Omega = (0,1)^3$. We assume that $\omega = \omega(x,y,z) = \omega(x,y)$ is defined similarly to Example 15, with $\Omega_1 = [\frac{3}{16}, \frac{1}{4}] \times [\frac{3}{16}, \frac{1}{4}] \times (0,1)$. Finite difference discretization gives the tensor problem

$$(M_1 \otimes A_1 \otimes I + T \otimes I \otimes I + I \otimes T \otimes I + I \otimes I \otimes T)\text{vec}(X) = \mathbf{1} \otimes \mathbf{1} \otimes \mathbf{1}, \tag{31}$$

where $T = (n+1)^2\text{tridiag}(-1,2,-1)$ is again the $n \times n$ 3-point stencil discretization of the second-order operator in one dimension, and I is the identity matrix of conforming size. In addition, M_1, A_1 are diagonal matrices with nonzero diagonal entries only corresponding to the support grid values associated with Ω_1 of the two functions $\phi(x), \psi(y)$, respectively. Using the formulation in (29), for $j = 1$ the equation for the transformed solution $\tilde{Z}_1$ can be written as

$$A_1\tilde{Z}_1 M_1^T + (T + R_{1,1}I)\tilde{Z}_1 + \tilde{Z}_1 T = \mathbf{1}\gamma_1\mathbf{1}^T, \tag{32}$$

whose structure naturally adapts to our setting. For $j > 1$ the equation changes accordingly. The numerical results are reported in Table 13 and illustrate the potential of the matrix approach for solving the tensor problem, compared with the use of the sparse direct solver applied to the Kronecker version of the original tensor equation. Equation (32) could also be solved with methods other than the matrix-oriented SMW formula. Comparisons similar to those reported in previous sections would arise, and we refrain from repeating this whole discussion. Here, we focus on the major advantages obtained by unfolding the tensor problem.

9 | CONCLUSIONS

We have analyzed in detail a fully matrix-oriented procedure for solving general small and medium scale multiterm linear matrix equations, when some of the terms have low rank. The procedure relies on a matrix-oriented use of the classical vector SMW formula, while avoiding the generation of the large dense matrices necessary to deal with the original formula. By means of a variety of small and medium size application problems, we have shown that the matrix-oriented method can achieve order of magnitude CPU time improvements over the vectorized method, and can solve in a few seconds dense problems that cannot be treated by the vectorized SMW formula.

We have shown how to employ this procedure in the large-scale case as the reduced equation solver within a Galerkin projection method. Moreover, we have illustrated that the new procedure is a crucial ingredient for a new effective method that explicitly solves a class of linear tensor equations.

ACKNOWLEDGMENTS

We would like to thank two anonymous reviewers for their many insightful comments and for pointing to important references. Moreover, we thank Tobias Damm for helpful discussions and Leonardo Robol for sharing the Matlab code associated with Reference 17. The first author is funded by the China Scholarship Council (Contract No. 201906180033) and by the National Natural Science Foundation of China (Grant Nos. 11471150). Her work was performed during her visit at the Università di Bologna, Italy. The second author is a member of Indam-GNCS. Its support is gratefully acknowledged. Part of this work was also supported by the Grant AlmaIdea 2017–2020 - Università di Bologna.

CONFLICT OF INTEREST

The authors declare no potential conflict of interests.

ORCID

Valeria Simoncini https://orcid.org/0000-0003-0795-5865

REFERENCES

1. Lancaster P. Explicit solutions of linear matrix equations. SIAM Rev. 1970;12(4):544–66.
2. Konstantinov M, Gu D, Mehrmann V, Petkov P. Perturbation theory for matrix equations. Studies in Computational Mathematics. Vol 9. Amsterdam: Elsevier; 2003.
3. Benner P, Damm T. Lyapunov equations, energy functionals, and model order reduction of bilinear and stochastic systems. SIAM J Control Optim. 2011;49(2):686–711.
4. Gray WS, Mesko J. Energy functions and algebraic gramians for bilinear systems. IFAC Proc Vols. 1998;31(17):101–6.
5. Benner P, Breiten T. Low rank methods for a class of generalized Lyapunov equations and related issues. Numer Math. 2013;124(3):441–70.
6. Breiten T, Damm T. Krylov subspace methods for model order reduction of bilinear control systems. Syst Control Lett. 2010;59(8):443–50.
7. Shank SD, Simoncini V, Szyld DB. Efficient low-rank solutions of generalized Lyapunov equations. Numer Math. 2016;134(2):327–42.
8. Hao Y, Simoncini V. Matrix equation solving of PDEs on regular domains. J Numer Math. 2020. https://hal.archives-ouvertes.fr/hal-02902456.
9. Palitta D, Simoncini V. Matrix-equation-based strategies for convection-diffusion equations. BIT Numer Math. 2016;56:751–76.
10. Powell CE, Silvester D, Simoncini V. An efficient reduced basis solver for stochastic Galerkin matrix equations. SIAM J Sci Comput. 2017;39(1):A141–63.
11. Buenger A, Simoncini V, Stoll M. A low-rank matrix equation method for solving PDE-constrained optimization problems; 2020. arXiv preprint arXiv:2005.14499.
12. Bouhamidi A, Jbilou K. A note on the numerical approximate solutions for generalized Sylvester matrix equations with applications. Appl Math Comp. 2008;206:687–94.
13. Chehab JP, Raydan M. An implicit preconditioning strategy for large-scale generalized Sylvester equations. Appl Math Comput. 2011;217(21):8793–803.
14. Kressner D, Sirković P. Truncated low-rank methods for solving general linear matrix equations. Numer Linear Algebra Appl. 2015;22(3):564–83. https://doi.org/10.1002/nla.1973.
15. Simoncini V. Computational methods for linear matrix equations. SIAM Rev. 2016;58(3):377–441.
16. Damm T. Direct methods and ADI-preconditioned Krylov subspace methods for generalized Lyapunov equations. Num Lin Alg Appl. 2008;15:853–71.
17. Massei S, Palitta D, Robol L. Solving rank-structured sylvester and Lyapunov equations. SIAM J Matrix Anal Appl. 2018;39(4):1564–90. https://doi.org/10.1137/17M1157155.
18. Ringh E, Mele G, Karlsson J, Jarlebring E. Sylvester-based preconditioning for the waveguide eigenvalue problem. Linear Algebra Appl. 2018;542:441–63.
19. Golub G, Van Loan CF. Matrix computations. 4th ed. Baltimore: The Johns Hopkins University Press; 2013.
20. Henderson HV, Searle SR. On deriving the inverse of a sum of matrices. SIAM Rev. 1981;23(1):53–60.
21. Sherman J, Morrison WJ. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. Ann Math Stat. 1950;21:124–7. https://doi.org/10.1214/aoms/1177729959.
22. Hager W. Updating the inverse of a matrix. SIAM Rev. 1989;31(2):221–39.
23. Malyshev A, Sadkane M. Using the Sherman-Morrison-Woodbury inversion formula for a fast solution of tridiagonal block Toeplitz systems. Linear Algebra Appl. 2011;435:2693–2707.
24. Bru R, Cerdán J, Marín J, Mas J. Preconditioning sparse nonsymmetric linear systems with the Sherman–Morrison formula. Appl Math. 2003;11(25):701–15.
25. Li R, Saad Y. Divide and conquer low-rank preconditioners for symmetric matrices. SIAM J Sci Comput. 2013;35(4):A2069–95.
26. Nocedal J, Wright S. Numerical optimization. New York, NY: Springer; 1999.
27. Bunch JR, Rose DJ. Partitioning, tearing and modification of sparse linear systems. J Math Appl. 1974;48:574–93.
28. Duff IS, Erisman AM, Reid JK. Direct methods for sparse matrices. Oxford, UK: Clarendon Press; 1989.
29. Chan TF, Saad Y. Iterative methods for solving bordered systems with applications to continuation methods. SIAM J Sci Comput. 1985;6(2):438–51.
30. Yip EL. A note on the stability of solving a rank-p modification of a linear system by the Sherman-Morrison-Woodbury formula. SIAM J Sci Stat Comput. 1986;7(2):507–13.
31. Richter S, Davis LD, Collins EG. Efficient computation of the solutions to modified Lyapunov equations. SIAM J Matrix Anal Appl. 1993 Apr;14(2):420–31.

32. Collins EG, Hodel S. Efficient solution of linearly coupled Lyapunov equations. SIAM J Matrix Anal Appl. 1997 Apr;18(2): 291–304.
33. Kuzmanović I, Truhar N. Sherman-Morrison-Woodbury formula for Sylvester and T-Sylvester equations with applications. Int J Comput Math. 2013;90(2):306–24.
34. Bartels RH, Stewart GW. Algorithm 432: solution of the matrix equation $AX + XB = C$. Commun ACM. 1972;15(9):820–6.
35. Jonsson I, Kågström B. Recursive blocked algorithms for solving triangular systems – Part I: one-sided and coupled Sylvester-type matrix equations. ACM Trans Math Softw. 2002;28(4):392–415.
36. MATLAB 7; 2017.
37. Higham NJ. Accuracy and stability of numerical algorithms. Philadelphia, PA: Society for Industrial and Applied Mathematics; 1996. http://www.ma.man.ac.uk/~higham/asna.html.
38. Druskin V, Simoncini V. Adaptive rational Krylov subspaces for large-scale dynamical systems. Syst Control Lett. 2011;60:546–60.
39. Simoncini V. A new iterative method for solving large-scale Lyapunov matrix equations. SIAM J Sci Comput. 2007;29(3):1268–88.
40. Gavrilyuk I, Khoromskij BN. Tensor numerical methods: actual theory and recent applications [editorial]. Comput Methods Appl Math. 2019;19(1):1–4. https://doi.org/10.1515/cmam-2018-0014.
41. Hackbusch W, Khoromskij BN, Tyrtyshnikov EE. Hierarchical Kronecker tensor-product approximations. J Numer Math. 2005;13(2):119–56.
42. Khoromskij BN. Tensors-structured numerical methods in scientific computing: survey on recent advances. Chemom Intell Lab Syst. 2012;110:1–19.
43. Kressner D, Kumar R, Nobile F, Tobler C. Low-rank tensor approximation for high-order correlation functions of Gaussian random fields. SIAM/ASA J Uncertain Quant. 2015;3:393–416.
44. Oseledets IV. Tensor-train decomposition. SIAM J Sci Comput. 2011;33(5):2295–317.
45. Dolgov SV, Khoromskij BN, Oseledets IV. Fast solution of parabolic problems in the tensor train/quantized tensor train format with initial application to the Fokker–Planck equation. SIAM J Sci Comput. 2013;34(6):A3016–38.
46. Khoromskij BN, Schwab C. Tensor-structured Galerkin approximation of parametric and stochastic elliptic PDEs. SIAM J Sci Comput. 2011;33(1):364–85.
47. Kressner D, Tobler C. Low-rank tensor Krylov subspace methods for parametrized linear systems. SIAM J Matrix Anal Appl. 2011;32(4):1288–316.
48. Kolda TG, Bader BW. Tensor decompositions and applications. SIAM Rev. 2009;51:455–500.
49. Grasedyck L, Kressner D, Tobler C. A literature survey of low-rank tensor approximation techniques. GAMM-Mitteilungen. 2013;36(1):53–78.
50. Simoncini V. Numerical solution of a class of third order tensor linear equations. Boll UMI. 2020;13(3):429–39.

How to cite this article: Hao Y, Simoncini V. The Sherman–Morrison–Woodbury formula for generalized linear matrix equations and applications. *Numer Linear Algebra Appl*. 2021;28:e2384. https://doi.org/10.1002/nla.2384

APPENDIX A

In this section, we report the Matlab implementation of the algorithm for solving (2) with dense data of modest dimensions and $\ell = 2$. The implementation for $\ell > 2$ follows exactly the same lines.

```
function [X] = SMW_matrix_general(U1,V1,U2,V2,U3,V3,U4,V4,A,F)
%function [X] = SMW_matrix_general(U1,V1,U2,V2,U3,V3,U4,V4,A,F)
% Solve AX + XA' + M1 X M2' + M3 X M4' = F,
% where M_i=U_i*V_i' and U_i, V_i are low rank s_i,
% via the Sherman--Morrison--Woodbury matrix-oriented formula
%
symmA=(norm(A-A',1)<1e-12);
nonmulti=isempty(U3);

n=size(A,1);
s1=size(U1,2); s2=size(U2,2);
s3=size(U3,2); s4=size(U4,2);
```

```
if (symmA)
     [Q,R]=eig(A); L=diag(R)*ones(1,n)+ones(n,1)*diag(R).';
     rhs=(Q'*F*Q)./L;
else
     [Q,R]=schur(A,'real');
     rhs=lyap(R,-Q'*F*Q);
end

p=0;
% Change of basis
U1Q=Q'*U1; U2Q=Q'*U2;
V1Q=Q'*V1; V2Q=Q'*V2;

if (~nonmulti) % extra low-rank term
     U3Q=Q'*U3; U4Q=Q'*U4;
     V3Q=Q'*V3; V4Q=Q'*V4;
end

% computed blocks of H associated with M1,M2
for k=1:s2
     if (symmA), G=U2Q(1:n,k)'./L;end % replica of the row U2Q((:,k)'
     for i=1:s1
         p=p+1;
         if (symmA)
              VV1(1:n,1:n,p)=U1Q(1:n,i).*G; % replica of column U1Q(:,i) (*)
         else
              VV1(1:n,1:n,p)=lyap(R,-U1Q(:,i)*U2Q(:,k)');
         end

         vk=V1Q'*(VV1(1:n,1:n,p)*V2Q);
         H(1:s1*s2,p)=reshape(vk,s1*s2,1);
         if (~nonmulti)
             vk=V3Q'*(VV1(1:n,1:n,p)*V4Q);
             H(s1*s2+1:s1*s2+s3*s4,p)=reshape(vk,s3*s4,1);
         end
     end
end
coef(1:s1*s2,1)=reshape(V1Q'*rhs*V2Q,s1*s2,1);

% computed blocks of H associated with M3,M4
for k=1:s4
     if (symmA), G=U4Q(1:n,k)'./L;end % replica of the row u4Q((:,k)'
     for i=1:s3
         p=p+1;
         if (symmA)
              VV1(1:n,1:n,p)=U3Q(1:n,i).*G; % replica of column U3Q(:,i) (**)
         else
              VV1(1:n,1:n,p)=lyap(R,-U3Q(:,i)*U4Q(:,k)');
         end

         vk=V1Q'*(VV1(1:n,1:n,p)*V2Q);
         H(1:s1*s2,p)=reshape(vk,s1*s2,1);
```

```
          vk=V3Q'*(VV1(1:n,1:n,p)*V4Q);
          H(s1*s2+1:s1*s2+s3*s4,p)=reshape(vk,s3*s4,1);
     end
end
if (~nonmulti)
     coef(s1*s2+1:s1*s2+s3*s4,1)=reshape(V3Q'*rhs*V4Q,s3*s4,1);
end
C=(eye(s1*s2+s3*s4)+H)\coef;
Stot=reshape(VV1,n^2,s1*s2+s3*s4)*C;
X=Q*(rhs-reshape(Stot,n,n))*Q';
```

Unless n is significantly larger than $\sum_i s_i^2$, the commands (*) and (**) are the most expensive steps in the whole procedure, usually taking about 50% of the overall CPU time. For instance, this is the case for $n = 200$ and $s_1 = s_2 = s_3 = s_4 = 8$, whereas for $n = 400$ and $s_1 = s_2 = s_3 = s_4 = 2$ the eigenvalue decomposition takes over 35% of the total time for A dense and symmetric (Matlab profile was used for this analysis).

APPENDIX B

In this appendix, we report the algorithm described in Reference 16, together with a detailed computational cost. Only the costs marked with a star are counted in the comparison, to only account for the generic implementation.

Algorithm B1. The algorithm proposed in Reference [16] for $\ell = 1$

```
1:  Set $A_1 = [U_1 \otimes U_1] \in \mathbb{R}^{n^2 \times s^2}$, $A_2 = [V_1 \otimes V_1] \in \mathbb{R}^{n^2 \times s^2}$ $\boxed{2(n^2s^2) \star}$
2:  Compute $Q = [Q_1, Q_2] \in \mathbb{R}^{n^2 \times s_q}$ w/orth. columns s.t., range$(A_1)$ = range$(Q_1)$ and range$(Q)$ = range$([A_1, A_2])$ $\boxed{8s^4n^2 - \frac{8}{3}s^6}$
3:  Compute $H = Q_1^T(U_1 \otimes U_1)(V_1 \otimes V_1)^T Q \in \mathbb{R}^{s_{q_1} \times s_q}$ $\boxed{6s^4n^2 + 4s^6 - 5s^4}$;
4:  Perform SVD: $H = \tilde{U}_1 \Sigma \tilde{V}_1^T$ with $r$ = rank$(\Sigma)$ $\boxed{32s^6}$
5:  Set $P_1 = (p_1, \dots, p_r) = Q_1 \tilde{U}_1 \Sigma$, $P_2 = \tilde{V}_1^T Q^T$ $\boxed{s^4 + 6s^4n^2 - 2s^2n^2}$
6:  Compute Schur decomposition of $A = MRM^*$ $\boxed{25n^3 \star}$
7:  For $k = 1, \dots, s^2$
8:    Solve $AS_k + S_kA^T = \text{vec}^{-1}(p_k)$ using Schur dec. $\boxed{\text{total: } (10n^3 - 4n^2)s^2 \star}$
9:    Compute $g_k = P_2\text{vec}(\mathcal{L}^{-1}(\text{vec}^{-1}(p_k)))$ $\boxed{\text{total: } s^4(2n^2 - 1) \star}$
10:   Set $G = (g_1, \dots, g_r)$.
11: Compute $y = P_2\text{vec}(\mathcal{L}^{-1}(Y))$ $\boxed{10n^3 - 4n^2 + s^2(2n^2 - 1) \star}$ and $w = (I - G)^{-1}y$ $\boxed{\frac{2}{3}s^6 \star}$
12: $x = \mathcal{L}^{-1}(Y - \text{vec}^{-1}(P_1 w))$ $\boxed{10n^3 + 2n^2s^2 - 4n^2 \star}$
```