Recursive blocked algorithms for linear systems with Kronecker product structure

Minhong Chen[*]   Daniel Kressner[†]

May 24, 2019

Abstract

Recursive blocked algorithms have proven to be highly efficient at the numerical solution of the Sylvester matrix equation and its generalizations. In this work, we show that these algorithms extend in a seamless fashion to higher-dimensional variants of generalized Sylvester matrix equations, as they arise from the discretization of PDEs with separable coefficients or the approximation of certain models in macroeconomics. By combining recursions with a mechanism for merging dimensions, an efficient algorithm is derived that outperforms existing approaches based on Sylvester solvers.

1 Introduction

In computations with matrices, recursive blocked algorithms offer an elegant way to arrive at implementations that benefit from increased data locality and efficiently utilize highly tuned kernels. See [7] for a survey and [22] for a more recent testimony of this principle. These algorithms have proven particularly effective for solving Sylvester equations, that is, matrix equations of the form

$$A_1 X + X A_2^T = B, \tag{1.1}$$

where $A_1 \in \mathbb{R}^{n_1 \times n_1}, A_2 \in \mathbb{R}^{n_2 \times n_2}, B \in \mathbb{R}^{n_1 \times n_2}$ are given and $X \in \mathbb{R}^{n_1 \times n_2}$ is unknown. In the Bartels-Stewart algorithm [2], the matrices A_1 and A_2 are first reduced to block upper form by real Schur decompositions. The reduced problem is then solved by a variant of backward substitution. Both stages of the algorithms require $O(n^3)$ operations, with $n = \max\{n_1, n_2\}$. Entirely consisting of level 2 BLAS operations, the backward substitution step performs quite poorly. To avoid this, Jonsson and Kågström [12, 13] have proposed recursive algorithms for triangular Sylvester and related matrix equations. The recursive algorithm for solving (1.1) with upper quasi-triangular A_1, A_2 starts with

partitioning the matrix of larger size. Assuming $n_1 \geq n_2$, let $A_1 = \begin{pmatrix} A_{1,11} & A_{1,12} \\ 0 & A_{1,22} \end{pmatrix}$ with

$A_{1,11} \in \mathbb{R}^{k \times k}$ such that $k \approx n/2$ and partition $X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$, $B = \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$ correspondingly. Then (1.1) becomes equivalent to

$$A_{1,11}X_1 + X_1 A_2^T = B_1 - A_{1,12}X_2, \tag{1.2a}$$

$$A_{1,22}X_2 + X_2 A_2^T = B_2. \tag{1.2b}$$

First the Sylvester equation (1.2b) is solved recursively, then the right-hand side (1.2a) is updated, and finally (1.2a) is solved recursively. Apart from the solution of small-sized Sylvester equations at the lowest recursion level, the entire algorithm consists of matrix-matrix multiplications $A_{1,12}X_2$ and thus attains high performance by leveraging level 3 BLAS. As emphasized in [7, 22], recursive algorithm are less sensitive to parameter tuning compared to blocked algorithms.

The described algorithm extends to generalized and coupled Sylvester equations, such as $A_1 X M_1 + M_2 X A_2^T = B$; see [13, 23]. Interestingly, the numerically stable recursive formulation of Hammarling's method [11] for solving stable Lyapunov equations remains an open problem [16].

In this paper, we propose several new extensions that address high-dimensional variants of Sylvester equations. More specifically, we aim at computing a tensor $\mathbf{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ satisfying the linear equation

$$\mathcal{A}\mathbf{X} = \mathbf{B}, \tag{1.3}$$

where $\mathcal{A} : \mathbb{R}^{n_1 \times \cdots \times n_d} \to \mathbb{R}^{n_1 \times \cdots \times n_d}$ is a linear operator and $\mathbf{B} \in \mathbb{R}^{n_1 \times n_2 \cdots \times n_d}$. For $d = 2$, this formulation includes the Sylvester equation (1.1) and its generalizations mentioned above as special cases. For example, for (1.1) the matrix representation of $\mathcal{A}$ is given by $A = A_2 \otimes I_{n_1} + I_{n_2} \otimes A_1$.

The operator $\mathcal{A}$ needs to be of a very particular form such that (1.3) is amenable to the techniques discussed in this work. Motivated by their relevance in applications, we focus on two classes of operators.

Linear systems with Laplace-like structure. In Section 2, we consider discrete Laplace-like operators $\mathcal{A}$ having the matrix representation

$$A = A_d \otimes I_{n_{d-1}} \otimes \cdots \otimes I_{n_1} + I_{n_d} \otimes A_{d-1} \otimes I_{n_{d-2}} \otimes \cdots \otimes I_{n_1} + \cdots + I_{n_d} \otimes \cdots \otimes I_{n_2} \otimes A_1, \tag{1.4}$$

with $A_\mu \in \mathbb{R}^{n_\mu \times n_\mu}$, $\mu = 1, \ldots, d$. Using the vectorization of tensors, (1.3) can equivalently be written as $A \operatorname{vec}(\mathbf{X}) = \operatorname{vec}(\mathbf{B})$. Discrete Laplace-like operators arise from the structured discretization of d-dimensional PDEs with separable coefficients on tensorized domains. For more general PDEs, matrices of the form (1.4) can sometimes be used to construct effective preconditioners; see [24, 25] for examples. Other applications of (1.4) arise from Markov chain models [5, 26] used, e.g., for simulating interconnected systems.

Generalized Sylvester equations with Kronecker structure. Section 3 is concerned with the second class of operators $\mathcal{A}$ considered in this work, which have a matrix representation of the form

$$A = I_{n_d} \otimes I_{n_{d-1}} \otimes \cdots \otimes I_{n_2} \otimes A_1 + A_d \otimes A_{d-1} \otimes \cdots \otimes A_2 \otimes C, \qquad (1.5)$$

with $A_\mu \in \mathbb{R}^{n_\mu \times n_\mu}$ for $\mu = 1, \ldots, d$ and $C \in \mathbb{R}^{n_1 \times n_1}$. For $d = 2$, the linear system (1.3) now becomes equivalent to the generalized Sylvester equation $A_1 X + CXA_2^T$. For $d > 2$, we can view (1.3) equivalently as a generalized Sylvester equations with coefficients that feature Kronecker structure. If $A_1 = -\lambda I$ for some $\lambda \in \mathbb{R}$ then

$$A = A_d \otimes \cdots \otimes A_2 \otimes C - \lambda I_{n_1 \cdots n_d}. \qquad (1.6)$$

Linear systems featuring such shifted Kronecker products have been discussed in [19]. The more general case (1.5) arises from approximations of discrete time DSGE models [3], which play a central role in macroeconomics.

Recent work on the solution of linear tensor equations (1.3) has focused on the development of highly efficient approximate and iterative solvers that assume and exploit low-rank tensor structure in the right-hand side and the solution; see [9, 10] for overviews. In some cases, these developments can be combined with the methods developed in this work, which do not assume any such structure. For example, if the tensor Krylov subspace method [17] is applied to (1.4) for large-scale coefficients A_μ then our method can be used to solve the smaller-sized linear systems occurring in the method. As far as we know, all existing direct non-iterative solvers for linear tensor equations combine the Bartels-Stewart method for (generalized) Sylvester equations with a recursive traversal of the dimension. Instances of this approach can be found in [18, 27] for (1.4), in [14] for (1.5), and in [19] for (1.6). For $d \geq 3$, we are not aware of any work on (recursive) blocked methods that would allow for the effective use of level 3 BLAS.

2   A recursive blocked algorithm for Laplace-like equations

Let us first recall two basic operations for tensors from [15]. The μth matricization of a tensor $\mathbf{X} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is the matrix $X_{(\mu)} \in \mathbb{R}^{n_\mu \times (n_1 \cdots n_{\mu-1} n_{\mu+1} \cdots n_d)}$ obtained by mapping the μth index to the rows and all other indices to the columns:

$$X_{(\mu)}(i_\mu, j) = \mathcal{X}(i_1, \ldots, i_d),$$

with the column index j defined via the index map

$$j = \mathrm{i}(i_1, \ldots, i_{\mu-1}, i_{\mu+1}, \ldots, i_d) := 1 + \sum_{\substack{\nu=1 \\ \nu \neq \mu}}^{d} (i_\nu - 1) \prod_{\substack{\eta=1 \\ \eta \neq \mu}}^{\nu-1} n_\eta. \qquad (2.1)$$

The μ-mode matrix multiplication of $\mathbf{X}$ with a matrix $A \in \mathbb{R}^{n_1 \times m}$ is the tensor $\mathbf{Y} = \mathbf{X} \times_\mu A$ satisfying $Y_{(\mu)} = AX_{(\mu)}$. This allows us to rewrite (1.3)–(1.4) as

$$\mathbf{X} \times_1 A_1 + \mathbf{X} \times_2 A_2 + \cdots + \mathbf{X} \times_d A_d = \mathbf{B}. \qquad (2.2)$$

3

It is well known that this equation has a unique solution if and only if $\lambda_1 + \cdots + \lambda_d \neq 0$ for any eigenvalues λ_1 of A_1, λ_2 of A_2, etc. In the following, we will assume that this condition is satisfied.

Algorithm 1 describes our general framework for solving (2.2). Using real Schur decompositions [8, Sec. 7.4], the coefficient matrices are first transformed to reduced form. More specifically, for each $\mu = 1, \ldots, d$ an orthogonal matrix U_μ is computed such that $\tilde{A}_\mu := U_\mu^T A_\mu U_\mu$ is in upper quasi-triangular form, that is, $\tilde{A}_\mu$ is an upper block triangular matrix with 1×1 blocks containing its real eigenvalues and 2×2 blocks containing its complex eigenvalues in conjugate pairs. The right-hand side and the solution tensor need to be transformed accordingly by μ-mode matrix multiplications. For the rest of this section, we focus on line 3 of Algorithm 1, that is, the solution of the tensor equation with the reduced coefficients.

Algorithm 1 Solution of general Laplace-like equation (1.3)

1: Compute real Schur decompositions $A_1 = U_1 \tilde{A}_1 U_1^T, \ldots, A_d = U_d \tilde{A}_d U_d^T$.
2: Update $\tilde{\mathbf{B}} = \mathbf{B} \times_1 U_1^T \times_2 U_2^T \cdots \times_d U_d^T$.
3: Compute solution $\tilde{\mathbf{X}}$ of tensor equation (2.2) with quasi-triangular coefficients $\tilde{A}_1, \ldots, \tilde{A}_d$ and right-hand side $\tilde{\mathbf{B}}$.
4: Update $\mathbf{X} = \tilde{\mathbf{X}} \times_1 U_1 \times_2 U_2 \cdots \times_d U_d$

2.1 Recursion

By Algorithm 1, we may assume that $A_1 \in \mathbb{R}^{n_1 \times n_1}, \ldots, A_d \in \mathbb{R}^{n_d \times n_d}$ are already in upper quasi-triangular form. Choose μ such that $n_\mu = \max_\nu n_\nu$ and k such that $k \approx n_\mu/2$ and $A_\mu(k+1, k) = 0$. Partitioning $A_\mu = \begin{pmatrix} A_{\mu,11} & A_{\mu,12} \\ 0 & A_{\mu,22} \end{pmatrix}$ with $A_{\mu,11} \in \mathbb{R}^{k \times k}$, equation (2.2) becomes equivalent to

$$\mathbf{X}_1 \times_\mu A_{\mu,11} + \sum_{\substack{\nu=1 \\ \nu \neq \mu}}^{d} \mathbf{X}_1 \times_\nu A_\nu = \mathbf{B}_1 - \mathbf{X}_2 \times_\mu A_{\mu,12}, \tag{2.3a}$$

$$\mathbf{X}_2 \times_\mu A_{\mu,22} + \sum_{\substack{\nu=1 \\ \nu \neq \mu}}^{d} \mathbf{X}_2 \times_\nu A_\nu = \mathbf{B}_2, \tag{2.3b}$$

where

$$\mathbf{X}_1 = \mathbf{X}(1:n_1, \ldots, 1:n_{\mu-1}, 1:k, 1:n_{\mu+1}, \ldots, 1:n_d), \tag{2.4a}$$

$$\mathbf{X}_2 = \mathbf{X}(1:n_1, \ldots, 1:n_{\mu-1}, k+1:n_\mu, 1:n_{\mu+1}, \ldots, 1:n_d), \tag{2.4b}$$

and $\mathbf{B}_1, \mathbf{B}_2$ are defined analogously. Noting that (2.3b) and (2.3a) are again equations with Laplace-like operators, they can be solved recursively. The recursion is stopped once the maximal size is below a user-specified block size $n_{\min} \geq 2$. These considerations lead to Algorithm 2.

4

Algorithm 2 Recursive sol. of quasi-triang. tensor eqn: $\mathbf{X} = \mathtt{reclap}(A_1, \ldots, A_d, \mathbf{B})$

1: Determine $n_\mu = \max_\nu n_\nu$.
2: **if** $n_\mu \leq n_{\min}$ **then** solve full system and return **end if**
3: Set $k = \lfloor n_\mu/2 \rfloor$. **if** $A_\mu(k+1, k) \neq 0$ **then** $k = k+1$ **end if**
4: Set $\mathbf{B}_1 = \mathbf{B}(:, \ldots, :, 1 : k, :, \ldots, :)$, $\mathbf{B}_2 = \mathbf{B}(:, \ldots, :, k+1 : n_\mu, :, \ldots, :)$.
5: Call $\mathbf{X}_2 = \mathtt{reclap}(A_1, \ldots, A_{\mu-1}, A_\mu(k+1 : n_\mu, k+1 : n_\mu), A_{\mu+1}, \ldots, A_d, \mathbf{B}_2)$.
6: Update $\mathbf{B}_1 \leftarrow \mathbf{B}_1 - \mathbf{X}_2 \times_\mu A_\mu(1 : k, k+1 : n_\mu)$.
7: Call $\mathbf{X}_1 = \mathtt{reclap}(A_1, \ldots, A_{\mu-1}, A_\mu(1 : k, 1 : k), A_{\mu+1}, \ldots, A_d, \mathbf{B}_2)$.
8: Concatenate $\mathbf{X}_1, \mathbf{X}_2$ along μth mode into $\mathbf{X}$.

Let $\mathtt{comp}(n)$ denote the complexity of Algorithm 2 for even $n = n_1 = \cdots = n_d$. On the top level of recursion Algorithm 2 is applied to one $n \times \cdots \times n$ tensor, on the second level to two $n/2 \times n \times \cdots \times n$ tensors, on the third level to four $n/2 \times n/2 \times n \times \cdots \times n$ tensors, and so on. Under the slightly simplified assumption that the multiplication of an $n/2 \times n/2$ quasi-triangular matrix with a vector requires $n^2/4$ floating point operations (flops), each level of the first d recursions requires a total of $n^{d+1}/4$ flops to execute the matrix-matrix multiplications in line 6 of Algorithm 2. After d recursions of Algorithm 2, n has been reduced to $n/2$ in each mode and, therefore, $\mathtt{comp}(n) = dn^{d+1}/4 + 2^d\mathtt{comp}(n/2)$. Assuming that $n/n_{\min}$ is a power of two, we obtain

$$\mathtt{comp}(n) = O\big(n^{d+1}\big) + (2^d)^{\log_2 n/n_{\min}}\mathtt{comp}(n_{\min}) = O\big(n^{d+1}\big) + \frac{n^d}{n_{\min}^d}\mathtt{comp}(n_{\min}). \quad (2.5)$$

Once the maximal size of the tensor is $n_{\min}$ or below, line 2 of Algorithm 2 assembles the matrix A defined in (1.4) and solves the block triangular linear system $A \operatorname{vec}(\mathbf{X}) = \operatorname{vec}(\mathbf{B})$ by backward substitution. This requires $O\big((n_{\min})^{2d}\big)$ flops and therefore

$$\mathtt{comp}(n) = O\big(n^{d+1} + n_{\min}^d n^d\big). \quad (2.6)$$

This compares favorably with the $O(n^{2d})$ operations needed by backward substitution applied to the assembled full triangular linear system. The complexity estimate (2.6) also reflects the critical role played by the solution of the small systems in line 2. On the one hand, the operation count suggests to choose $n_{\min}$ as small as possible, say, $n_{\min} = 2$. On the other hand, it has been observed for $d = 2$ in [12] that a small value of $n_{\min}$ creates significant overhead and requires very well tuned kernels. In the following section, we describe a technique that alleviates this difficulty.

2.2 Merging dimensions: triangular case

To avoid the critical dependence on $n_{\min}$ observed in (2.6) we replace line 2 of Algorithm 2 by the following procedure. Once $n_1 n_2 \leq n_{\min}^2$, the matrix

$$A_1' = I_{n_2} \otimes A_1 + A_2 \otimes I_{n_1} \quad (2.7)$$

is formed explicitly. For the moment, let us suppose that A_1 and A_2 are upper triangular. This can be achieved by computing complex instead of real Schur decompositions in

Algorithm 1, leading to a triangular tensor equation with complex coefficients. Because of roundoff error, the computed solution to the original equation will now have a (small) imaginary part. This can be safely set to zero [20].

The matrix A_1' inherits the triangular structure from A_1, A_2 and the d-dimensional tensor equation (2.2) is equivalent to the $(d-1)$-dimensional equation

$$\mathbf{X}' \times_1 A_1' + \mathbf{X}' \times_3 A_3 + \cdots + \mathbf{X}' \times_d A_d = \mathbf{B}', \tag{2.8}$$

with reshaped $\mathbf{X}', \mathbf{B}' \in \mathbb{C}^{n_1 n_2 \times n_3 \cdots \times n_d}$. This equation is solved recursively. A major advantage, this approach allows us to reduce d. For $d = 3$, the system (2.8) becomes the triangular Sylvester equation

$$A_1' \mathbf{X}' + \mathbf{X}' A_3^T = \mathbf{B}',$$

to which the efficient solvers described in Section 1 can be applied. Note that A_3^T now refers to the *complex* transpose of $A_3 \in \mathbb{C}^{n_3 \times n_3}$. Algorithm 3 summarizes the proposed procedure.

Algorithm 3 Recursive sol. of triangular tensor equation: $\mathbf{X} = \texttt{mrglap}(A_1, \ldots, A_d, \mathbf{B})$

1: **if** $n_1 n_2 \leq n_{\min}^2$ **then**
2:     Compute $A_1' = I_{n_2} \otimes A_1 + A_2 \otimes I_{n_1}$.
3:     Reshape $\mathbf{B}'(1 : n_1 n_2, :, \ldots, :) = \mathbf{B}(1 : n_1, 1 : n_2, :, \ldots, :)$.
4:     **if** $d = 3$ **then**
5:         Solve Sylvester equation $A_1' \mathbf{X}' + \mathbf{X}' A_3^T = \mathbf{B}'$.
6:     **else**
7:         Call $\mathbf{X}' = \texttt{mrglap}(A_1', A_3, \ldots, A_d, \mathbf{B}')$
8:     **end if**
9:     Reshape $\mathbf{X}(1 : n_1, 1 : n_2, :, \ldots, :) = \mathbf{X}'(1 : n_1 n_2, :, \ldots, :)$.
10: **else**
11:     Determine $n_\mu = \max_\nu n_\nu$ and set $k = \lfloor n_\mu / 2 \rfloor$.
12:     Set $\mathbf{B}_1 = \mathbf{B}(:, \ldots, :, 1 : k, :, \ldots, :)$, $\mathbf{B}_2 = \mathbf{B}(:, \ldots, :, k+1 : n_\mu, :, \ldots, :)$.
13:     Call $\mathbf{X}_2 = \texttt{mrglap}(A_1, \ldots, A_{\mu-1}, A_\mu(k+1 : n_\mu, k+1 : n_\mu), A_{\mu+1}, \ldots, A_d, \mathbf{B}_2)$.
14:     Update $\mathbf{B}_1 \leftarrow \mathbf{B}_1 - \mathbf{X}_2 \times_\mu A_\mu(1 : k, k+1 : n_\mu)$.
15:     Call $\mathbf{X}_1 = \texttt{mrglap}(A_1, \ldots, A_{\mu-1}, A_\mu(1 : k, 1 : k), A_{\mu+1}, \ldots, A_d, \mathbf{B}_2)$.
16:     Concatenate $\mathbf{X}_1, \mathbf{X}_2$ along μth mode into $\mathbf{X}$.
17: **end if**

To analyze the complexity of Algorithm 3 for $n_1 = \cdots = n_d = n > 2n_{\min}$, we observe that all sizes are first reduced to $2n_{\min}$ or below before the condition in line 1 is met. Hence, up to constant factors the recursive estimate (2.5) holds and it remains to discuss the complexity for $n_1 = \cdots = n_d = n_{\min}$, which will be denoted by $\overline{\mathsf{comp}}_d(n_{\min})$. The merge in line 2 reduces the order to $d - 1$ but increases the first mode size to $n_{\min}^2$. Approximately $\log_2(n_{\min}^2 / n_{\min}) = \log_2 n_{\min}$ recursions are needed to reduce it back to $n_{\min}$. Similarly as in Section 2.1 we calculate

$$\overline{\mathsf{comp}}_d(n_{\min}) = O\left(n_{\min}^{d+2}\right) + n_{\min} \overline{\mathsf{comp}}_{d-1}(n_{\min}) = O\left(n_{\min}^{d+2}\right) + n_{\min}^{d-3} \overline{\mathsf{comp}}_3(n_{\min}).$$

For $d = 3$, the solution of the triangular Sylvester equation in line 5 requires $O\big(n_{\min}^5\big)$ flops. In turn, $\overline{\mathsf{comp}}_d(n_{\min}) = O\big(n_{\min}^{d+2}\big)$. Inserted into (2.5), we arrive at

$$O\big(n^{d+1} + n_{\min}^2 n^d\big)$$

flops for Algorithm 3. For $d \geq 3$, this compares favorably with the complexity estimate (2.6) for Algorithm 2; the dependence on $n_{\min}$ has been reduced significantly. Equally importantly, Algorithm 3 allows us to leverage efficient solvers for triangular Sylvester equations, such as the ones described in [12].

2.3 Merging dimensions: quasi-triangular case

The use of complex arithmetic, which increases the cost (by a constant factor) in terms of operations and memory, can be avoided when using the real Schur form and working with quasi-triangular coefficients. However, a few modifications are needed because the matrix A_1' formed in (2.7) does *not* inherit the quasi-triangular structure from A_1 and A_2. To illustrate what happens, let us consider the following example for $n_1 = 3, n_2 = 4$:

$$A_1 = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix}, \quad A_2 = \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix}, \quad A_1' = \begin{bmatrix} \times & \times & \times & \times & \times & 0 & 0 & 0 & \times & 0 & 0 & 0 \\ 0 & \times & \times & \times & 0 & \times & 0 & 0 & 0 & \times & 0 & 0 \\ 0 & \times & \times & \times & 0 & 0 & \times & 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & \times & 0 & 0 & 0 & \times & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \times & \times & \times & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \times & \times & \times & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \times & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & \times & 0 & 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times & 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & \times & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \times & 0 & 0 & 0 & \times \end{bmatrix}. \quad (2.9)$$

The diagonal matrix at the (3,2) block disturbs the quasi-triangular structure of A_1'. More generally, assuming $n_1 = n_2 = n_{\min}$ the matrix A_1' is an $n_{\min}^2 \times n_{\min}^2$ block upper triangular matrix with diagonal blocks of size at most $n_{\min}$. This matrix can be returned to quasi-triangular form by computing a real Schur decomposition of A_1'. The impact of this operation on the overall cost of Algorithm 3 can be made negligible by exploiting the structure of A_1':

- When the structure of A_1' is completely ignored, its real Schur decomposition takes $O(n_{\min}^6)$ flops and, in turn, the complexity of Algorithm 3 increases to $O\big(n^{d+1} + n_{\min}^3 n^d\big)$.

- When the block triangular structure of A_1' is taken into account, the cost of computing its real Schur decomposition reduces to $O(n_{\min}^5)$ flops. When used within Algorithm 3, the additional flops spent on performing these decompositions and applying the resulting orthogonal transformations amounts to $O\big(n_{\min}^2 n^d\big)$ in total. In turn, this operation does not increase the complexity of Algorithm 3 but its dependence on $n_{\min}^2$ is not negligible either.

- The diagonal structure of the off-diagonal blocks of A_1' can be exploited to reduce the cost further, using a permutation trick similar to the one discussed in [19]. To illustrate this, consider the 12×12 matrix A_1' from (2.9). By applying a perfect shuffle permutation [28] to the last 8 rows and columns, we obtain the permuted matrix

$$P^T A_1' P = \left[\begin{array}{cccc|cc|cccc|cc}
\times & \times & \times & \times & \times & \times & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \times & \times & \times & 0 & 0 & \times & \times & 0 & 0 & 0 & 0 \\
0 & \times & \times & \times & 0 & 0 & 0 & 0 & \times & \times & 0 & 0 \\
0 & 0 & 0 & \times & 0 & 0 & 0 & 0 & 0 & 0 & \times & \times \\
\hline
0 & 0 & 0 & 0 & \times & \times & \times & 0 & \times & 0 & \times & 0 \\
0 & 0 & 0 & 0 & \times & \times & 0 & \times & 0 & \times & 0 & \times \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & \times & \times & \times & 0 & \times & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \times & \times & 0 & \times & 0 & \times \\
0 & 0 & 0 & 0 & 0 & 0 & \times & 0 & \times & \times & \times & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \times & \times & \times & 0 & \times \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \times & \times \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \times & \times
\end{array}\right].$$

In the general case, applying such a permutation to each $n_{\min} \times n_{\min}$ diagonal block transforms A_1' into a block upper triangular matrix with diagonal blocks of size at most 4. This reduces the cost of computing its real Schur decomposition to $O(n_{\min}^4)$ flops and the overall impact of this operation on the cost of Algorithm 3 becomes negligible.

2.4 Numerical experiments

All algorithms proposed in this work have been implemented in MATLAB R2019a and executed on a Lenovo ThinkPad T460, which comes with an Intel Core i5-6300U processor and 8 Gbytes of DDR3L-RAM. The implementation of the algorithms together with scripts for reproducing each of the experiments reported in this work are available from https://anchp.epfl.ch/misc/.

Care has been taken to avoid unnecessary overhead in our MATLAB implementation. For example, the `tensor` object from the Tensor Toolbox [1] is very convenient for realizing tensor operations but our preliminary experiments indicated that its use in Algorithms 2 and 3 would lead to significant performance loss, possibly due to excessive memory transfer. Instead, we directly use MATLAB arrays, combined with the `permute` and `reshape` functions for implementing μ-mode matrix multiplications. For solving triangular Sylvester equations, as needed, e.g., in Algorithm 3, we utilize the internal MATLAB function `sylvester_tri`. This function seems to be based on the algorithms presented in [12, 13] and avoids performing any additional Schur decomposition.

The techniques from Section 2.3, which allow for the use of real arithmetic in Algorithm 3, have been implemented and verified. However, we observed that none of the three described variants leads to competitive performance, any benefit from structure exploitation is offset by the overhead it incurs in MATLAB, due to the relatively small values of $n_{\min}$ needed for reaching good performance. In the following, we therefore consistently use complex Schur decompositions for reducing all coefficients to triangular form. All reported times include the time needed by Algorithm 1 for performing these

(a) Algorithm 2	(b) Algorithm 3

Figure 1: Execution times (in seconds) vs. $n_{\min}$ for Algorithms 2 and 3 applied to random $n \times \cdots \times n$ tensors with $n = 80$ for $d = 3$ and $n = 25$ for $d = 4$.

decompositions and applying the corresponding transformations. The coefficients used in our experiments have been generated with `randn`.

Choice of $n_{\min}$. Figure 1 shows the execution times obtained for fixed n and varying $n_{\min}$. All numbers have been averaged over five consecutive runs. As to be expected from the complexity estimates, the performance of Algorithm 2 is very sensitive to the choice of $n_{\min}$, especially for $d = 4$. The smallest execution times are attained by $n_{\min} = 7$ for $d = 3$ and $n_{\min} = 3$ for $d = 4$. The performance of Algorithm 3 is not very sensitive to the choice of $n_{\min}$, provided that its value is not chosen too small. The smallest execution times are attained by $n_{\min} = 26$ for $d = 3$, $n_{\min} = 18$ for $d = 4$, and $n_{\min} = 14$ for $d = 5$. These values of $n_{\min}$ are used in the following.

Comparison. We have compared our newly proposed algorithms with the following procedure termed "Sylvester solver": After reducing the coefficients $A_1, \ldots, A_d$ of the Laplace-like equation (2.2) to triangular form and reshaping **B** suitably into a matrix B, one of the Sylvester equations

$$(I \otimes A_1 + A_2 \otimes I)X + XA_3^T = B, \quad (I \otimes A_1 + A_2 \otimes I)X + X(I \otimes A_3 + A_4 \otimes I)^T = B$$
$$(I \otimes A_1 + A_2 \otimes I)X + X(I \otimes I \otimes A_3 + I \otimes A_4 \otimes I + A_5 \otimes I \otimes I)^T = B$$

is solved for $d = 3, 4, 5$ by calling `sylvester_tri`. The results reported in Figure 2 confirm that Algorithms 2 and 3 have the same asymptotic cost. However, Algorithm 3 is always faster, by an order of magnitude for sufficiently large n. For $d = 3$, the Sylvester solver is nearly always slower than Algorithm 3. For $d = 4$, the picture is less clear; only for $n \geq 50$ becomes Algorithm 3, which has complexity $O(n^5)$, consistently faster than the Sylvester solver, which has complexity $O(n^6)$. For $d = 5$, the difference in complexity is more pronounced and, in turn, Algorithm 3 is nearly always faster.

9

(a) $d = 3$

(b) $d = 4$

(c) $d = 5$

Figure 2: Execution times (in seconds) vs. n for Algorithms 2 and 3 compared to Sylvester solver.

For all experiments performed, the norm of the residual was checked and no significant differences in terms of numerical stability were observed between the different algorithms tested.

3 A recursive blocked algorithm for generalized Sylvester equations with Kronecker structure

In this section, we extend the developments from Section 2 to the second class of operators $\mathcal{A}$ considered in this work, which have the matrix representation (1.5). The corresponding linear system reads in tensor notation as

$$\mathbf{X} \times_1 A_1 + \mathbf{X} \times_1 C \times_2 A_2 \times_3 A_3 \cdots \times_d A_d = \mathbf{B}. \qquad (3.1)$$

Because of its connection to generalized Sylvester equations [4] explained in the introduction, this equation has a unique solution if and only if the matrix pencil $A_1 + \lambda C$ is regular and none of its eigenvalues is an eigenvalue of $-A_d \otimes A_{d-1} \otimes \cdots \otimes A_2$. In the following, we assume that this condition is satisfied.

Algorithm 4 is the equivalent of Algorithm 1 for reducing (3.1) to quasi-triangular form. The most notable difference is that now a generalized Schur decomposition [8, Sec. 7.7.2] of $A_1 + \lambda C$ needs to be computed, using the QZ algorithm.

Algorithm 4 Solution of generalized Sylvester equation (3.1)

1: Compute real generalized Schur decomposition $A_1 = U_1 \tilde{A}_1 Z^T$, $C = U_1 \tilde{C} Z^T$.
2: Compute real Schur decompositions $A_2 = U_2 \tilde{A}_2 U_2^T, \cdots, A_d = U_d \tilde{A}_d U_d^T$.
3: Update $\hat{\mathbf{B}} = \mathbf{B} \times_1 U_1^T \times_2 U_2^T \cdots \times_d U_d^T$.
4: Compute solution $\tilde{\mathbf{X}}$ of tensor equation (3.1) with (quasi-)triangular coefficients $\tilde{C}, \tilde{A}_1, \ldots, \tilde{A}_d$ and right-hand side $\hat{\mathbf{B}}$.
5: Update $\mathbf{X} = \tilde{\mathbf{X}} \times_1 Z \times_2 U_2 \cdots \times_d U_d$

3.1 Recursion

The rest of this section is concerned with line 4 of Algorithm 4, solving (3.1) with upper quasi-triangular coefficients $A_1 \in \mathbb{R}^{n_1 \times n_1}, \ldots, A_d \in \mathbb{R}^{n_d \times n_d}$ and upper triangular $C \in \mathbb{R}^{n_1 \times n_1}$. Again we proceed recursively and choose μ such that $n_\mu = \max_\nu n_\nu$ and k such that $k \approx n_\mu/2$ and $A_\mu(k+1, k) = 0$. We partition $A_\mu = \begin{pmatrix} A_{\mu,11} & A_{\mu,12} \\ 0 & A_{\mu,22} \end{pmatrix}$, $A_{\mu,11} \in \mathbb{R}^{k \times k}$ and split the tensors $\mathbf{X}$ and $\mathbf{B}$ along their μth mode into $\mathbf{X}_1, \mathbf{X}_2$ and $\mathbf{B}_1, \mathbf{B}_2$, respectively, in accordance with (2.4).

Case 1: $\mu = 1$. We additionally partition $C = \begin{pmatrix} C_{11} & C_{12} \\ 0 & C_{22} \end{pmatrix}$ and decouple (3.1) along the first mode:

$$\mathbf{X}_1 \times_1 A_{1,11} + \mathbf{X}_1 \times_1 C_{11} \times_2 A_2 \times_3 \cdots \times_d A_d = \hat{\mathbf{B}}_1,$$
$$\mathbf{X}_2 \times_1 A_{1,22} + \mathbf{X}_2 \times_1 C_{22} \times_2 A_2 \times_3 \cdots \times_d A_d = \mathbf{B}_2,$$

11

with $\hat{\mathbf{B}}_1 := \mathbf{B}_1 - \mathbf{X}_2 \times_1 A_{1,12} - \mathbf{X}_2 \times_1 C_{12} \times_2 A_2 \times_3 \cdots \times_d A_d$. Both equations take the form of the tensor equation (3.1) with (quasi-)triangular coefficients. We recursively solve for $\mathbf{X}_2$ and then solve for $\mathbf{X}_1$, after computing $\hat{\mathbf{B}}_1$.

Case 2: $\mu \neq 1$. Decoupling (3.1) along the μth mode gives the two tensor equations

$$\mathbf{X}_1 \times_1 A_1 + \mathbf{X}_1 \times_1 C \times_2 \cdots \times_\mu A_{\mu,11} \times_{\mu+1} \cdots \times_d A_d = \hat{\mathbf{B}}_1,$$
$$\mathbf{X}_2 \times_1 A_1 + \mathbf{X}_2 \times_1 C \times_2 \cdots \times_\mu A_{\mu,22} \times_{\mu+1} \cdots \times_d A_d = \mathbf{B}_2,$$

with $\hat{\mathbf{B}}_1 := \mathbf{B}_1 - \mathbf{X}_2 \times_1 C \times_2 \cdots \times_\mu A_{\mu,12} \times_{\mu+1} \cdots \times_d A_d$. Again, we first solve for $\mathbf{X}_2$ and then for $\mathbf{X}_1$.

Algorithm 5 Recursive solution of triangular generalized Sylvester equation with tensor structure: $\mathbf{X} = \mathtt{recgsylvten}(A_1, \ldots, A_d, C, \mathbf{B})$

1: Determine $n_\mu = \max_\nu n_\nu$.
2: **if** $n_\mu \leq n_{\min}$ **then** solve full system and return **end if**
3: Set $k = \lfloor n/2 \rfloor$. **if** $A_\mu(k+1, k) \neq 0$ **then** $k = k+1$ **end if**
4: Set $\mathbf{B}_1 = \mathbf{B}(:, \ldots, :, 1:k, :, \ldots, :)$, $\mathbf{B}_2 = \mathbf{B}(:, \ldots, :, k+1:n_\mu, :, \ldots, :)$.
5: **if** $\mu = 1$ **then**
6:     Call $\mathbf{X}_2 = \mathtt{recgsylvten}(A_1(k+1:n_1, k+1:n_1), A_2, \ldots, A_d, C(k+1:n_1, k+1:n_1), \mathbf{B}_2)$.

7:     Update $\mathbf{B}_1 \leftarrow \mathbf{B}_1 - \mathbf{X}_2 \times_1 A_1(1:k, k+1:n_1) - \mathbf{X}_2 \times_1 C(1:k, k+1:n_1) \times_2 A_2 \times_3 \ldots \times_d A_d$.

8:     Call $\mathbf{X}_1 = \mathtt{recgsylvten}(A_1(1:k, 1:k), A_2 \ldots, A_d, C(1:k, 1:k), \mathbf{B}_2)$.
9: **else**
10:     Call $\mathbf{X}_2 = \mathtt{recgsylvten}(A_1, \ldots, A_{\mu-1}, A_\mu(k+1:n_\mu, k+1:n_\mu), A_{\mu+1}, \ldots, A_d, C, \mathbf{B}_2)$.
11:     Update $\mathbf{B}_1 \leftarrow \mathbf{B}_1 - \mathbf{X}_2 \times_1 C \times_2 A_2 \times_3 \cdots \times_\mu A_\mu(1:k, k+1:n_\mu) \times_{\mu+1} A_{\mu+1} \times_{\mu+2} \ldots \times_d A_d$.

12:     Call $\mathbf{X}_1 = \mathtt{recgsylvten}(A_1, \ldots, A_{\mu-1}, A_\mu(1:k, 1:k), A_{\mu+1}, \ldots, A_d, C, \mathbf{B}_2)$.
13: **end if**
14: Concatenate $\mathbf{X}_1, \mathbf{X}_2$ along μth mode into $\mathbf{X}$.

Algorithm 5 summarizes the described procedure. Compared to Algorithm 2, the largest difference is that the right-hand side updates in lines 7 and 11 require up to d matrix multiplications instead of only one. While potentially having an impact on computational time, this has no impact on the asymptotic complexity, which remains $O(n^{d+1} + n_{\min}^d n^d)$.

3.2 Merging dimensions: triangular case

In analogy to the discussion in Section 2.2, we now discuss the combination of Algorithm 5 with a merging procedure that helps to alleviate the critical dependence of its performance on $n_{\min}$. Again, we first suppose that all coefficients triangular. This can always be achieved by a variant of Algorithm 4 that uses complex (generalized) Schur decompositions.

Line 2 of Algorithm 5 is replaced with the following procedure. When $n_{d-1}n_d \leq n_{\min}^2$, the matrix

$$A'_{d-1} = A_d \otimes A_{d-1}$$

is formed explicitly. In turn, the d-dimensional tensor equation (3.1) can equivalently be viewed as the $(d-1)$-dimensional equation

$$\mathbf{X}' \times_1 A_1 + \mathbf{X}' \times_1 C \times A_2 \times_3 \cdots \times_d A'_{d-1} = \mathbf{B}',$$

with reshaped $\mathbf{X}', \mathbf{B}' \in \mathbb{R}^{n_1 \times \cdots \times n_{d-2} \times n_{d-1}n_d}$. For $d = 2$, this corresponds to the triangular generalized Sylvester equation $A_1 X + CXA_2^T = B$, for which a recursive blocked algorithm has been described in [13].

Algorithm 6 Recursive solution of triangular generalized Sylvester equation with tensor structure: $\mathbf{X} = \mathtt{mrggsylv}(A_1, \ldots, A_d, C, \mathbf{B})$

1: **if** $n_1 n_2 \leq n_{\min}^2$ **then**
2:    Compute $A'_{d-1} = A_d \otimes A_{d-1}$.
3:    Reshape $\mathbf{B}'(:, \ldots, :, 1 : n_{d-1}n_d) = \mathbf{B}(:, \ldots, :, 1 : n_{d-1}, 1 : n_d)$.
4:    **if** $d = 3$ **then**
5:      Solve generalized Sylvester equation $A_1 \mathbf{X}' + C\mathbf{X}'(A'_2)^T = B'$.
6:    **else**
7:      Call $\mathbf{X}' = \mathtt{mrggsylv}(A_1, A_2, \ldots, A_{d-2}, A'_{d-1}, \mathbf{B}')$.
8:    **end if**
9:    Reshape $\mathbf{X}(:, \ldots, :, 1 : n_{d-1}, 1 : n_d) = \mathbf{X}'(:, \ldots, :, 1 : n_{d-1}n_d)$.
10: **else**
11:    Determine $n_\mu = \max_\nu n_\nu$ and set $k = \lfloor n_\mu/2 \rfloor$.
12:    Execute lines 4 to 14 of Algorithm 5 with calls to $\mathtt{recgsylv}$ replaced by calls to $\mathtt{mrggsylv}$.
13: **end if**

A straightforward extension of the complexity analysis of Algorithm 3 shows that Algorithm 6 requires $O(n^{d+1} + n_{\min}^2 n^d)$ flops.

3.3 Merging dimensions: quasi-triangular case

When using real (generalized) Schur decompositions and, in turn, dealing with upper quasi-triangular coefficients $A_1, \ldots, A_d$, we are facing a situation similar to the one discussed in Section 2.3: The merged coefficient matrix $A'_{d-1} = A_d \otimes A_{d-1}$ is, in general, not quasi-triangular. The structure of A'_{d-1} is very similar but not identical with the

Laplace-like case. For example, comparing (2.9) with

$$
A_{d-1} = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix}, \quad A_d = \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix}, \quad A'_{d-1} = \left[\begin{array}{cccc|cccc|cccc} \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & 0 & \times & \times & \times & 0 & \times & \times & \times \\ 0 & \times & \times & \times & 0 & \times & \times & \times & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & 0 & 0 & 0 & \times & 0 & 0 & 0 & \times \\ \hline 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times & \times & \times & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times & \times & \times & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \times & 0 & 0 & 0 & \times \\ \hline 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times & \times & \times & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & \times & \times & \times & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \times & 0 & 0 & 0 & \times \end{array}\right], \quad (3.2)
$$

we see that the off-diagonal blocks now have quasi-triangular instead of diagonal structure. Nevertheless, the properties and techniques discussed in Section 2.3 carry over verbatim to A'_{d-1}. In particular, A'_{d-1} is a block diagonal matrix with diagonal blocks of size at most $2n_{d-1}$. Moreover, a perfect shuffle permutation of the diagonal blocks can again be used to further reduce the size of diagonal blocks. For example, applying this permutation to the second diagonal block of the matrix in (3.2) yields:

$$
P^T A'_{d-1} P = \left[\begin{array}{cccc|cc|cccc|cc} \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & 0 & 0 & \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & 0 & 0 & \times & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & 0 & 0 & 0 & 0 & 0 & 0 & \times & \times \\ \hline 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times & \times & \times \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \times & \times \end{array}\right].
$$

This modification allows to apply Algorithm 6 to quasi-triangular matrices without increased complexity.

3.4 Numerical experiments

To give some insight into the performance of Algorithms 5 and 6, we have implemented them in MATLAB and conducted numerical experiments in the setting described in Section 2.4. In particular, we again make use of complex (generalized) Schur decompositions, to avoid that the overhead incurred by the techniques described in Section 3.3 distorts the picture. To solve the triangular generalized Sylvester equation in Line 5 of Algorithm 6, we apply `sylvester_tri` to $A_1 \mathbf{X}' E^T + C \mathbf{X}' (A'_2)^T = B'$ with $E = I_{n_2}$.

Choice of $n_{\min}$. Figure 3 shows the performance of Algorithms 5 and 6 with respect to the choice of $n_{\min}$. Compared with Algorithms 2 and 3, see Figure 1, the findings do not differ much. In the following we set $n_{\min} = 8$ for $d = 3$, $n_{\min} = 6$ for $d = 4$ when using Algorithm 5, and $n_{\min} = 15$ for $d = 3$, $n_{\min} = 13$ for $d \geq 4$ when using Algorithm 6.

(a) Algorithm 5   (b) Algorithm 6

Figure 3: Execution times (in seconds) vs. $n_{\min}$ for Algorithms 5 and 6 applied to random $n \times \cdots \times n$ tensors with $n = 80$ for $d = 3$ and $n = 25$ for $d = 4$.

Comparison. Figure 4 compares the performance of Algorithm 5 and Algorithm 6 with the following "Sylvester solver": After reducing the coefficients $A_1, \ldots, A_d, C$ to triangular form and suitably reshaping $\mathbf{B}$, one of the Sylvester equations

$$
\begin{aligned}
A_1 X + C X (A_3 \otimes A_2)^T &= B \\
(I \otimes A_1)X + (A_2 \otimes C)X(A_4 \otimes A_3)^T &= B \\
(I \otimes A_1)X + (A_2 \otimes C)X(A_5 \otimes A_4 \otimes A_3)^T &= B
\end{aligned}
$$

is solved for $d = 3, 4, 5$ by calling `sylvester_tri`. The results from Figure 4 show that Algorithm 6 is always faster than Algorithm 5. The Sylvester solver is slower for sufficiently large n; the difference is most pronounced for $d = 3$. Moreover, the Sylvester solver encounters out of memory errors for $n > 110$, $n > 50$, $n > 20$ for $d = 3, 4, 5$, respectively.

4 Conclusions, extensions and future work

We have extended the concept of blocked recursive algorithms to higher-order tensor equations. Both, the complexity estimates and the numerical results, clearly show the importance of combining recursion with merging dimensions in order to arrive at efficient algorithms. For third-order tensor equations, these algorithms seem to constitute the methods of choice. For fourth-order tensor equations with coefficients of nearly equal sizes, reshaping the tensor equation into a Sylvester equation and applying an existing solver is a viable alternative, provided that sufficient memory is available.

The blocked recursive algorithms developed in this work certainly admit extensions

(a) $d = 3$

(b) $d = 4$

(c) $d = 5$

Figure 4: Execution times (in seconds) vs. n for Algorithms 5 and 6 compared to Sylvester solver.

to general linear tensor equations taking the form

$$\sum_{k=1}^{K} \mathbf{X} \times_1 A_1^{(k)} \times_2 A_2^{(k)} \times_3 \cdots \times_d A_d^{(k)} = \mathbf{B},$$

assuming that all coefficients $A_\mu^{(k)} \in \mathbb{R}_{n_\mu \times n_\mu}$ are (quasi-)triangular. To transform general coefficients $A_\mu^{(k)}$ into this form requires the existence of invertible matrices Q_μ, Z_μ such that $Q_\mu^T A_\mu^{(k)} Z_\mu$ is (quasi-)triangular for every $k = 1, \ldots, K$. For $K \geq 3$, this simultaneous triangularization is only possible under strong additional assumptions on the coefficients. A sufficient condition is that each matrix family $\{A_\mu^{(1)}, \ldots, A_\mu^{(K)}\}$ contains at most two different matrices for $\mu = 1, \ldots, d$. The two classes (1.4) and (1.5) appear to constitute the practically most important examples satisfying this condition.

This work also raises an interesting open question: Is it possible to combine block recursion with low-rank compression, for example in the tensor train format [21], such that the complexity does not grow exponentially with d, assuming that the involved ranks stay constant? It would also be interesting to explore which other numerical linear algebra problems allow for the combination of Kronecker product structure with block recursion. The computation of certain matrix functions, such as the matrix square root [6], appears to be a likely candidate.

References

[1] B. W. Bader, T. G. Kolda, et al. Matlab tensor toolbox version 2.6. Available from `http://www.sandia.gov/~tgkolda/TensorToolbox/`, 2015.

[2] R. H. Bartels and G. W. Stewart. Algorithm 432: The solution of the matrix equation $AX + XB = C$. *Communications of the ACM*, 15(9):820–826, 1972.

[3] A. Binning. Solving second and third-order approximations to DSGE models: A recursive Sylvester equation solution. Norges Bank Working Paper 18, 2013.

[4] E. K.-W. Chu. The solution of the matrix equations $AXB - CXD = E$ and $(YA - DZ, YC - BZ) = (E, F)$. *Linear Algebra Appl.*, 93:93–105, 1987.

[5] T. Dayar. *Kronecker modeling and analysis of multidimensional Markovian systems.* Springer, Cham, 2018.

[6] E. Deadman, N. J. Higham, and R. Ralha. *Blocked Schur algorithms for computing the matrix square root*, pages 171–182. Lecture Notes in Comput. Sci. 7782. Springer, 2013.

[7] E. Elmroth, F. Gustavson, I. Jonsson, and B. Kågström. Recursive blocked algorithms and hybrid data structures for dense matrix library software. *SIAM Rev.*, 46(1):3–45, 2004.

[8] G. H. Golub and C. F. Van Loan. *Matrix computations.* Johns Hopkins University Press, Baltimore, MD, fourth edition, 2013.

[9] L. Grasedyck, D. Kressner, and C. Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitt.*, 36(1):53–78, 2013.

[10] W. Hackbusch. *Tensor Spaces and Numerical Tensor Calculus.* Springer, Heidelberg, 2012.

[11] S. Hammarling. Numerical solution of the stable, nonnegative definite Lyapunov equation. *IMA J. Numer. Anal.*, 2(3):303–323, 1982.

[12] I. Jonsson and B. Kågström. Recursive blocked algorithm for solving triangular systems. I. One-sided and coupled Sylvester-type matrix equations. *ACM Trans. Math. Software*, 28(4):392–415, 2002.

[13] I. Jonsson and B. Kågström. Recursive blocked algorithm for solving triangular systems. II. Two-sided and generalized Sylvester and Lyapunov matrix equations. *ACM Trans. Math. Software*, 28(4):416–435, 2002.

[14] O. Kamenik. Solving SDGE models: A new algorithm for the Sylvester equation. *Computational Economics*, 25(1):167–187, 2005.

[15] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[16] D. Kressner. Block variants of Hammarling's method for solving Lyapunov equations. *ACM Trans. Math. Software*, 34(1):1–15, 2008.

[17] D. Kressner and C. Tobler. Krylov subspace methods for linear systems with tensor product structure. *SIAM J. Matrix Anal. Appl.*, 31(4):1688–1714, 2010.

[18] B.-W. Li, S. Tian, Y.-S. Sun, and Z.-M. Hu. Schur-decomposition for 3D matrix equations and its application in solving radiative discrete ordinates equations discretized by Chebyshev collocation spectral method. *J. Comput. Phys.*, 229(4):1198–1212, 2010.

[19] C. D. Moravitz Martin and C. F. Van Loan. Shifted Kronecker product systems. *SIAM J. Matrix Anal. Appl.*, 29(1):184–198, 2006.

[20] C. D. Moravitz Martin and C. F. Van Loan. Solving real linear systems with the complex Schur decomposition. *SIAM J. Matrix Anal. Appl.*, 29(1):177–183, 2006/07.

[21] I. V. Oseledets. Tensor-train decomposition. *SIAM J. Sci. Comput.*, 33(5):2295–2317, 2011.

[22] E. Peise and P. Bientinesi. Algorithm 979: recursive algorithms for dense linear algebra—the ReLAPACK collection. *ACM Trans. Math. Software*, 44(2):Art. 16, 19, 2017.

[23] E. S. Quintana-Ortí and R. A. van de Geijn. Formal derivation of algorithms: the triangular Sylvester equation. *ACM Trans. Math. Software*, 29(2):218–243, 2003.

[24] G. Sangalli and M. Tani. Isogeometric preconditioners based on fast solvers for the Sylvester equation. *SIAM J. Sci. Comput.*, 38(6):A3644–A3671, 2016.

[25] V. Simoncini. Computational methods for linear matrix equations. *SIAM Rev.*, 58(3):377–441, 2016.

[26] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, Princeton, NJ, 1994.

[27] A. Touzene. Approximated tensor sum preconditioner for stochastic automata networks. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, 2006.

[28] C. F. Van Loan. The ubiquitous Kronecker product. *J. Comput. Appl. Math.*, 123(1-2):85–100, 2000.