# RECURSIVE ALGORITHMS TO UPDATE A NUMERICAL BASIS MATRIX OF THE NULL SPACE OF THE BLOCK ROW, (BANDED) BLOCK TOEPLITZ, AND BLOCK MACAULAY MATRIX[*]

CHRISTOF VERMEERSCH[†] AND BART DE MOOR[†][‡]

**Abstract.** We propose recursive algorithms to update an orthogonal numerical basis matrix of the null space of the block row, (banded) block Toeplitz, and block Macaulay matrix, which is the multivariate generalization of the (banded) block Toeplitz matrix. These structured matrices are often constructed in an iterative way and, for some applications, a basis matrix of the null space is required in every iteration. Consequently, recursively updating a numerical basis matrix of the null space, while exploiting the inherent structure of the matrices involved, induces large savings in the computation time. Moreover, we also develop a sparse adaptation of one of the recursive algorithms that avoids the explicit construction of the block Macaulay matrix and results in a considerable reduction of the required memory. We provide several numerical experiments to illustrate the proposed algorithms: for example, we solve four multiparameter eigenvalue problems via the null space of the block Macaulay matrix and notice that the recursive and sparse approach are, on average, 450 and 1300 times faster than the standard approach, respectively.

**Key words.** orthogonalization, computational methods for sparse matrices, (banded) block Toeplitz and block Macaulay matrices.

**AMS subject classifications.** 65F25, 65F50, 15B05.

## 1. Introduction.

In various engineering applications, we encounter matrices that have a particular structure, like the (banded) block Toeplitz and block Macaulay matrix. The (banded) block Toeplitz matrix emerges in several system identification and signal processing problems, where applications lead to (univariate) polynomial eigenvalue problems (PEPs). Typical examples are the stiffness and vibration analysis of large structures [11, 17, 19, 27], finite element discretizations of continuous models [16, 17, 27], and the design of MIMO filters [11, 12, 27]. A multiparameter eigenvalue problem (MEP), on the other hand, naturally gives rise to the block Macaulay matrix, which is the multivariate extension of the (banded) block Toeplitz matrix. MEPs can be found when identifying the least-squares optimal parameters of linear time-invariant (LTI) systems [7, 28], when solving partial differential equations (PDEs) via the method of separation of variables [3, 23, 24], or when reducing the model order of existing high-order models [2]. In recent work [7, 28, 30], we have exploited the structure of the null space of the (banded) block Toeplitz and block Macaulay matrix to determine the solutions of the generating PEP and MEP, respectively. Unsurprisingly, the computation of a numerical basis matrix of this null

[*]Submitted to the editors October 21, 2022.

**Funding:** This work was supported in part by the KU Leuven: Research Fund (projects C16/15/059, C3/19/053, C24/18/022, and C3/20/117), Industrial Research Fund (fellowships 13-0260, IOFm/16/004, and IOFm/20/002), and several Leuven Research & Development projects, in part by Flemish Government agencies: FWO (EOS project G0F6718N, SBO project S005319, infrastructure project I013218N, and TBM project T001919N), EWI (Flanders AI Research Program VR-21-29.01-0077), and VLAIO (City of Things COT.2018.018, Baekeland PhD mandate HBC.2019.2204, and innovation mandate HBC.2019.2209), and in part by the European Commission (ERC Adv. Grant under grant 885682). The work of Christof Vermeersch was supported by the FWO Strategic Basic Research fellowship under grant SB/1SA1319N. *(Corresponding author: Christof Vermeersch)*

[†]Center for Dynamical Systems, Signal Processing, and Data Analytics (STADIUS), Department of Electrical Engineering (ESAT), KU Leuven, Kasteelpark Arenberg 10, 3001 Leuven, Belgium (christof.vermeersch@esat.kuleuven.be, bart.demoor@esat.kuleuven.be)

[‡]Bart De Moor is a SIAM, IFAC, and IEEE fellow.

space is an important step in the solution methods. In the case of a zero-dimensional solution set (every solution of the PEP or MEP is an isolated point in the solution space), the nullity of these structured matrices reveals the total number of solutions, both affine and at infinity. Rank checks on growing submatrices of this numerical basis matrix are required to separate the affine solutions from the solutions at infinity. Since a numerical basis matrix of the null space of a (banded) block Toeplitz or block Macaulay matrix is typically a dense (i.e., non-sparse) tall matrix, it can be considered as a block row matrix, where we iterate over its subsequent (block) rows in order to determine the rank structure (i.e., we determine the change of the rank for every additional block of the numerical basis matrix)[1].

All three types of matrices considered in this paper are often constructed in an iterative way. On the one hand, the block rows of the block row matrix are considered iteratively, since a basis matrix of its null space is important in every iteration (e.g., to determine the rank structure of the block row matrix). Moreover, in many signal processing applications [1, 20, 21], new data vectors in the (block) rows are appended continuously. The process of appending new (block) rows induces the iterative structure naturally. A mature body of literature already covers the (block) row-wise updating of the singular value decomposition [6, 21] or tracking of a subspace [1, 21, 25, 26]. In this paper, we restrict ourselves to the particular subproblem where we only update in every iteration a basis matrix of the null space of the block row matrix using results from the previous iteration. On the other hand, the required size of the (banded) block Toeplitz matrix and block Macaulay matrix in system processing and system identification problems often depends on the properties of its null space. Because these properties can not be deduced in advance, we need to enlarge the (banded) block Toeplitz and block Macaulay matrix iteratively, and compute in every iteration a new numerical basis matrix of the null space. Several authors have already addressed the direct null space computation of these structured matrices [12, 18], but a recursive approach that exploits the structure and sparsity of these special matrices clearly has a lot of potential.

Therefore, in this paper, we address these questions and propose recursive[2] algorithms to update an orthogonal numerical basis matrix of the null space of the block row, (banded) block Toeplitz, and block Macaulay matrix, using results from the previous iteration. Batselier et al. [5] have developed a similar recursive algorithm to update a numerical basis matrix of the null space of the traditional (scalar) Macaulay matrix. However, they have not addressed the block Macaulay matrix, nor have they tackled block row or (banded) block Toeplitz matrices. Moreover, we also develop a sparse algorithm that avoids the explicit construction of the block Macaulay matrix and results in a considerable memory improvement compared to its dense counterparts. Exploiting the structure and sparsity of the block Macaulay matrix leads to impressive results: for example, when we use the null space of the block Macaulay matrix to solve four multiparameter eigenvalue problems, we notice that the recursive and sparse approach proposed in this paper are, on average, 450 and 1300 times faster

[1] A numerical basis matrix of the null space of the traditional (scalar) Macaulay matrix also has a block row structure [9, 29]. The recursive updating algorithm of the block row matrix proposed in this paper fits perfectly in the (scalar) Macaulay matrix approach to solve systems of multivariate polynomial equations.

[2] We do not use the term *recursion* in its strict computer science meaning ("an algorithm that calls itself on smaller input values"), but see it as an algorithm that performs the same steps on different input values ("an algorithm that uses in every iteration the same approach on new input values"), cf., the recursive least-squares algorithm.

than the standard approach, respectively.

*Outline.* The remainder of this paper proceeds as follows: In section 2 and section 3, we consider recursive algorithms to update a numerical basis matrix of the null space of the block row and (banded) block Toeplitz matrix, respectively. We develop in both sections a recursive updating algorithm, followed by a discussion of the computational complexity and several numerical experiments. We use an analogous rationale in section 4, where we discuss the null space of the block Macaulay matrix, but in this section we also consider a sparse implementation that avoids an explicit construction of the block Macaulay matrix. We close this paper by giving our conclusions and pointing at ideas for future research in section 5.

*Notation and preliminaries.* We denote scalars by italic lowercase letters, e.g., $a$, and vectors by boldface lowercase letters, e.g., $\boldsymbol{a}$. Matrices are characterized by boldface uppercase letters, e.g., $\boldsymbol{A}$. The computational complexity of an operation is given by its number of floating-point operations (FLOP). We use NULL$(\boldsymbol{A})$ and RANK$(\boldsymbol{A})$ to denote the computation via established numerical linear algebra tools of an orthogonal numerical basis matrix of the null space and the numerical rank of a matrix $\boldsymbol{A}$, respectively. $\boldsymbol{I}_{l\times l}$ is the identity matrix of size $l \times l$.

*Hardware and software.* We use for all our numerical experiments a Red Hat Enterprise Linux server infrastructure with nodes that have two Xeon Gold 6140 CPUs working at 2.3 GHz (18 Skylake cores each) and 192 GB RAM (or 768 GB RAM for the big memory nodes). The algorithms proposed in this paper are implemented in MATLAB and accessible via https://www.macaulaylab.net.

## 2. Block row matrix.

After $d$ iterations, a block row matrix $\boldsymbol{R}_d \in \mathbb{C}^{p_d \times q_d}$ consists of $d+1$ consecutive blocks[3] (or block rows) $\boldsymbol{A}_i \in \mathbb{C}^{k \times l}$:

$$\boldsymbol{R}_d = \begin{bmatrix} \boldsymbol{A}_0 \\ \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \\ \vdots \\ \boldsymbol{A}_d \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_{d-1} \\ \boldsymbol{A}_d \end{bmatrix}. \tag{2.1}$$

The block row matrix $\boldsymbol{R}_d$ has $p_d = k\,(d+1)$ rows and $q_d = l$ columns. Block row matrices appear in applications where the data only gradually becomes available (e.g., online signal processing problems) or where intermediate results are required (e.g., to determine the rank structure of the matrix). In the former situation, the desired iteration $d^*$ of the block row matrix is often not known in advance. Since the block row matrix $\boldsymbol{R}_d$ grows in every iteration $d$, also its null space changes with respect to $d$. We denote an orthogonal numerical basis matrix of the null space of $\boldsymbol{R}_d$ by $\boldsymbol{Z}_d \in \mathbb{C}^{q_d \times n_d}$, such that

$$\boldsymbol{R}_d \boldsymbol{Z}_d = \boldsymbol{0}, \tag{2.2}$$

where $n_d$ corresponds to the nullity of the block row matrix $\boldsymbol{R}_d$. Algorithm 2.1 states the problem more clearly: we extend the block row matrix $\boldsymbol{R}_d$ in an iterative way and compute a numerical basis matrix $\boldsymbol{Z}_d$ of its null space in every iteration using $\boldsymbol{Z}_{d-1}$, until we reach the desired iteration $d^*$.

[3]Although we consider in this paper consecutive blocks $\boldsymbol{A}_i$ with an equal number of rows for didactical purposes, an extension to consecutive blocks with a different number of rows is trivial and does not alter the proposed algorithm.

The standard algorithm to determine this numerical basis matrix is via the singular value decomposition and it does not consider the iterative characteristic of the problem. In subsection 2.1, we propose a recursive algorithm that uses the existing numerical basis matrix $\boldsymbol{Z}_{d-1} \in \mathbb{C}^{q_{d-1} \times n_{d-1}}$ of the null space of the block row matrix $\boldsymbol{R}_{d-1} \in \mathbb{C}^{p_{d-1} \times q_{d-1}}$ to obtain $\boldsymbol{Z}_d$. We do not assume any structure in the blocks $\boldsymbol{A}_i$ of $\boldsymbol{R}_d$, apart from the iterative construction in (2.1). Afterwards, in subsection 2.2, we asses the computational complexity of this recursive algorithm and compare it with the standard approach. Subsection 2.3 illustrates the theoretical derivations by means of some numerical experiments.

**Algorithm 2.1** Iterative null space updating problem of the block row matrix

```
Require: $\boldsymbol{A}_0, \boldsymbol{A}_1, \dots$
1: $\boldsymbol{Z}_0 \leftarrow \textsc{null}(\boldsymbol{R}_0)$ with $\boldsymbol{R}_0 = \boldsymbol{A}_0$
2: $d \leftarrow 1$
3: while $d \leq d^*$ do
4:     $\boldsymbol{R}_d \leftarrow \begin{bmatrix} \boldsymbol{R}_{d-1} \\ \boldsymbol{A}_d \end{bmatrix}$
5:     $\boldsymbol{Z}_d \leftarrow \textsc{null}(\boldsymbol{R}_d)$ via standard or recursive approach (e.g., Algorithm 2.2)
6:     $d \leftarrow d + 1$
7: end while
8: return $\boldsymbol{Z}_{d^*}$
```

**2.1. Recursive algorithm.** We consider a block row matrix $\boldsymbol{R}_{d-1} \in \mathbb{C}^{p_{d-1} \times q_{d-1}}$ after $d-1$ iterations and an orthogonal numerical basis matrix $\boldsymbol{Z}_{d-1} \in \mathbb{C}^{q_{d-1} \times n_{d-1}}$ of its null space:

$$\boldsymbol{R}_{d-1} \boldsymbol{Z}_{d-1} = \boldsymbol{0}. \tag{2.3}$$

When we append a new block $\boldsymbol{A}_d$ to obtain $\boldsymbol{R}_d$, we know that there exists an orthognal matrix $\boldsymbol{V}_d \in \mathbb{R}^{n_{d-1} \times n_d}$, so that

$$\underbrace{\begin{bmatrix} \boldsymbol{R}_{d-1} \\ \boldsymbol{A}_d \end{bmatrix}}_{\boldsymbol{R}_d} \boldsymbol{Z}_{d-1} \boldsymbol{V}_d = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{A}_d \boldsymbol{Z}_{d-1} \end{bmatrix} \boldsymbol{V}_d = \boldsymbol{0}, \tag{2.4}$$

because of (2.3). The matrix $\boldsymbol{V}_d$, on the one hand, is a basis matrix of the null space of the matrix $\boldsymbol{W}_d = \boldsymbol{A}_d \boldsymbol{Z}_{d-1} \in \mathbb{C}^{k \times n_{d-1}}$. The nullity $n_d$ of a block row matrix is at most $n_{d-1}$ because the block $\boldsymbol{A}_d$ adds (sometimes zero) linearly independent rows to $\boldsymbol{R}_d$. The matrix product $\boldsymbol{Z}_d = \boldsymbol{Z}_{d-1} \boldsymbol{V}_d = \prod_{i=0}^{d} \boldsymbol{V}_i \in \mathbb{C}^{l \times n_d}$ (with $\boldsymbol{V}_0 = \boldsymbol{Z}_0$), on the other hand, is a numerical basis matrix of the null space of $\boldsymbol{R}_d$. This insight, hence, yields a recursive algorithm to update an orthogonal numerical basis matrix of the null space of the block row matrix. Algorithm 2.2 summarizes the different steps to obtain $\boldsymbol{Z}_d$, given $\boldsymbol{A}_d$ and $\boldsymbol{Z}_{d-1}$, and fits perfectly in Algorithm 2.1.

*Importance of correct rank decisions.* In Algorithm 2.2, a correct rank decision is essential to obtain correct results. For example, in the (limit) case when we add a new block $\boldsymbol{A}_d$ of which all the rows depend linearly on the rows of the previous blocks $(\boldsymbol{A}_0, \dots, \boldsymbol{A}_{d-1})$, the numerical basis matrix of the null space of $\boldsymbol{R}_{d-1}$ also annihilates the matrix $\boldsymbol{A}_d$. Hence, $\boldsymbol{W}_d = \boldsymbol{A}_d \boldsymbol{Z}_{d-1}$ (theoretically) equals zero. When we determine $\boldsymbol{V}_d$ in Algorithm 2.2 (line 2), we should obtain an orthogonal matrix

**Algorithm 2.2** Recursive null space algorithm for the block row matrix

```
Require: $\boldsymbol{Z}_{d-1}$ and $\boldsymbol{A}_d$
1: $\boldsymbol{W}_d \leftarrow \boldsymbol{A}_d \boldsymbol{Z}_{d-1}$
2: $\boldsymbol{V}_d \leftarrow \textsc{null}\,(\boldsymbol{W}_d)$
3: $\boldsymbol{Z}_d \leftarrow \boldsymbol{Z}_{d-1} \boldsymbol{V}_d$
4: return $\boldsymbol{Z}_d$
```

of full rank $n_{d-1}$, e.g., an identity matrix. However, due to numerical floating-point errors, the matrix $\boldsymbol{W}_d$ is only close to zero and we need to be very careful when computing $\boldsymbol{V}_d$. Let us consider a rank 10 block row matrix $\boldsymbol{R}_1 \in \mathbb{R}^{40\times 20}$, which consists of two blocks $\boldsymbol{A}_0 \in \mathbb{R}^{20\times 20}$ and $\boldsymbol{A}_1 \in \mathbb{R}^{20\times 20}$ each with rank equal to 5, and a orthogonal basis matrix of its null space $\boldsymbol{Z}_1 \in \mathbb{C}^{20\times 10}$. We create a new block $\boldsymbol{A}_2 = 2\boldsymbol{A}_0 + 3\boldsymbol{A}_1 \in \mathbb{R}^{20\times 20}$ and construct $\boldsymbol{R}_2 \in \mathbb{R}^{60\times 20}$ as

$$\boldsymbol{R}_2 = \begin{bmatrix} \boldsymbol{A}_0 \\ \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_1 \\ \boldsymbol{A}_2 \end{bmatrix}. \tag{2.5}$$

Since the rows of $\boldsymbol{A}_2$ depend linearly on the rows of the first two blocks by construction, the matrix $\boldsymbol{W}_2 = \boldsymbol{A}_2\boldsymbol{Z}_1$ is close (but not exactly) zero. All singular values have the same order of magnitude and, when using a relative tolerance, the matrix $\boldsymbol{W}_2$ could be considered to be of full rank. A careful rank check in Algorithm 2.2 alleviates this problem in most situations, for example by using an additional absolute tolerance or a more advanced rank decision approach (see for example [13, 22]).

**2.2. Computational complexity.** When computing a numerical basis matrix $\boldsymbol{Z}_d$ of the null space of the block row matrix $\boldsymbol{R}_d$ via the **standard algorithm** (i.e., the singular value decomposition), we only use the singular values and right singular vectors. This takes, in iteration $d$, about $4p_d q_d^2 + 8q_d^3$ FLOP (floating-point operations) [10, p. 493]. A substitution of the number of rows and columns of $\boldsymbol{R}_d$ yields the computational complexity of the standard algorithm:

$$4kl^2(d+1) + 8l^3 = 4kl^2 d + 4kl^2 + 8l^3 = \mathcal{O}(d)\ \text{FLOP}. \tag{2.6}$$

In some applications, the blocks $\boldsymbol{A}_i$ are square, i.e., $k = l$, which simplifies (2.6):

$$4l^3 d + 12l^3 = \mathcal{O}(d)\ \text{FLOP}. \tag{2.7}$$

The proposed **recursive algorithm** consists of three main steps (see Algorithm 2.2), each with their respective number of floating-point operations:

$$\begin{aligned} 2kln_{d-1}\ \text{FLOP} &\quad \text{(multiplication – line 1)} \\ 4kn_{d-1}^2 + 8n_{d-1}^3\ \text{FLOP} &\quad \text{(null space computation – line 2)} \\ 2ln_{d-1}n_d\ \text{FLOP} &\quad \text{(multiplication – line 3)} \end{aligned}$$

The nullity $n_d$ of $\boldsymbol{R}_d$ is equal to $l - r_d \leq l = \mathcal{O}(1)$, where $r_d$ is the rank of $\boldsymbol{R}_d$. The total computational complexity of the recursive algorithm is thus bounded above by

$$6kl^2 + 10l^3 = \mathcal{O}(1)\ \text{FLOP}, \tag{2.8}$$

or when the blocks $\boldsymbol{A}_i$ are square, i.e., $k = l$, by

$$16l^3 = \mathcal{O}(1)\ \text{FLOP}. \tag{2.9}$$

Table 2.1: The computational complexity (given in FLOP per iteration $d$) of the standard and recursive approach to determine a numerical basis matrix of the null space of the block row matrix $\boldsymbol{R}_d$, for both rectangular $k \times l$ and square $l \times l$ blocks $\boldsymbol{A}_i$. The given computational complexity of the recursive approach is an upper bound and depends in practice on the rank of the blocks $\boldsymbol{A}_i$ $(i = 0, \ldots, d)$.

| **Algorithm** | **Rectangular** | **Square** |
|---|---|---|
| standard | $4kl^2d + 4kl^2 + 8l^3$ | $4l^3d + 12l^3$ |
| recursive | $6kl^2 + 10l^3$ | $16l^3$ |

When we compare the (theoretical) computational complexity of both approaches (see Table 2.1), we notice that the number of floating-point operations of the recursive algorithm remains constant with respect to the iteration $d$, while the computational complexity of the standard algorithm depends linearly on $d$. This behavior, of course, does not sound surprising, as the recursive algorithm uses results from the previous iterations and matrices of (more or less) fixed sizes, while the block row matrix $\boldsymbol{R}_d$ in the standard approach grows in every iteration.

**2.3. Numerical experiments.** We consider two experiments to illustrate the numerical properties of the recursive algorithm: a block row matrix with increasing rank (or decreasing nullity) and a block row matrix of which the rank (and also the nullity) stabilizes after $d = 10$ iterations.

**2.3.1. Block row matrix with increasing rank.** The first experiment consists of a block row matrix $\boldsymbol{R}_d \in \mathbb{R}^{100d \times 100}$, which we extend in every iteration $d$ by a random matrix[4] $\boldsymbol{A}_i \in \mathbb{R}^{100 \times 100}$ with rank $r = 2$. The rank of $\boldsymbol{R}_d$ is equal to $r_d = \max\left(2\left(d+1\right), 100\right)$. The recursive algorithm clearly outperforms the standard algorithm (see Figure 2.1), while the relative errors $\frac{\|\boldsymbol{R}_d \boldsymbol{Z}_d\|}{\|\boldsymbol{R}_d\|}$ remain stable within the same order of magnitude. As mentioned in subsection 2.2, the computation time of the standard algorithm grows linearly with the respect to $d$, while the computation time of the recursive algorithm remains more or less constant. Figure 2.1 even shows a small decrease in the computation time for higher iterations, which is mainly because of the decrease in the nullity (remember that we used the upper bound of the nullity to determine the computational complexity of the recursive algorithm, which is especially a good approximation when the number of blocks is still small).

**2.3.2. Block row matrix with stabilizing rank.** In the second experiment, we look at a block row matrix $\boldsymbol{R}_d \in \mathbb{R}^{100d \times 100}$ in which the new blocks $\boldsymbol{A}_i$ after $d = 10$ iterations are linear combinations of the previously appended blocks (i.e., $\boldsymbol{A}_0, \ldots, \boldsymbol{A}_{10}$). The rank and nullity of $\boldsymbol{R}_d$ stabilize after $d = 10$ iterations, and we notice that the computation time of the recursive algorithm (see Figure 2.2) becomes constant, i.e., the computational complexity now follows the theoretical $\mathcal{O}\left(1\right)$ FLOP.

Notice that the computation time first jumps at $d = 11$ before stabilizing. Due to the rank stabilization after 10 iterations, the matrix $\boldsymbol{W}_{11}$ is numerically zero and

[4]In order to construct a random matrix $\boldsymbol{M} \in \mathbb{R}^{p \times q}$ with a specific rank $r$, we multiply two random matrices $\boldsymbol{N} \in \mathbb{R}^{p \times r}$ and $\boldsymbol{P} \in \mathbb{R}^{r \times q}$, which have by construction a rank equal to $r$. Throughout the entire paper, we always use MATLAB's RANDN function to generate normally distributed (pseudo)random matrices.

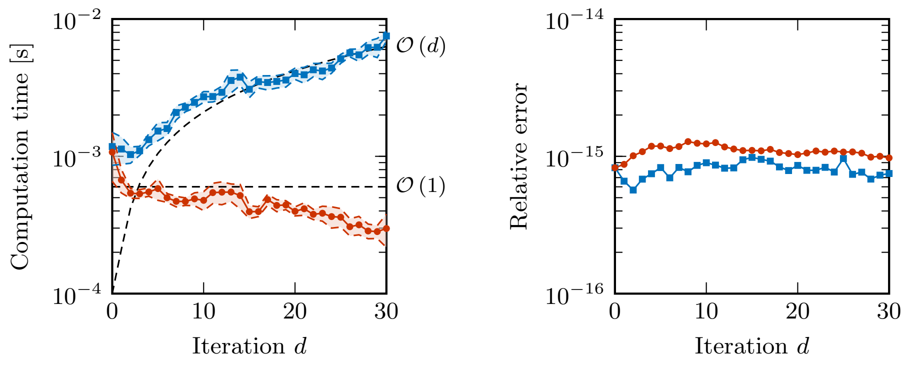


Fig. 2.1: A comparison of the mean computation time and the mean relative error $\frac{\|\boldsymbol{R}_d \boldsymbol{Z}_d\|}{\|\boldsymbol{R}_d\|}$ between the standard (■) and recursive (●) algorithm applied to a block row matrix $\boldsymbol{R}_d$, averaged over 15 experiments (the dashed lines indicate one standard deviation). In every iteration $d$, we extend the block row matrix $\boldsymbol{R}_{d-1}$ with a random block $\boldsymbol{A}_d \in \mathbb{R}^{100\times 100}$ of rank $r = 2$. The computation times of both algorithms follow the theoretical complexities (---). The computation time of the recursive algorithm decreases for higher iterations, because the input matrices become smaller in every iteration (since the nullity decreases in every iteration).

considered as a matrix of full rank, the singular value decomposition of which is computationally more expensive than of a low-rank matrix (like $\boldsymbol{W}_{10}$). This is completely in line with our earlier discussion about the importance of a correct rank decision (see subsection 2.1): when we are not careful and use wrong rank decisions, the relative error of the recursive algorithm can rise quickly. The combination of a relative and absolute tolerance avoids wrong rank decisions in this numerical experiment.

## 3. Block Toeplitz matrix.

Next, we consider the (banded[5]) block Toeplitz matrix $\boldsymbol{T}_d$, for example

$$\boldsymbol{T}_d = \left.\begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{A}_2 & \boldsymbol{0} & \boldsymbol{0} & \cdots \\ \boldsymbol{0} & \boldsymbol{A}_1 & \boldsymbol{A}_2 & \boldsymbol{0} & \cdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A}_1 & \boldsymbol{A}_2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}\right\} d+1 \text{ block rows}, \tag{3.1}$$

with seed matrices $\boldsymbol{A}_1, \boldsymbol{A}_2 \in \mathbb{C}^{k\times l}$. Block Toeplitz matrices often consist of more than two seed matrices, i.e., $\boldsymbol{A}_i \in \mathbb{C}^{k\times l}$, for $i = 1, \ldots, x+y$. Therefore, we gather (in iteration $d$) all seed matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_x$ below the block Toeplitz matrix $\boldsymbol{T}_{d-1}$ in the matrix $\boldsymbol{X} \in \mathbb{C}^{k\times s}$ and the remaining seed matrices $\boldsymbol{A}_{x+1}, \ldots, \boldsymbol{A}_{x+y}$ in the matrix $\boldsymbol{Y} \in \mathbb{C}^{k\times t}$ (with $s = lx$ and $t = ly$). We can, hence, define the block Toeplitz matrix $\boldsymbol{T}_d \in \mathbb{C}^{p_d\times q_d}$ in iteration $d$ recursively as

$$\boldsymbol{T}_d = \begin{bmatrix} \boldsymbol{T}^1_{d-1} & \boldsymbol{T}^2_{d-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{X} & \boldsymbol{Y} \end{bmatrix}, \tag{3.2}$$

[5]In the literature, this type of block Toeplitz matrices is often called banded block Toeplitz matrices, in order to make the distinction with full and circulant block Toeplitz matrices. To soften the notation in this paper, we only consider the banded block Toeplitz matrix and drop the term "banded".

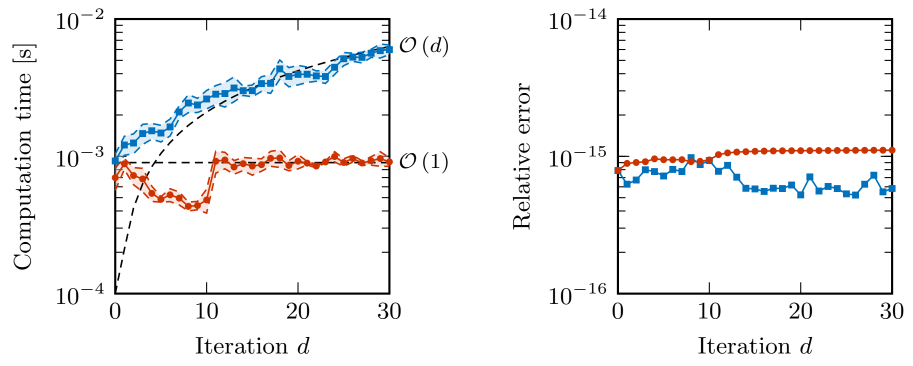


Fig. 2.2: A comparison of the mean computation time and the mean relative error $\frac{\|\boldsymbol{R}_d \boldsymbol{Z}_d\|}{\|\boldsymbol{R}_d\|}$ between the standard (−■−) and recursive (−●−) algorithm applied to a block row matrix $\boldsymbol{R}_d$, averaged over 15 experiments (the dashed lines indicate one standard deviation). In every iteration $d$, we extend the block row matrix $\boldsymbol{R}_{d-1}$ with a random block $\boldsymbol{A}_d \in \mathbb{R}^{100\times 100}$ of rank $r = 2$, until iteration $d = 10$. After 10 iterations, the newly appended blocks are linear combinations of previously added blocks, hence the computational complexity of the recursive algorithm stabilizes. The computation times of both algorithms follow the theoretical complexities (- - -). The jump in computation time at $d = 11$ for the recursive algorithm is due to the fact that the matrix $\boldsymbol{W}_{11}$ is numerically zero, hence the singular value decomposition of a full-rank instead of low-rank matrix has to be computed.

in which we partition $\boldsymbol{T}_{d-1}$ accordingly into $\boldsymbol{T}^1_{d-1} \in \mathbb{C}^{p_{d-1}\times(q_{d-1}-s)}$ and $\boldsymbol{T}^2_{d-1} \in \mathbb{C}^{p_{d-1}\times s}$. The block Toeplitz matrix $\boldsymbol{T}_d$ has $p_d$ rows and $q_d$ columns, which are given by

$$\begin{aligned} p_d &= k\,(d+1) \\ q_d &= t\,(d+1) + s = ly\,(d+1) + lx, \end{aligned} \tag{3.3}$$

which reduces in the square case with only two seed matrices (i.e., $\boldsymbol{X} = \boldsymbol{A}_1 \in \mathbb{C}^{l\times l}$ and $\boldsymbol{Y} = \boldsymbol{A}_2 \in \mathbb{C}^{l\times l}$) to

$$\begin{aligned} p_d &= l\,(d+1) \\ q_d &= l\,(d+2)\,. \end{aligned} \tag{3.4}$$

The block Toeplitz matrix contains a repetition of the same two shifted blocks $\boldsymbol{X}$ and $\boldsymbol{Y}$ in every block row of the matrix. It is very sparse and structured, in contrary to the previously discussed block row matrix. In every iteration $d$, the null space of this block Toeplitz matrix changes. When the desired iteration $d^*$ is not known in advance, a basis matrix of the null space has to be recomputed in every iteration and a recursive algorithm to do this sounds very interesting. Algorithm 3.1 sketches the problem setting.

Subsection 3.1 develops a recursive algorithm to compute an orthogonal numerical basis matrix $\boldsymbol{Z}_d \in \mathbb{C}^{q_d\times n_d}$ of the null space of $\boldsymbol{T}_d$, using $\boldsymbol{Z}_{d-1}$. In subsection 3.2 and subsection 3.3, we compare the standard and recursive algorithm via a complexity analysis and numerical experiments, respectively.

**Algorithm 3.1** Iterative null space updating of the block Toeplitz matrix

```
Require: $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{x+y}$
1: $\boldsymbol{X} \leftarrow \begin{bmatrix} \boldsymbol{A}_1 & \cdots & \boldsymbol{A}_x \end{bmatrix}$ and $\boldsymbol{Y} \leftarrow \begin{bmatrix} \boldsymbol{A}_{x+1} & \cdots & \boldsymbol{A}_{x+y} \end{bmatrix}$
2: $\boldsymbol{Z}_0 \leftarrow \text{NULL}(\boldsymbol{T}_0)$ with $\boldsymbol{T}_0 = \begin{bmatrix} \boldsymbol{X} & \boldsymbol{Y} \end{bmatrix}$
3: $d \leftarrow 1$
4: while $d \leq d^*$ do
5:     $\boldsymbol{T}_d \leftarrow \begin{bmatrix} \boldsymbol{T}_{d-1}^1 & \boldsymbol{T}_{d-1}^2 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{X} & \boldsymbol{Y} \end{bmatrix}$
6:     $\boldsymbol{Z}_d \leftarrow \text{NULL}(\boldsymbol{T}_d)$ via standard or recursive approach (e.g., Algorithm 3.2)
7:     $d \leftarrow d+1$
8: end while
9: return $\boldsymbol{Z}_{d^*}$
```

**3.1. Recursive algorithm.** We consider a block Toeplitz matrix $\boldsymbol{T}_{d-1}$ after $d-1$ iterations and an orthogonal numerical basis matrix $\boldsymbol{Z}_{d-1} \in \mathbb{C}^{q_{d-1} \times n_{d-1}}$ of its null space, with nullity $n_{d-1}$, such that

$$\boldsymbol{T}_{d-1}\boldsymbol{Z}_{d-1} = \boldsymbol{0}. \tag{3.5}$$

If we now extend $\boldsymbol{T}_{d-1}$ with $t = ly$ zero columns, then we can write

$$\begin{bmatrix} \boldsymbol{T}_{d-1} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{Z}_{d-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_{t\times t} \end{bmatrix} = \boldsymbol{0}. \tag{3.6}$$

The nullity of this extended matrix $\begin{bmatrix} \boldsymbol{T}_{d-1} & \boldsymbol{0} \end{bmatrix}$ equals $n_{d-1} + t$. If we add the next block row of the block Toeplitz matrix, i.e., we consider the block Toeplitz matrix $\boldsymbol{T}_d$, then we know that there exists an orthogonal matrix $\boldsymbol{V}_d \in \mathbb{C}^{(n_{d-1}+t)\times n_d}$, such that

$$\begin{bmatrix} \boldsymbol{T}_{d-1}^1 & \boldsymbol{T}_{d-1}^2 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{X} & \boldsymbol{Y} \end{bmatrix} \begin{bmatrix} \boldsymbol{Z}_{d-1}^1 & \boldsymbol{0} \\ \boldsymbol{Z}_{d-1}^2 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_{t\times t} \end{bmatrix} \boldsymbol{V}_d = \begin{bmatrix} \boldsymbol{T}_{d-1}\boldsymbol{Z}_{d-1} & \boldsymbol{0} \\ \boldsymbol{X}\boldsymbol{Z}_{d-1}^2 & \boldsymbol{Y} \end{bmatrix} \boldsymbol{V}_d = \boldsymbol{0}, \tag{3.7}$$

where $\boldsymbol{Z}_{d-1}$ is partitioned in accordance with $\boldsymbol{T}_{d-1}$. From the bottom part of (3.7), it follows that

$$\boldsymbol{X}\boldsymbol{Z}_{d-1}^2\boldsymbol{V}_d^1 + \boldsymbol{Y}\boldsymbol{V}_d^2 = \boldsymbol{0}, \tag{3.8}$$

where $\boldsymbol{V}_d$ is partitioned into matrices $\boldsymbol{V}_d^1 \in \mathbb{C}^{n_{d-1}\times n_d}$ and $\boldsymbol{V}_d^2 \in \mathbb{C}^{t\times n_d}$. Hence,

$$\begin{bmatrix} \boldsymbol{X}\boldsymbol{Z}_{d-1}^2 & \boldsymbol{Y} \end{bmatrix} \boldsymbol{V}_d = \boldsymbol{0}, \tag{3.9}$$

which means that the matrix $\boldsymbol{V}_d$ is a basis matrix of the null space of $\begin{bmatrix} \boldsymbol{X}\boldsymbol{Z}_{d-1}^2 & \boldsymbol{Y} \end{bmatrix}$ and

$$\underbrace{\begin{bmatrix} \boldsymbol{T}_{d-1}^1 & \boldsymbol{T}_{d-1}^2 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{X} & \boldsymbol{Y} \end{bmatrix}}_{\boldsymbol{T}_d} \underbrace{\begin{bmatrix} \boldsymbol{Z}_{d-1}^1\boldsymbol{V}_d^1 \\ \boldsymbol{Z}_{d-1}^2\boldsymbol{V}_d^1 \\ \boldsymbol{V}_d^2 \end{bmatrix}}_{\boldsymbol{Z}_d} = \boldsymbol{0}. \tag{3.10}$$

An orthogonal numerical basis matrix $\boldsymbol{Z}_d$ of $\boldsymbol{T}_d$ can be computed as

$$\boldsymbol{Z}_d = \begin{bmatrix} \boldsymbol{Z}_{d-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_{t\times t} \end{bmatrix} \boldsymbol{V}_d = \begin{bmatrix} \boldsymbol{Z}_{d-1}\boldsymbol{V}_d^1 \\ \boldsymbol{V}_d^2 \end{bmatrix}. \tag{3.11}$$

Algorithm 3.2 summarizes the different steps of this recursive algorithm.

**Algorithm 3.2** Recursive null space algorithm for the block Toeplitz matrix

```
Require: $\boldsymbol{Z}_{d-1}$, $\boldsymbol{X}$, and $\boldsymbol{Y}$
1: $\boldsymbol{W}_d \leftarrow \boldsymbol{X}\boldsymbol{Z}_{d-1}^2$
2: $\boldsymbol{V}_d \leftarrow \textsc{null}\left(\begin{bmatrix}\boldsymbol{W}_d & \boldsymbol{Y}\end{bmatrix}\right)$
3: $\boldsymbol{Z}_d \leftarrow \begin{bmatrix}\boldsymbol{Z}_{d-1}\boldsymbol{V}_d^1 \\ \boldsymbol{V}_d^2\end{bmatrix}$
4: return $\boldsymbol{Z}_d$
```

*Block banded matrix without fixed seed matrices.* Since the recursive algorithm does not explicitly make use of the repetitive structure in the block Toeplitz matrix (i.e., the same seed matrices appear in every block row), it can also be applied to tackle block banded matrices without fixed seed matrices: the matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ are different in every iteration. Subsection 3.3.4 contains a numerical experiment with such a block banded matrix.

**3.2. Computational complexity.** We determine again the computational cost of the **standard algorithm** by substituting the number of rows and columns of the block Toeplitz matrix $\boldsymbol{T}_d$ into the computational cost of computing the singular value decomposition (see subsection 2.2), i.e., $4p_dq_d^2 + 8q_d^3$ FLOP [10, p. 493]:

$$4k\,(d+1)\,(t\,(d+1)+s)^2 + 8\,(t\,(d+1)+s)^3 = \mathcal{O}\left(d^3\right) \text{ FLOP}. \tag{3.12}$$

In some applications, $\boldsymbol{T}_d$ consists of two square submatrices $\boldsymbol{X} = \boldsymbol{A}_1 \in \mathbb{C}^{l\times l}$ and $\boldsymbol{Y} = \boldsymbol{A}_2 \in \mathbb{C}^{l\times l}$, so we can simplify (3.12):

$$l^3\left(12d^3 + 68d^2 + 128d + 80\right) = \mathcal{O}\left(d^3\right) \text{ FLOP}. \tag{3.13}$$

The proposed **recursive algorithm**, on the other hand, contains three main steps (see Algorithm 3.2):

$$\begin{aligned} 2ksn_{d-1} \text{ FLOP} &\quad \text{(multiplication – line 1)} \\ 4k\,(n_{d-1}+t)^2 + 8\,(n_{d-1}+t)^3 \text{ FLOP} &\quad \text{(null space computation – line 2)} \\ 2\,(td+s)\,n_{d-1}n_d \text{ FLOP} &\quad \text{(multiplication – line 3)} \end{aligned}$$

The nullity $n_d$ of $\boldsymbol{T}_d$ with respect to the iteration $d$ is given by

$$\begin{aligned} n_d &= q_d - r_d \\ &= t\,(d+1) + s - rd \\ &= (t-r)d + t + s = \mathcal{O}(d), \end{aligned} \tag{3.14}$$

with $r_d = rd$ the rank of the block Toeplitz matrix when $\textsc{rank}\left(\begin{bmatrix}\boldsymbol{X} & \boldsymbol{Y}\end{bmatrix}\right) = r$. We assume that rank of $\begin{bmatrix}\boldsymbol{X} & \boldsymbol{Y}\end{bmatrix}$ is very close to the number of columns, i.e., $r \approx t$ and, therefore, we consider the nullity to remain almost constant with respect to the iteration $d$. The computational complexity of the recursive algorithm then corresponds to (with $n_d = n_{d-1} = t+s$)

$$2ks\,(t+s) + 4k\,(s+2t)^2 + 8\,(s+2t)^3 + 2\,(td+s)\,(t+s)^2 = \mathcal{O}\,(d) \text{ FLOP}, \tag{3.15}$$

or, for two square $l\times l$ submatrices $\boldsymbol{X}$ and $\boldsymbol{Y}$, to

$$8l^3d + 108l^3 = \mathcal{O}\,(d) \text{ FLOP}. \tag{3.16}$$

Table 3.1: The computational complexity (given in FLOP per iteration $d$) of the standard and recursive approach to compute a numerical basis matrix of the null space of the block Toeplitz matrix $\boldsymbol{T}_d$, for both the rectangular ($\boldsymbol{X} \in \mathbb{R}^{k\times s}$ and $\boldsymbol{Y} \in \mathbb{R}^{k\times t}$) and square case ($\boldsymbol{X} \in \mathbb{R}^{l\times l}$ and $\boldsymbol{Y} \in \mathbb{R}^{l\times kl}$).

| **Algorithm** | **Rectangular** | **Square** |
|---|---|---|
| standard | $4kt^2d^3 + 8t^3d^3 + \mathcal{O}\left(d^2\right)$ | $l^3\left(12d^3 + 68d^2 + 128d + 80\right)$ |
| recursive | $2t\left(s+t\right)^2 d + \mathcal{O}\left(1\right)$ | $8l^3d + 108l^3$ |

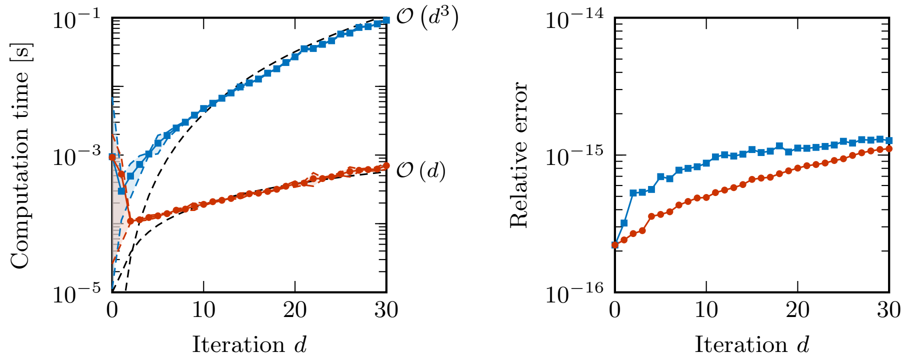


Fig. 3.1: A comparison of the mean computation cost and the mean relative error $\frac{\|\boldsymbol{T}_d\boldsymbol{Z}_d\|}{\|\boldsymbol{T}_d\|}$ between the standard (■) and recursive (●) algorithm applied to a block Toeplitz matrix $\boldsymbol{T}_d$, averaged over 15 experiments (the dashed lines indicate one standard deviation). $\boldsymbol{T}_d$ consists of two square random seed matrices $\boldsymbol{A}_1, \boldsymbol{A}_2 \in \mathbb{R}^{20\times 20}$, such that the rank $r$ of $\begin{bmatrix}\boldsymbol{A}_1 & \boldsymbol{A}_2\end{bmatrix} = \begin{bmatrix}\boldsymbol{X} & \boldsymbol{Y}\end{bmatrix}$ is equal to 16. The computation times of both algorithms follow the theoretical computational complexities (- - -).

The computational complexity of the recursive algorithm is equal to $\mathcal{O}(d)$, which is due to the dominating multiplication. If we compare this to the standard algorithm, which has a computational complexity $\mathcal{O}(d^3)$, then the recursive approach gains two orders of magnitude. Table 3.1 summarizes the computational complexities.

**3.3. Numerical experiments.** Four numerical experiments with random seed matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ illustrate the numerical properties of the recursive algorithm.

**3.3.1. Block Toeplitz matrix with high-rank seed matrices.** In the first numerical experiment, we consider a block Toeplitz matrix $\boldsymbol{T}_d$ that consists of two square seed matrices $\boldsymbol{A}_1, \boldsymbol{A}_2 \in \mathbb{R}^{20\times 20}$ with $\text{RANK}\left(\begin{bmatrix}\boldsymbol{A}_1 & \boldsymbol{A}_2\end{bmatrix}\right) = \text{RANK}\left(\begin{bmatrix}\boldsymbol{X} & \boldsymbol{Y}\end{bmatrix}\right) =$ 16 (which is close to the number of columns $l = 20$). In every iteration $d$, we compute a numerical basis matrix of the null space of this block Toeplitz matrix via the standard and recursive algorithm. Figure 3.1 visualizes the computation time and relative error for every iteration $d$. Clearly, the recursive approach outperforms the full singular value decomposition, while the relative error $\frac{\|\boldsymbol{T}_d\boldsymbol{Z}_d\|}{\|\boldsymbol{T}_d\|}$ remains more or less the same. The computation times of the standard and recursive algorithm grow cubicly and linearly with respect to the iteration $d$, respectively (as in Table 3.1).

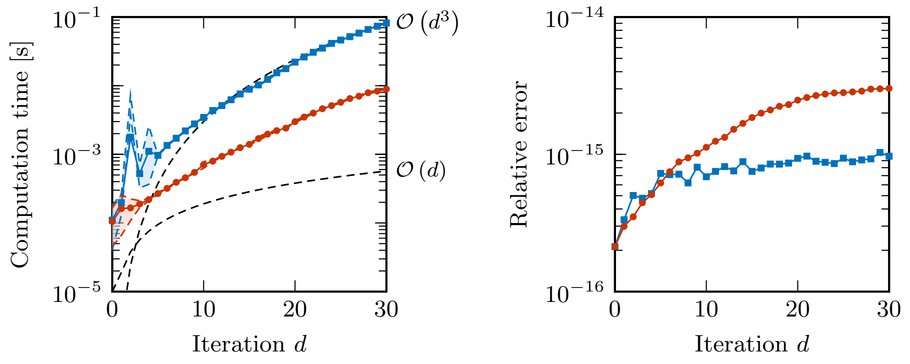


Fig. 3.2: A comparison of the mean computation cost and the mean relative error $\frac{\|\boldsymbol{T}_d\boldsymbol{Z}_d\|}{\|\boldsymbol{T}_d\|}$ between the standard (-■-) and recursive (-●-) algorithm applied to a block Toeplitz matrix $\boldsymbol{T}_d$, averaged over 15 experiments (the dashed lines indicate one standard deviation). $\boldsymbol{T}_d$ consists of two square random seed matrices $\boldsymbol{A}_1, \boldsymbol{A}_2 \in \mathbb{R}^{20\times 20}$, such that the rank $r$ of $\begin{bmatrix}\boldsymbol{A}_1 & \boldsymbol{A}_2\end{bmatrix} = \begin{bmatrix}\boldsymbol{X} & \boldsymbol{Y}\end{bmatrix}$ is equal to 3. The computation time of the recursive algorithm is higher than the theoretical computational complexity (- - -).

**3.3.2. Block Toeplitz matrix with low-rank seed matrices.** In subsection 3.2, we assume that the rank of the seed blocks of the block Toeplitz matrix $\boldsymbol{T}_d$ is quite high, which means that the nullity is almost constant with respect to the iteration $d$. When we use random low-rank seed matrices, like in Figure 3.2, we violate this assumption and we notice that the recursive algorithm takes more time than the theoretical computational complexity. However, the recursive algorithm still outperforms the standard algorithm, since the input matrices are smaller.

**3.3.3. Block Toeplitz matrix with seed matrices of different sizes.** Next, we investigate the influence of the size of the seed matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ on the computation time. In Figure 3.3, we visualize the total computation time to determine a numerical basis matrix of the null space of a block Toeplitz matrix $\boldsymbol{T}_{30}$ for desired iteration $d^* = 30$ from $d = 0$, i.e., the total computation time to iteratively reach $d^*$. We consider both the situation in which the rank $r$ of the seed matrices $\begin{bmatrix}\boldsymbol{A}_1 & \boldsymbol{A}_2\end{bmatrix} = \begin{bmatrix}\boldsymbol{X} & \boldsymbol{Y}\end{bmatrix}$ grows with the size ($r = \frac{4l}{5}$) and the situation in which the rank $r$ remains fixed $r = 16$. The computation times of the standard and recursive algorithms grow in both experiments cubicly with respect to the size of the seed matrices.

**3.3.4. Block banded matrix without fixed seed matrices.** In this example, we consider a block banded matrix $\boldsymbol{S}_d$, which consists of two different square random matrices $\boldsymbol{A}_1, \boldsymbol{A}_2 \in \mathbb{R}^{20\times 20}$ in every iteration, so that the rank $r$ of $\begin{bmatrix}\boldsymbol{A}_1 & \boldsymbol{A}_2\end{bmatrix} = \begin{bmatrix}\boldsymbol{X} & \boldsymbol{Y}\end{bmatrix}$ is equal to 16 (which is close to the number of columns $l = 20$). In every iteration $d$, we compute a numerical basis of the null space of $\boldsymbol{S}_d$ via the standard and recursive algorithm. Figure 3.4 visualizes the computation time and relative error $\frac{\|\boldsymbol{S}_d\boldsymbol{Z}_d\|}{\|\boldsymbol{S}_d\|}$ for every iteration $d$, which are very similar to Figure 3.1.

## 4. Block Macaulay matrix.

Finally, we study the null space of the block Macaulay matrix, an extension of the traditional (scalar) Macaulay matrix from resultant theory [14, 15]. The block Macaulay matrix incorporates the coefficient matrices

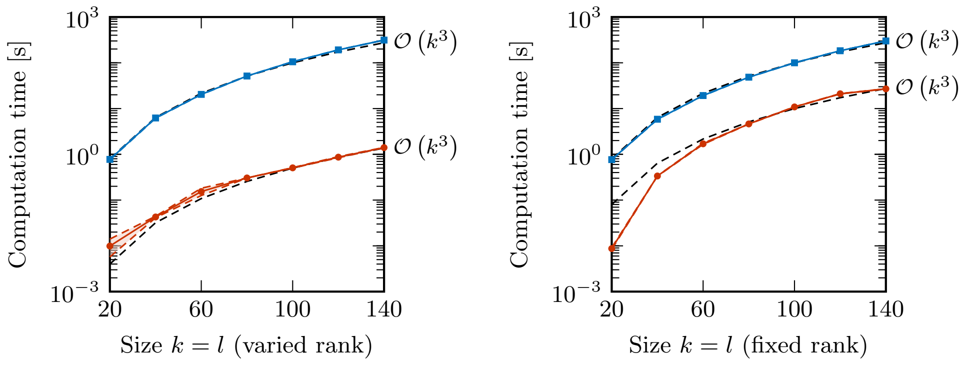


Fig. 3.3: A comparison of the total mean computation cost between the standard (■) and recursive (●) algorithm to compute a numerical basis matrix of a block Toeplitz matrix $\boldsymbol{T}_{30}$, averaged over 15 experiments (the dashed lines indicate one standard deviation). $\boldsymbol{T}_d$ consists of two square random seed matrices $\boldsymbol{A}_1, \boldsymbol{A}_2 \in \mathbb{R}^{k \times l}$. In the left figure the rank $r$ of $\begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{A}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{X} & \boldsymbol{Y} \end{bmatrix}$ grows with the size of the seed matrices as $r = \frac{4l}{5}$ (so remains high-rank), while in the right figure the rank $r$ is fixed at 16. The computation time of the standard and recursive algorithm grows in both experiments cubicly with respect to the size of the seed matrices (---).

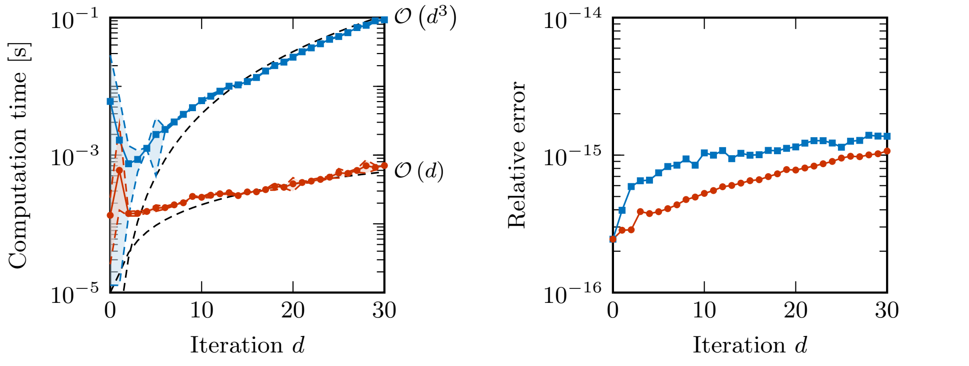


Fig. 3.4: A comparison of the mean computation cost and the mean relative error $\frac{\|\boldsymbol{S}_d \boldsymbol{Z}_d\|}{\|\boldsymbol{S}_d\|}$ between the standard (■) and recursive (●) algorithm applied to a block banded matrix $\boldsymbol{S}_d$, averaged over 15 experiments (the dashed lines indicate one standard deviation). The block banded matrix $\boldsymbol{S}_d$ consists of two different square random seed matrices $\boldsymbol{A}_1, \boldsymbol{A}_2 \in \mathbb{R}^{20 \times 20}$ in every iteration $d$, such that the rank $r$ of $\begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{A}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{X} & \boldsymbol{Y} \end{bmatrix}$ is equal to 16. The computation times of both algorithms follow the theoretical complexities of the block Toeplitz matrix (---).

of a multiparameter eigenvalue problem (MEP), which are shifted in every block row according to a particular pattern (we refer the interested reader to our previous papers in which we have introduced the block Macaulay matrix to solve MEPs [7, 28, 30]). For example, the block Macaulay matrix that incorporates the quadratic two-parameter eigenvalue problem (with eigenvalues $\alpha$ and $\beta$ and eigenvectors $\boldsymbol{z}$)

$$\left(\boldsymbol{A}_1 + \boldsymbol{A}_\alpha \alpha + \boldsymbol{A}_\beta \beta + \boldsymbol{A}_{\alpha^2}\alpha^2 + \boldsymbol{A}_{\alpha\beta}\alpha\beta + \boldsymbol{A}_{\beta^2}\beta^2\right)\boldsymbol{z} = \boldsymbol{0}, \tag{4.1}$$

looks like

$$\boldsymbol{M}_d = \begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{A}_\alpha & \boldsymbol{A}_\beta & \boldsymbol{A}_{\alpha^2} & \boldsymbol{A}_{\alpha\beta} & \boldsymbol{A}_{\beta^2} & \boldsymbol{0} & \boldsymbol{0} & \cdots \\ \boldsymbol{0} & \boldsymbol{A}_1 & \boldsymbol{0} & \boldsymbol{A}_\alpha & \boldsymbol{A}_\beta & \boldsymbol{0} & \boldsymbol{A}_{\alpha^2} & \boldsymbol{A}_{\alpha\beta} & \cdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A}_1 & \boldsymbol{0} & \boldsymbol{A}_\alpha & \boldsymbol{A}_\beta & \boldsymbol{0} & \boldsymbol{A}_{\alpha^2} & \cdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A}_1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A}_\alpha & \boldsymbol{A}_\beta & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{4.2}$$

The coefficient matrices (e.g., $\boldsymbol{A}_1$ and $\boldsymbol{A}_{\alpha^2}$) of the MEP are often referred to as the seed matrices of $\boldsymbol{M}_d$, since the MEP generates the entire block Macaulay matrix[6]. In order to keep our notation consistent throughout the entire paper, we denote the seed matrices again by a single subscript $i$, i.e., $\boldsymbol{A}_i \in \mathbb{C}^{k\times l}$ $(i = 1, \dots, x+y)$. Consequently, we can recursively define the block Macaulay matrix $\boldsymbol{M}_d \in \mathbb{C}^{p_d \times q_d}$ in iteration $d$ as

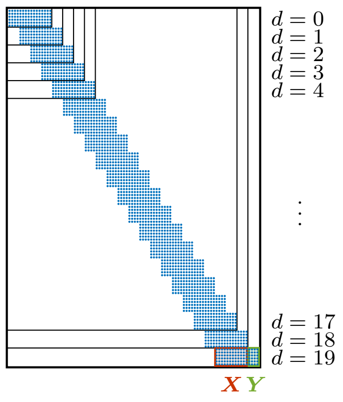
$$\boldsymbol{M}_d = \begin{bmatrix} \boldsymbol{M}_{d-1}^1 & \boldsymbol{M}_{d-1}^2 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{X}_d & \boldsymbol{Y}_d \end{bmatrix}, \tag{4.3}$$

where the matrix $\boldsymbol{X}_d \in \mathbb{C}^{m_d \times s_d}$ gathers all the seed matrices $\boldsymbol{A}_1, \dots, \boldsymbol{A}_x$ (but also some zero matrices) below $\boldsymbol{M}_{d-1}^2$ and the matrix $\boldsymbol{Y}_d \in \mathbb{C}^{m_d \times t_d}$ contains the remaining seed matrices $\boldsymbol{A}_{x+1}, \dots, \boldsymbol{A}_{x+y}$ (and also some zero matrices) under the zero block. Notice that, in contrast to the matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ of the block Toeplitz matrix, the matrices $\boldsymbol{X}_d$ and $\boldsymbol{Y}_d$ of the block Macaulay matrix depend on the iteration $d$, because every iteration adds a different number of block rows to the matrix. The sizes of the matrices $\boldsymbol{X}_d$ and $\boldsymbol{Y}_d$ depend on the number of shifts $s_{\max}$ in that particular iteration:

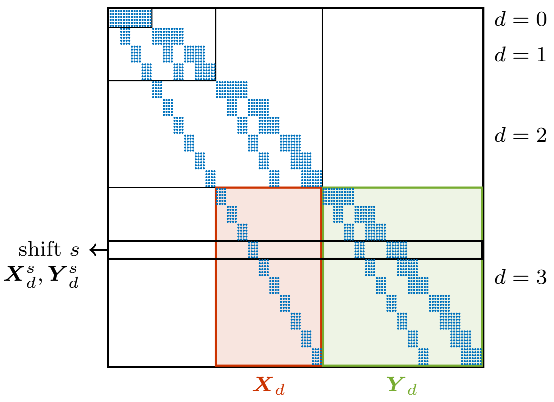
$$\begin{aligned} m_d &= k\binom{d+n-1}{n-1} = \frac{k}{(n-1)!}d^{n-1} + \mathcal{O}\left(d^{n-2}\right) \\ s_d &= l\sum_{i=0}^{d_{\mathrm{M}}-1}\binom{d+i+n-1}{n-1} = \frac{l}{(n-1)!}d^{n-1} + \mathcal{O}\left(d^{n-2}\right) \\ t_d &= l\binom{d+d_{\mathrm{M}}+n-1}{n-1} = \frac{l}{(n-1)!}d^{n-1} + \mathcal{O}\left(d^{n-2}\right), \end{aligned} \tag{4.4}$$

where $n$ is the number of eigenvalues of the generating MEP (i.e., the number of variables in the block Macaulay matrix) and $d_{\mathrm{M}}$ is the degree of the generating MEP (i.e., the highest total degree of the monomials in the MEP). The block Macaulay matrix $\boldsymbol{M}_d$ has a typical quasi-Toeplitz structure, as visualized in Figure 4.1, and its number of rows $p_d$ and columns $q_d$ grows quickly very large, due to the combinatorial

[6]We do not elaborate on the particular structure of the shifts in this paper. Essentially, every seed matrix corresponds to a particular monomial, e.g., $\alpha$, and in every iteration this monomial is multiplied by different (shift) monomials, resulting in a repeated shifted quasi-Toeplitz structure of the different seed matrices. The precise structure depends on the number of variables of the seed equation, the degree of the seed equation, and the chosen monomial ordering. A more detailed explanation can be found in [28, 30].



(a) Block Toeplitz matrix



(b) Block Macaulay matrix

Fig. 4.1: A visualization of a block Toeplitz $\boldsymbol{T}_{19}$ and block Macaulay matrix $\boldsymbol{M}_3$ ($n = 3$ and $d_{\mathrm{M}} = 1$), both with rectangular seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{6\times 4}$. Due to the combinatorial explosion of the number of shifts, the block Macaulay matrix grows quickly very large, even after much less iterations $d$ than the block Toeplitz matrix. Note that the sizes of $\boldsymbol{X}_d$ and $\boldsymbol{Y}_d$ of the block Macaulay matrix depend on $d$, while this is not the case for $\boldsymbol{X}$ and $\boldsymbol{Y}$ in the block Toeplitz matrix.

explosion of the number of shifts[7]:

$$\begin{aligned} p_d &= k\binom{d+n}{n} = \frac{k}{n!}d^n + \mathcal{O}\left(d^{n-1}\right) \\ q_d &= l\binom{d+d_{\mathrm{M}}+n}{n} = \frac{l}{n!}d^n + \mathcal{O}\left(d^{n-1}\right). \end{aligned} \tag{4.5}$$

Typically, the desired iteration $d^*$ of the block Macaulay matrix depends on the structure of its null space and is not known in advance. Hence, when we want to compute a numerical basis matrix of the null space for every iteration $d$, e.g., in order to determine the solutions of the generating multiparameter eigenvalue problem, we have to extend the block Macaulay matrix in an iterative way and recompute a numerical basis matrix of its null space in every iteration. Clearly, a recursive algorithm to update this numerical basis matrix poses itself useful in this type of practical situations. Algorithm 4.1 sketches the problem of iteratively updating the block Macaulay matrix and a numerical basis matrix of its null space.

As in the previous sections, we develop a recursive algorithm to determine an orthogonal numerical basis matrix $\boldsymbol{Z}_d$ of the null space of $\boldsymbol{M}_d$ in subsection 4.1 and determine the computational complexity afterwards in subsection 4.2. Furthermore, we also propose a sparse adaptation of the recursive algorithm in subsection 4.3. The numerical experiments in subsection 4.4 illustrate the standard, recursive, and sparse

[7]The block Macaulay matrix is the multivariate generalization of the block Toeplitz matrix, with a multiparameter eigenvalue problem (MEP) instead of a polynomial eigenvalue problem (PEP) as its seed equation, i.e., with monomials of eigenvalues instead of powers of single eigenvalues [30]. Notice that the expressions for $p_d$ and $q_d$ reduce to the block Toeplitz case of (3.3) when we consider a PEP instead of an MEP ($n = 1$, $d_{\mathrm{M}} = t$, and $s = l$).

algorithm. Afterwards, in subsection 4.5, we solve several multiparameter eigenvalue problems via the null space of the block Macaulay matrix.

**Algorithm 4.1** Iterative null space updating of the block Macaulay matrix

```
Require: $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{x+y}$
1: $\boldsymbol{Z}_0 \leftarrow \textsc{null}\,(\boldsymbol{M}_0)$
2: $d \leftarrow 1$
3: while $d \leq d^*$ do
4:     Determine $\boldsymbol{X}_d$ and $\boldsymbol{Y}_d$
5:     $\boldsymbol{M}_d \leftarrow \begin{bmatrix} \boldsymbol{M}_{d-1}^1 & \boldsymbol{M}_{d-2}^2 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{X}_d & \boldsymbol{Y}_d \end{bmatrix}$
6:     $\boldsymbol{Z}_d \leftarrow \textsc{null}\,(\boldsymbol{M}_d)$ via standard or recursive approach (e.g., Algorithm 4.2)
7:     $d \leftarrow d + 1$
8: end while
9: return $\boldsymbol{Z}_d$
```

**4.1. Recursive algorithm.** We extend the ideas of the block Toeplitz matrix to the block Macaulay matrix in this subsection. Since the block Macaulay matrix is a quasi-block Toeplitz matrix, a generalization of the recursive algorithm is quite easy. Similar to (3.7), we partition the block Macaulay matrix $\boldsymbol{M}_d$ and suppose that we have a block Macaulay matrix $\boldsymbol{M}_{d-1}$ of which we know a numerical basis matrix $\boldsymbol{Z}_{d-1}$ of its null space. As in the block Toeplitz matrix case, we can add $t_d$ zero columns at the end, multiply by an orthogonal matrix $\boldsymbol{V}_d \in \mathbb{C}^{(n_{d-1}+t_d)\times n_{d-1}}$, and obtain

$$\underbrace{\begin{bmatrix} \boldsymbol{M}_{d-1}^1 & \boldsymbol{M}_{d-1}^2 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{X}_d & \boldsymbol{Y}_d \end{bmatrix}}_{\boldsymbol{M}_d} \underbrace{\begin{bmatrix} \boldsymbol{Z}_{d-1}^1 & \boldsymbol{0} \\ \boldsymbol{Z}_{d-1}^2 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_{t\times t} \end{bmatrix}}_{\boldsymbol{Z}_d} \boldsymbol{V}_d = \begin{bmatrix} \boldsymbol{M}_{d-1}\boldsymbol{Z}_{d-1} & \boldsymbol{0} \\ \boldsymbol{X}_d \boldsymbol{Z}_{d-1}^2 & \boldsymbol{Y}_d \end{bmatrix} \boldsymbol{V}_d = \boldsymbol{0}. \tag{4.6}$$

The most important difference with (3.7) is that the matrices $\boldsymbol{X}_d$ and $\boldsymbol{Y}_d$ are now indexed by $d$ and can contain many zero blocks (see Figure 4.1b). We compute $\boldsymbol{V}_d$ again as a numerical basis matrix of a null space,

$$\begin{bmatrix} \boldsymbol{X}_d \boldsymbol{Z}_{d-1}^2 & \boldsymbol{Y}_d \end{bmatrix} \boldsymbol{V}_d = \boldsymbol{0}, \tag{4.7}$$

and construct $\boldsymbol{Z}_d \in \mathbb{C}^{q_d \times n_d}$ as

$$\boldsymbol{Z}_d = \begin{bmatrix} \boldsymbol{Z}_{d-1}\boldsymbol{V}_d^1 \\ \boldsymbol{V}_d^2 \end{bmatrix}. \tag{4.8}$$

Algorithm 4.2 summarizes the different steps of the entire recursive algorithm. An efficient implementation, of course, tries to avoid the zero blocks and uses fast multiplications that exploit structure, an improvement that is naturally incorporated in a sparse adaptation (see subsection 4.3).

*On the iteration-wise versus block row-wise implementation.* Algorithm 4.2 considers an iteration-wise growth of the block Macaulay matrix and recomputes the numerical basis matrix in an iteration-wise fashion[8]. One notices easily that the same

[8]This distinction between iteration-wise and block row-wise does not exist in the block Toeplitz matrix, since the number of block rows coincides with the number of iterations.

**Algorithm 4.2** Recursive null space algorithm for the block Macaulay matrix

```
Require: $\boldsymbol{Z}_{d-1}$, $\boldsymbol{X}_d$, and $\boldsymbol{Y}_d$
1: $\boldsymbol{W}_d \leftarrow \boldsymbol{X}_d \boldsymbol{Z}_{d-1}^2$
2: $\boldsymbol{V}_d \leftarrow \text{NULL}\left(\begin{bmatrix} \boldsymbol{W}_d & \boldsymbol{Y}_d \end{bmatrix}\right)$
3: $\boldsymbol{Z}_d \leftarrow \begin{bmatrix} \boldsymbol{Z}_{d-1}\boldsymbol{V}_d^1 \\ \boldsymbol{V}_d^2 \end{bmatrix}$
4: return $\boldsymbol{Z}_d$
```

idea could also work if the recursive approach is applied in a block row-wise fashion. Moreover, in a block row-wise fashion, the zero blocks are easier to identify and avoid. The main drawback of this alternative block row-wise implementation is the fact that, for every iteration, multiple multiplications and null space computations are necessary, which cancels the above-mentioned computational advantages (see the numerical experiment in subsection 4.4.1).

**4.2. Computational complexity.** As for the block Toeplitz matrix (but now for growing matrices $\boldsymbol{X}_d$ and $\boldsymbol{Y}_d$), we substitute the number of rows and columns of the block Macaulay matrix (4.5) into the computational cost of computing the singular values and right singular vectors, i.e., $4p_d q_d^2 + 8q_d^3$ FLOP [10, p. 493], which results in the computational cost of the **standard algorithm**:

$$\frac{4kl^2}{n!^3} d^{3n} + \frac{8l^3}{n!^3} d^{3n} + \mathcal{O}\left(d^{3n-1}\right) = \mathcal{O}\left(d^{3n}\right) \text{ FLOP}. \tag{4.9}$$

Most of the times, the seed matrices $\boldsymbol{A}_i$ are square or close to square, i.e., $k \approx l$:

$$\frac{12l^3}{n!^3} d^{3n} + \mathcal{O}\left(d^{3n-1}\right) = \mathcal{O}\left(d^{3n}\right) \text{ FLOP}. \tag{4.10}$$

he proposed **recursive algorithm** contains again three main steps (see Algorithm 4.2):

$$\begin{array}{rl} 2m_d s_d n_{d-1} \text{ FLOP} & \text{(multiplication – line 1)} \\ 4m_d \left(n_{d-1} + t_d\right)^2 + 8\left(n_{d-1} + t_d\right)^3 \text{ FLOP} & \text{(null space computation – line 2)} \\ 2q_{d-1} n_{d-1} n_d \text{ FLOP} & \text{(multiplication – line 3)} \end{array}$$

The polynomial $n_d$ describes the nullity of the block Macaulay matrix $\boldsymbol{M}_d$ with respect to the iteration $d$:

$$\begin{aligned} n_d &= q_d - r_d && (4.11)\\ &= q_d - p_d && (4.12)\\ &= \frac{l}{n!} d^n - \frac{k}{n!} d^n + \mathcal{O}\left(d^{n-1}\right) && (4.13)\\ &\leq \frac{\phi}{(n-1)!} d^{n-1} = \mathcal{O}\left(d^{n-1}\right), && (4.14) \end{aligned}$$

where we assume in (4.12) that the rank is equal to the number of rows for $d < d^*$ and introduce a factor $\phi$ (and also $\phi'$ below) in (4.14) that does not depend on the iteration $d$, but depends linearly on the size of the seed matrices (i.e., $\mathcal{O}(k, l)$). We remove the highest order terms in our upper bound, since $k \geq l$ in practical

Table 4.1: The dominant term(s) of the computational complexity (in FLOP per iteration $d$) of the standard and recursive approach to compute a numerical basis matrix of the null space of the block Macaulay matrix $\boldsymbol{M}_d$, for both rectangular $k \times l$ and square $l \times l$ seed matrices $\boldsymbol{A}_i$. We assume in these complexity numbers that the rank $r_d$ is equal to the number of rows $p_d$ of $\boldsymbol{M}_d$ for iteration $d \leq d^*$ and introduce two factors $\phi$ and $\phi'$ that do not depend on $d$.

| **Algorithm** | **Rectangular** | **Square** |
|---|---|---|
| standard | $\frac{4kl^2+8l^3}{n!^3} d^{3n}$ | $\frac{12l^3}{n!^3} d^{3n}$ |
| recursive | $\left(\frac{\phi'^3}{(n-1)!^3} + \frac{2l\phi^2}{n(n-1)!^3}\right) d^{3n-2}$ | $\frac{2k\phi^2}{n(n-1)!^3} d^{3n-2}$ |

applications (otherwise the nullity does not stabilize). The computational complexity of the recursive algorithm is then bounded above by

$$\frac{\phi'^3}{(n-1)!^3} d^{3n-3} + \frac{2l\phi^2}{n\,(n-1)!^3} d^{3n-2} = \mathcal{O}\left(d^{3n-2}\right) \text{ FLOP}, \tag{4.15}$$

which remains the same expression when $k = l$ (only the factors $\phi$ and $\phi'$ change).

The computational complexity of the recursive algorithm (per iteration $d$) corresponds to $\mathcal{O}\left(d^{3n-2}\right)$, which is due to the dominating multiplication. If we compare this to the standard singular value decomposition, which has a computational complexity $\mathcal{O}(d^{3n})$, the recursive algorithm gains two orders of magnitude. Table 4.1 summarizes the computational complexities.

**4.3. Sparse algorithm.** Although an efficient implementation of Algorithm 4.2 may exploit the structure and sparsity pattern of the block Macaulay matrix, it does not yet consider the fact that every block row contains the same generating seed matrices $\boldsymbol{A}_i$. Furthermore, since the block Macaulay matrix quickly grows very large, storing this matrix requires a considerable amount of memory. We propose in Algorithm 4.4 a sparse implementation that addresses these two shortcomings. It removes the explicit construction of the block Macaulay matrix $\boldsymbol{M}_d$ and incorporates the formation of $\boldsymbol{X}_d$ and $\boldsymbol{Y}_d$ into the recursive algorithm to build a basis matrix $\boldsymbol{Z}_d$ of the null space (the problem statement changes from Algorithm 4.1 to Algorithm 4.3). For every shift $s$ in iteration $d$, Algorithm 4.4 first determines the position of the shifted seed matrices $\boldsymbol{A}_i$ and partitions them into $\boldsymbol{X}_d^s$ and $\boldsymbol{Y}_d^s$. The blocks $\boldsymbol{X}_d^s$ yield together with the previous numerical basis matrix $\boldsymbol{Z}_{d-1}$ the matrix $\boldsymbol{W}_d$, similar to Algorithm 4.2, but now per shift, and the blocks $\boldsymbol{Y}_d^s$ result in $\boldsymbol{Y}_d$. The computation of the matrices $\boldsymbol{V}_d$ and $\boldsymbol{Z}_d$ are similar to Algorithm 4.4. At no point in this sparse algorithm, $\boldsymbol{M}_d$ is explicitly built or stored in memory, but only used implicitly through the position of its shifts.

Note that for some orderings of the monomials in the block Macaulay matrix, the structure can be exploited even further. For example, when using the degree negative lexicographic ordering (like in Figure 4.1b), $\boldsymbol{Y}_d$ always contains $\boldsymbol{Y}_{d-1}$ [4].

**4.4. Numerical experiments.** We illustrate the properties of the recursive and sparse algorithm via several numerical experiments with random seed matrices.

**4.4.1. Block Macaulay matrices with high-rank seed matrices.** In the first numerical experiment, we iteratively build a block Macaulay matrix $\boldsymbol{M}_d$ and

**Algorithm 4.3** Sparse null space updating of the block Macaulay matrix

```
Require: $\boldsymbol{A}_1, \dots, \boldsymbol{A}_{x+y}$
 1: $\boldsymbol{Z}_0 \leftarrow \textsc{null}\,(\boldsymbol{M}_0)$
 2: $d \leftarrow 1$
 3: while $d \leq d^*$ do
 4:     $\boldsymbol{Z}_d \leftarrow \textsc{sparse-null}\,(\boldsymbol{Z}_{d-1}, \boldsymbol{A}_1, \dots, \boldsymbol{A}_{x+y})$ via Algorithm 4.4
 5:     $d \leftarrow d + 1$
 6: end while
 7: return $\boldsymbol{Z}_d$
```

**Algorithm 4.4** Sparse null space algorithm for the block Macaulay matrix

```
Require: $\boldsymbol{Z}_{d-1}$, $\boldsymbol{A}_1, \dots, \boldsymbol{A}_{x+y}$
 1: for every shift $s$ of iteration $d$ ($s = 1, \dots s_{\mathtt{max}}$) do
 2:     $\textsc{col} \leftarrow$ positions of columns of $\boldsymbol{A}_1, \dots, \boldsymbol{A}_{x+y}$ at shift $s$
 3:     $\textsc{col}_x \leftarrow \textsc{col} \leq q_d$ (positions of columns of $\boldsymbol{A}_1, \dots, \boldsymbol{A}_x$ at shift $s$)
 4:     $\textsc{col}_y \leftarrow \textsc{col} > q_d$ (positions of columns of $\boldsymbol{A}_{x+1}, \dots, \boldsymbol{A}_{x+y}$ at shift $s$)
 5:     $\boldsymbol{W}_d^s \leftarrow \begin{bmatrix} \boldsymbol{A}_1 & \cdots & \boldsymbol{A}_x \end{bmatrix} \boldsymbol{Z}_{d-1}\,(\textsc{col}_x)$
 6:     $\boldsymbol{Y}_d^s\,(\textsc{col}_y) \leftarrow \begin{bmatrix} \boldsymbol{A}_{x+1} & \cdots & \boldsymbol{A}_{x+y} \end{bmatrix}$
 7: end for
 8: $\boldsymbol{W}_d \leftarrow \begin{bmatrix} \boldsymbol{W}_d^1 \\ \vdots \\ \boldsymbol{W}_d^{s_{\max}} \end{bmatrix}$ and $\boldsymbol{Y}_d \leftarrow \begin{bmatrix} \boldsymbol{Y}_d^1 \\ \vdots \\ \boldsymbol{Y}_d^{s_{\max}} \end{bmatrix}$
 9: $\boldsymbol{V}_d \leftarrow \textsc{null}\left(\begin{bmatrix} \boldsymbol{W}_d & \boldsymbol{Y}_d \end{bmatrix}\right)$
10: $\boldsymbol{Z}_d \leftarrow \begin{bmatrix} \boldsymbol{Z}_{d-1}\boldsymbol{V}_d^1 \\ \boldsymbol{V}_d^2 \end{bmatrix}$
11: return $\boldsymbol{Z}_d$
```

compute a numerical basis matrix $\boldsymbol{Z}_d$ of its null space. We consider both a linear 2-parameter eigenvalue problem with 3 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{21\times 20}$ (see Figure 4.2) and a quadratic 3-parameter eigenvalue problem with 10 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{22\times 20}$ (see Figure 4.3). As Table 4.1 indicates, we observe experimentally that we gain two orders of magnitude in the computational complexity, while the relative error $\frac{\|\boldsymbol{M}_d\boldsymbol{Z}_d\|}{\|\boldsymbol{M}_d\|}$ remains more or less the same.

**4.4.2. Block Macaulay matrix with seed matrices of different sizes and with different numbers of eigenvalues.** Next, we investigate the influence of the size of the seed matrices $\boldsymbol{A}_i$ and the number of eigenvalues on the computation time. In Figure 4.4, we visualize the total time to compute a numerical basis matrix of the null space of a block Macaulay matrix $\boldsymbol{M}_{15}$ for desired iteration $d^* = 15$ from $d = 0$, i.e., the total computation time to iteratively reach $d^*$. We consider a linear 2-parameter eigenvalue problem with 3 random seed matrices $\boldsymbol{A}_{\boldsymbol{i}} \in \mathbb{R}^{(l+1)\times l}$, where we increase the size of the seed matrices during the numerical experiment, and a linear $n$-parameter eigenvalue problem with $n+1$ random seed matrices $\boldsymbol{A}_{\boldsymbol{i}} \in \mathbb{R}^{(19+n)\times 20}$, where we increase the number of eigenvalues during the numerical experiment. The computation time grows cubicly with the number of columns of the seed matrices, while the influence of the number of eigenvalues is given in Table 4.1.

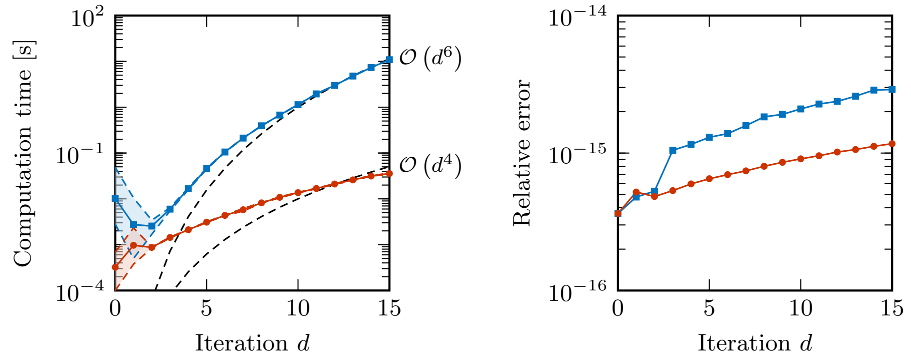


Fig. 4.2: A comparison of the mean computation time and the mean relative error $\frac{\|\boldsymbol{M}_d\boldsymbol{Z}_d\|}{\|\boldsymbol{M}_d\|}$ between the standard (-■-) and recursive (-●-) algorithm applied to a block Macaulay matrix $\boldsymbol{M}_d$, averaged over 15 experiments (the dashed lines indicate one standard deviation). The block Macaulay matrix $\boldsymbol{M}_d$ is generated by a linear 2-parameter eigenvalue problem with 3 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{21\times 20}$. The computation times of both algorithms follow the theoretical complexities of the block Macaulay matrix (- - -).

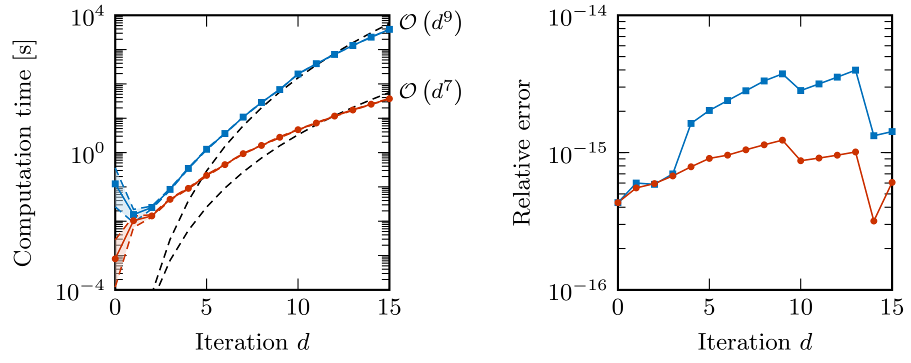


Fig. 4.3: A comparison of the mean computation time and the mean relative error $\frac{\|\boldsymbol{M}_d\boldsymbol{Z}_d\|}{\|\boldsymbol{M}_d\|}$ between the standard (-■-) and recursive (-●-) algorithm applied to a block Macaulay matrix $\boldsymbol{M}_d$, averaged over 15 experiments (the dashed lines indicate one standard deviation). The block Macaulay matrix $\boldsymbol{M}_d$ is generated by a quadratic 3-parameter eigenvalue problem with 10 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{22\times 20}$. The computation times of both algorithms follow the theoretical complexities of the block Macaulay matrix (- - -).

**4.4.3. Comparison between the iteration-wise and block row-wise algorithm.** Figure 4.5 compares the computation time and the relative error for the recursive algorithm, when applied iteration-wise and block row-wise. The results of this numerical experiment support our claim that a iteration-wise implementation of the recursive algorithm is faster than a block row-wise approach, especially when the iteration $d$ grows larger.

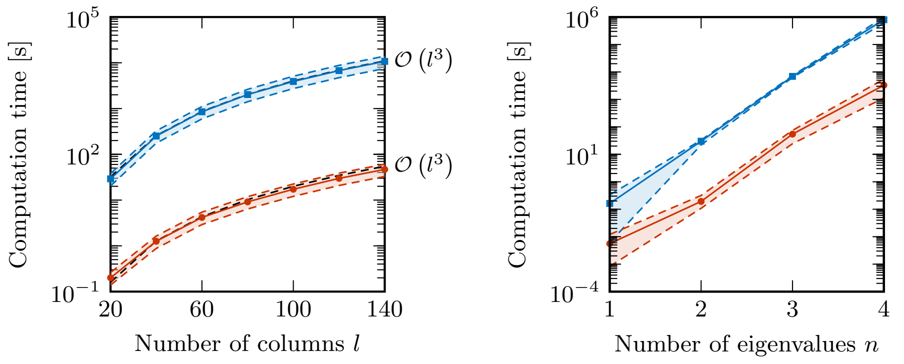


Fig. 4.4: A comparison of the total mean computation time between the standard (■) and recursive (●) algorithm applied to compute a numerical basis matrix of the null space of a block Macaulay matrix $\boldsymbol{M}_{15}$, averaged over 15 experiments (the dashed lines indicate one standard deviation). $\boldsymbol{M}_d$ is generated in the left figure by a linear 2-parameter eigenvalue problem with 3 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{(l+1)\times l}$ and in the right figure by a linear $n$-parameter eigenvalue problem with $n+1$ random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{(19+n)\times 20}$. The computation times of both algorithms follow the theoretical complexities of the block Macaulay matrix (---).

**4.4.4. Comparison between the recursive and sparse algorithm.** We repeat the experiment with a block Macaulay matrix $\boldsymbol{M}_d$ generated by a linear 2-parameter eigenvalue problem with 3 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{21\times 20}$, but we now compare the recursive and sparse algorithm. To make a fair comparison, we also include the time to build $\boldsymbol{M}_d$ (in a recursive fashion). Figure 4.6 shows that the sparse algorithm is clearly faster than the recursive approach (the construction of $\boldsymbol{M}_d$ also takes up a major part of the computation time) and is much more memory efficient.

**4.5. Solving multiparameter eigenvalue problems.** Finally, we use the proposed algorithms for their intended purpose: solving multiparameter eigenvalue problems via a numerical basis matrix of a corresponding block Macaulay matrix. The null space of a block Macaulay matrix has a special structure that we can exploit to obtain the eigentuples of the generating MEP. We do not elaborate on the details of this null space based solution approach, which we explain in-depth in [30]. It is important to know that this problem fits perfectly into the problem setting of Algorithm 4.1, where we do not know the desired iteration $d^*$ of the block Macaulay matrix because $d^*$ depends on the properties of the null space (the nullity has to stabilize at the total number of solutions).

**4.5.1. Random multiparameter eigenvalue problems.** We solve four different MEPs: a linear 2-parameter eigenvalue problem with 3 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{41\times 40}$ (Table 4.2), a linear 3-parameter eigenvalue problem with 4 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{22\times 20}$ (Table 4.3), a cubic 2-parameter eigenvalue problem with 10 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{11\times 10}$ (Table 4.4), and a quadratic 3-parameter eigenvalue problem with 10 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{12\times 10}$ (Table 4.5). Clearly, the computation times of the recursive and sparse approaches are much smaller than the

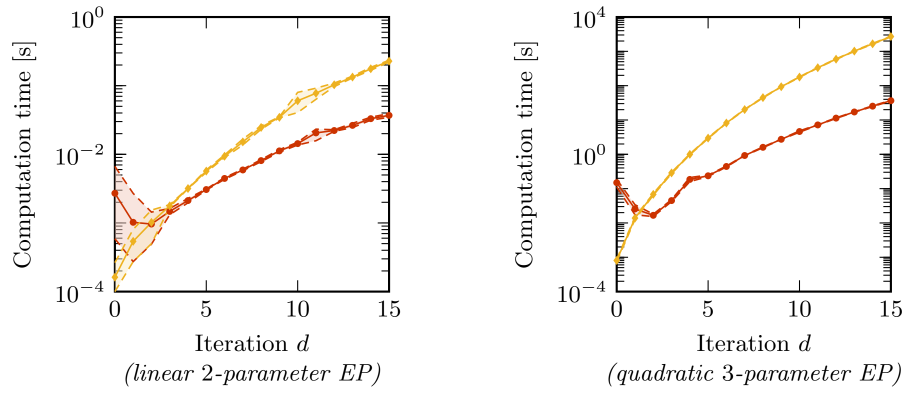


Fig. 4.5: A comparison of the mean computation time of a block Macaulay matrix, averaged over 15 experiments (the dashed lines indicate one standard deviation), when we apply the recursive algoritm iteration-wise (—●—) and block row-wise (—◆—). The block Macaulay matrix $\boldsymbol{M}_d$ is generated in the left figure by a linear 2-parameter eigenvalue problem with 3 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{21\times 20}$ and in the right figure by a quadratic 3-parameter eigenvalue problem with 10 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{22\times 20}$.

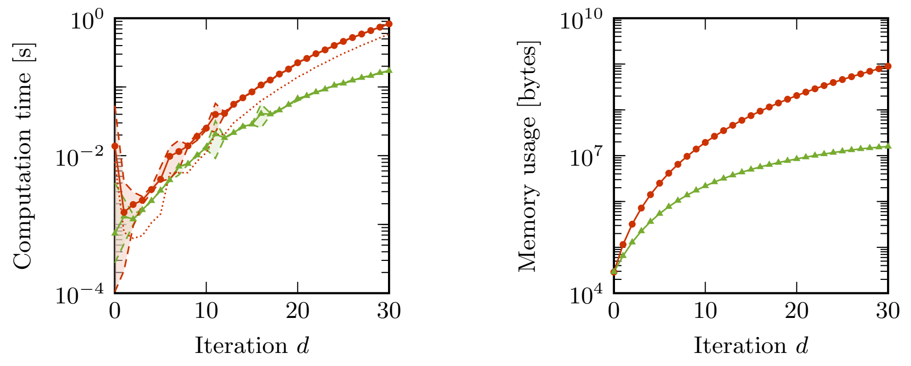


Fig. 4.6: A comparison of the mean computation time and the memory usage between the recursive (—●—) and sparse (—▲—) algorithm applied to a block Macaulay matrix $\boldsymbol{M}_d$, averaged over 15 experiments (the dashed lines indicate one standard deviation). $\boldsymbol{M}_d$ is generated by a linear 2-parameter eigenvalue problem with 3 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{21\times 20}$. The explicit recursive construction of $\boldsymbol{M}_d$ (⋯⋯) takes up a major part of the computation time of the recursive algorithm.

time to solve the MEPs via the standard approach, while the residual errors[9] of the solutions are more or less the same: for example, we notice that the recursive and sparse approach proposed in this paper are, on average, 450 and 1300 times faster than the standard approach, respectively. Moreover, the computation time required to

[9]The residual error corresponds to the norm of the residual vector after substituting the computed eigenvalues and eigenvectors in the MEP.

Table 4.2: A comparison between the standard, recursive, and sparse algorithm to solve a linear 2-parameter eigenvalue with 3 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{41\times 40}$. The table contains the total computation time to build a numerical basis matrix of the null space of the corresponding block Macaulay matrix (requires 41 iterations), the total memory usage to obtain this basis matrix, and the residual error of the solutions.

| **Algorithm** | **Comp. time** | **Memory usage** | **Residual error** |
|---:|:---:|:---:|:---:|
| standard (last iter.) | 168 630 s (25 771 s) | 11.46 GB | $3.9\times10^{-15}$ |
| recursive | 126.48 s | 11.46 GB | $1.8\times10^{-14}$ |
| sparse | 41.12 s | 0.25 GB | $1.8\times10^{-14}$ |

Table 4.3: A comparison between the standard, recursive, and sparse algorithm to solve a linear 3-parameter eigenvalue with 4 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{19\times 17}$. The table contains the total computation time to build a numerical basis matrix of the null space of the corresponding block Macaulay matrix (requires 28 iterations), the total memory usage to obtain this basis matrix, and the residual error of the solutions.

| **Algorithm** | **Comp. time** | **Memory usage** | **Residual error** |
|---:|:---:|:---:|:---:|
| standard (last iter.) | 21 421 s (8315 s) | 5.50 MB | $3.9\times10^{-15}$ |
| recursive | 128.50 s | 5.50 MB | $1.8\times10^{-14}$ |
| sparse | 104.07 s | 0.21 MB | $1.8\times10^{-14}$ |

perform the last iteration with the standard approach takes more time than the total computation time of the recursive and sparse approach. Hence, even if we know the desired iteration $d^*$ in advance, a recursive (or sparse) approach may still be better. The sparse approach has the additional advantage of requiring much less memory.

**4.5.2. Least-squares realization problem.** We solve an MEP that arises from a least-squares realization problem with $N = 7$ random data points: given a data sequence $y_0, \ldots, y_6$ ($\boldsymbol{y} \in \mathbb{R}^{7\times 1}$), find the adapted data sequence $\hat{y}_0, \ldots, \hat{y}_6$ so that the misfit $\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2$ is minimized and $\hat{\boldsymbol{y}} \in \mathbb{R}^{7\times 1}$ is the output of a second-order autonomous model [7, 8]:

$$\hat{y}_k = \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{x}_0, \tag{4.16}$$

where $\boldsymbol{x}_0 \in \mathbb{R}^{2\times 1}$ is the initial state, $\boldsymbol{A} \in \mathbb{R}^{2\times 2}$ is the system matrix, and $\boldsymbol{C} \in \mathbb{R}^{1\times 2}$ is the output vector. In [7], it has been shown how this identification problem corresponds to a quadratic two-parameter eigenvalue problem

$$\begin{aligned}\boldsymbol{\mathcal{M}}(\lambda_1,\lambda_2)\,\boldsymbol{z} = \big(&\boldsymbol{A}_{00} + \boldsymbol{A}_{10}\lambda_1 + \boldsymbol{A}_{01}\lambda_2 + \\ &\boldsymbol{A}_{20}\lambda_1^2 + \boldsymbol{A}_{11}\lambda_1\lambda_2 + \boldsymbol{A}_{02}\lambda_2^2\big)\boldsymbol{z} = \boldsymbol{0},\end{aligned} \tag{4.17}$$

where the coefficient matrices $\boldsymbol{A}_{\boldsymbol{\omega}} \in \mathbb{R}^{17\times 16}$ are as described in [7].

This problem has a positive-dimensional solution set at infinity, so the nullity of the block Macaulay matrix does not stabilize. In order to solve this system identification problem, we need to check in every iteration if the basis matrix of the block Macaulay matrix contains the solutions. For this specific problem, we need $d = 28$

Table 4.4: A comparison between the standard, recursive, and sparse algorithm to solve a cubic 2-parameter eigenvalue with 10 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{11\times 10}$. The table contains the total computation time to build a numerical basis matrix of the null space of the corresponding block Macaulay matrix (requires 33 iterations), the total memory usage to obtain this basis matrix, and the residual error of the solutions.

| **Algorithm** | **Comp. time** | **Memory usage** | **Residual error** |
|---:|:---:|:---:|:---:|
| standard (last iter.) | 805.57 s (144.74 s) | 377.59 MB | $3.9 \times 10^{-13}$ |
| recursive | 3.54 s | 377.59 MB | $1.8 \times 10^{-13}$ |
| sparse | 1.21 s | 26.56 MB | $1.8 \times 10^{-13}$ |

Table 4.5: A comparison between the standard, recursive, and sparse algorithm to solve a quadratic 3-parameter eigenvalue with 10 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{12\times 10}$. The table contains the total computation time to build a numerical basis matrix of the null space of the corresponding block Macaulay matrix (requires 23 iterations), the total memory usage to obtain this basis matrix, and the residual error of the solutions.

| **Algorithm** | **Comp. time** | **Memory usage** | **Residual error** |
|---:|:---:|:---:|:---:|
| standard (last iter.) | 48 363 s (15 720 s) | 8.64 GB | $3.9 \times 10^{-15}$ |
| recursive | 184.34 s | 8.64 GB | $1.8 \times 10^{-14}$ |
| sparse | 130.45 s | 0.46 GB | $1.8 \times 10^{-14}$ |

iterations to build a $7395 \times 7936$ block Macaulay matrix (i.e., total degree of highest monomial is equal to 30) that has a null space with the correct solutions of the problem. Table 4.6 compares the computation time and maximum residual error of the different algorithms. The recursive and sparse algorithm are also much faster than the standard algorithm in the case of this system identification problem, while resulting in more or less the same residual errors. In these practical problems, the proposed algorithms allow us to tackle problems that are much larger than possible with the standard algorithm.

## 5. Conclusions and future work.

In this paper, we presented recursive algorithms to update a numerical basis matrix of the null space of the block row, (banded) block Toeplitz, and block Macaulay matrix. These recursive algorithms use the numerical basis matrix computed during the previous iteration in order to efficiently determine an update. Furthermore, we also proposed a sparse alternative for the block Macaulay matrix, without explicitly constructing this large block Macaulay matrix. We provided several numerical experiments to illustrate the properties of these four algorithms and to compare them with the standard full singular value decomposition. The numerical experiments, like the least-square realization problem, motivated the need for faster algorithms: the proposed recursive (and sparse) algorithms clearly outperformed the standard approach.

The recursive approach and sparse adaptation have given us the opportunity to solve larger multiparameter eigenvalue problems (MEPs) than possible with the standard approach. In the future, we will consider memory-efficient implementations

Table 4.6: A comparison between the standard, recursive, and sparse algorithm to solve a least-squares realization problem with $N = 7$ data points, which corresponds to a quadratic 2-parameter eigenvalue with 6 random seed matrices $\boldsymbol{A}_i \in \mathbb{R}^{17\times 16}$. The table contains the total computation time to build a numerical basis matrix of the null space of the corresponding block Macaulay matrix (requires 28 iterations), the total memory usage to obtain this basis matrix, and the residual error of the solutions.

| **Algorithm** | **Comp. time** | **Memory usage** | **Residual error** |
|---:|:---:|:---:|:---:|
| standard (last iter.) | 1049.23 s (219.28 s) | 508.84 MB | $5.1 \times 10^{-10}$ |
| recursive | 22.24 s | 508.84 MB | $4.9 \times 10^{-10}$ |
| sparse | 19.05 s | 34.36 MB | $1.4 \times 10^{-9}$ |

and improve our current algorithms to further push the limits. Analogue approaches for Hankel and block Hankel matrices could also be useful in other application areas. Furthermore, we want to translate our efforts from the singular value decomposition to the QR-decomposition, enabling column space based solution approaches for MEPs.

## REFERENCES

[1] G. B. Adams, M. F. Griffin, and G. W. Stewart, *Direction-of-arrival estimation using the rank-revealing URV decomposition*, in Proc. of the 1991 International Conference on Acoustics, Speech, and Signal Processing (ICASSP 91), Toronto, Canada, 1991, pp. 1385–1388.

[2] M. O. Agudelo, C. Vermeersch, and B. De Moor, *Globally optimal $\mathcal{H}_2$-norm model reduction: A numerical linear algebra approach*, IFAC-PapersOnLine, 54 (2021), pp. 564–571. Part of special issue: 24th International Symposium on Mathematical Theory of Networks and Systems (MTNS).

[3] F. V. Atkinson and A. B. Mingarelli, *Multiparameter Eigenvalue Problems: Sturm–Liouville Theory*, CRC Press, Boca Raton, FL, USA, 2010.

[4] K. Batselier, P. Dreesen, and B. De Moor, *The geometry of multivariate polynomial division and elimination*, SIAM Journal on Matrix Analysis and Applications, 34 (2013), pp. 102–125.

[5] K. Batselier, P. Dreesen, and B. De Moor, *A fast recursive orthogonalization scheme for the Macaulay matrix*, Journal of Computational and Applied Mathematics, 267 (2014), pp. 20–32.

[6] J. R. Bunch and C. P. Nielsen, *Updating the singular value decomposition*, Numerische Mathematik, 31 (1978), pp. 111–129.

[7] B. De Moor, *Least squares realization of LTI models is an eigenvalue problem*, in Proc. of the 18th European Control Conference (ECC), Naples, Italy, 2019, pp. 2270–2275.

[8] B. De Moor, *Least squares optimal realisation of autonomous LTI systems is an eigenvalue problem*, Communications in Information and Systems, 20 (2020), pp. 163–207.

[9] P. Dreesen, K. Batselier, and B. De Moor, *Multidimensional realisation theory and polynomial system solving*, International Journal of Control, 91 (2018), pp. 2692–2704.

[10] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, USA, 4th ed., 2013.

[11] N. J. Higham, S. D. Mackey, and F. Tisseur, *The conditioning of linearizations of matrix polynomials*, SIAM Journal of Matrix Analysis and Applications, 28 (2006), pp. 1005–1028.

[12] G. T. Krishna, I. Singh, and K. Giridhar, *Null-space of block convolution matrix*, in Proc. of the 2013 National Conference on Communications (NCC), New Delhi, India, 2013, pp. 1–5.

[13] S. Kritchman and B. Nadler, *Non-parametric detection of the number of signals: Hypothesis testing and random matrix theory*, IEEE Transactions on Signal Processing, 57 (2009), pp. 3930–3941.

[14] F. S. Macaulay, *Some formulae in elimination*, Proc. of the London Mathematical Society, 1 (1902), pp. 3–27.

[15] F. S. Macaulay, *The Algebraic Theory of Modular Systems*, vol. 19 of Cambridge Tracts in

Mathematics and Mathematical Physics, Cambridge University Press, London, UK, 1916.

[16] S. D. Mackey, N. Mackey, C. Mehl, and V. Mehrmann, *Structured polynomial eigenvalue problems: Good vibrations from good linearizations*, SIAM Journal on Matrix Analysis and Applications, 28 (2006), pp. 1029–1051.

[17] S. D. Mackey, N. Mackey, and F. Tisseur, *Polynomial eigenvalue problems: Theory, computation, and structure*, in Numerical Algebra, Matrix Theory, Differential-Algebraic Equations and Control Theory, P. Benner, M. Bollhöfer, D. Kressner, C. Mehl, and T. Stykel, eds., Springer, Cham, Switzerland, 2015, pp. 319–348.

[18] N. Mastronardi, M. Van Barel, and R. Vandebril, *On the computation of the null space of Toeplitz-like matrices*, Electronic Transactions on Numerical Analysis, 33 (2009), pp. 151–162.

[19] V. Mehrmann and H. Voss, *Nonlinear eigenvalue problems: A challenge for modern eigenvalue methods*, GAMM-Mitteilungen, 27 (2004), pp. 121–152.

[20] M. Moonen, B. De Moor, L. Vandenberghe, and J. Vandewalle, *On- and off-line identification of linear state space models*, International Journal of Control, 49 (1989), pp. 219–232.

[21] M. Moonen, P. Van Dooren, and J. Vandewalle, *A singular value decomposition updating algorithm for subspace tracking*, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 1015–1038.

[22] P. O. Perry and P. J. Wolfe, *Minimax rank estimation for subspace tracking*, IEEE Journal of Selected Topics in Signal Processing, 4 (2010), pp. 504–513.

[23] B. Plestenjak, C. I. Gheorghiu, and M. E. Hochstenbach, *Spectral collocation for multiparameter eigenvalue problems arising from separable boundary value problems*, Journal of Computational Physics, 298 (2015), pp. 585–601.

[24] B. D. Sleeman, *Multiparameter spectral theory and separation of variables*, Journal of Physics A: Mathematical and Theoretical, 41 (2007), pp. 1–20.

[25] G. W. Stewart, *An updating algorithm for subspace tracking*, IEEE Transactions on Signal Processing, 40 (1992), pp. 1535–1541.

[26] G. W. Stewart, *Updating a rank-revealing ULV decomposition*, SIAM Journal on Matrix Analysis and Applications, 14 (1993), pp. 494–499.

[27] F. Tisseur and K. Meerbergen, *The quadratic eigenvalue problem*, SIAM Review, 43 (2001), pp. 235–286.

[28] C. Vermeersch and B. De Moor, *Globally optimal least-squares ARMA model identification is an eigenvalue problem*, IEEE Control Systems Letters, 3 (2019), pp. 1062–1067.

[29] C. Vermeersch and B. De Moor, *A column space based approach to solve systems of multivariate polynomial equations*, IFAC-PapersOnLine, 54 (2021), pp. 137–144. Part of special issue: 24th International Symposium on Mathematical Theory of Networks and Systems (MTNS).

[30] C. Vermeersch and B. De Moor, *Two complementary block Macaulay matrix algorithms to solve multiparameter eigenvalue problems*, Linear Algebra and its Applications, 654 (2022), pp. 177–209.