Randomized Rank-Structured Matrix Compression by Tagging

Katherine J. Pearce* Anna Yesypenko† James Levitt‡ Per-Gunnar Martinsson§

Abstract

In this work, we present novel randomized compression algorithms for flat rank-structured matrices with shared bases, known as uniform Block Low-Rank (BLR) matrices. Our main contribution is a technique called tagging, which improves upon the efficiency of basis matrix computation while preserving accuracy compared to alternative methods. Tagging operates on the matrix using matrix-vector products of the matrix and its adjoint, making it particularly advantageous in scenarios where accessing individual matrix entries is computationally expensive or infeasible.

Flat rank-structured formats use subblock sizes that asymptotically scale with the matrix size to ensure competitive complexities for linear algebraic operations, making alternative methods prohibitively expensive in such scenarios. In contrast, tagging reconstructs basis matrices using a constant number of matrix-vector products followed by linear post-processing, with the constants determined by the rank parameter and the problem's underlying geometric properties.

We provide a detailed analysis of the asymptotic complexity of tagging, demonstrating its ability to significantly reduce computational costs without sacrificing accuracy. We also establish a theoretical connection between the optimal construction of tagging matrices and projective varieties in algebraic geometry, suggesting a hybrid numeric-symbolic avenue of future work.

To validate our approach, we apply tagging to compress uniform BLR matrices arising from the discretization of integral and partial differential equations. Empirical results show that tagging outperforms alternative compression techniques, significantly reducing both the number of required matrix-vector products and overall computational time. These findings highlight the practicality and scalability of tagging as an efficient method for flat rank-structured matrices in scientific computing.

1 Introduction

In scientific computing and data science, many applications involve matrices that are dense but "data-sparse," admitting certain low-rank approximations that compress the matrices while preserving their critical information. Algorithms to compress data-sparse matrices can achieve better performance by invoking *rank structure*, where the input matrices are tessellated into blocks that are either small enough in size to apply dense algorithms or are of low numerical rank. Not only can rank-structured matrices be stored and applied to vectors efficiently, but often they can also be approximately inverted or LU-factorized in linear or close-to-linear time.

There are many rank-structured matrix formats that have been successfully utilized in engineering and data science applications. These formats are classified as either hierarchical [8, 9, 25, 12, 18, 21] or flat [3, 4, 6, 29, 32, 45], corresponding to either nested or non-nested matrix tessellations, respectively. Both hierarchical and flat rank-structured matrices are further characterized by either a weak or strong admissibility criterion. In the weakly admissible formats, every off-diagonal block is said to be admissible, or treated as low-rank, while the admissible blocks of strongly admissible rank-structured matrices correspond only to the "far-field" of a given matrix block (as in, e.g., the Fast-Multipole Method [22]).

*Department of Mathematics & Oden Institute, University of Texas at Austin (katherine.pearce@austin.utexas.edu).
†Oden Institute, University of Texas at Austin (anna@oden.utexas.edu).
‡Oden Institute, University of Texas at Austin (jlevitt@utexas.edu).
§Department of Mathematics & Oden Institute, University of Texas at Austin (pgm@oden.utexas.edu).

The last key defining feature of rank-structured matrices has historically only applied to hierarchical formats, namely whether all admissible blocks in the same block-row or block-column are well-approximated by the same low-rank basis matrices. This property does not hold for the $\mathcal{H}$-matrix format originally proposed in [25] which necessitates that separate basis matrices be computed for each admissible block. However, for many applications, basis matrix computations can be accelerated without significantly impacting the accuracy of the low-rank approximation by employing a shared basis assumption, where the row or column spaces of all admissible blocks within the same block-row or block-column are spanned by the same basis. This shared basis assumption characterizes the uniform $\mathcal{H}^1$-matrix [25, 38] and $\mathcal{H}^2$-matrix [24, 17] formats in the strong admissibility setting, while the HSS [12, 52] and HBS [18, 35] matrix formats make use of shared bases under a weak admissibility condition.

Despite the compelling analysis [4, 5, 29] and broad applicability of the flat block low-rank (BLR) format [3] in areas such as sparse direct solvers [5, 48], modeling [1, 11], and boundary integral equations [2], there has not been much formal investigation into shared basis matrices for flat matrix formats until the recent work of [6]. (It is noted in [6] that the construction of HBS matrices in [18] relies upon a flat format utilizing shared bases, but only as an intermediary step to the HBS format.) As such, *uniform BLR matrices*, as we will refer to them throughout this work in connotation of their connection to uniform $\mathcal{H}^1$-matrices, have not been given a thorough treatment in the existing literature of rank-structured matrices, specifically in the area of randomized rank-structured matrix compression.

Though the BLR format has enjoyed much practical success and performance optimization [1, 11, 30, 48, 50], its utility in many applications has not yet been explored, particularly those in which the matrix entries cannot be directly accessed. Rather, in these black-box problems, it is assumed that we can only interact with the input matrix through some fast algorithm to quickly evaluate matrix-vector products. In these applications, the goal is often matrix "reconstruction" in terms of low-rank basis matrices, which enables downstream matrix operations (e.g. inversion or LU factorization) and simplifies operations involving products of rank-structured matrices. These compressed matrix representations have broad applicability in scientific computing, for instance in deriving rank-structured representations of integral operators [44, 55] or accelerating sparse direct solvers [41, 56, 54].

Randomized algorithms have proven to be very effective in handling the black-box problem environment, particularly the method of *randomized sketching*, in which the row and column spaces of the input matrix are approximated by analyzing how the matrix and its transpose act on tall thin matrices drawn from random matrix distributions [26, 35, 38, 43]. Namely, suppose that $\mathbf{A} \in \mathbb{R}^{N\times N}$ has some given rank structure where the numerical ranks of admissible blocks are upper bounded by $k \ll N$, but that $\mathbf{A}$ is only accessible through some fast black-box algorithm. In order words, given tall thin $\mathbf{\Omega}, \mathbf{\Psi} \in \mathbb{R}^{N\times r}$, $r = \mathcal{O}(k)$, we can quickly evaluate $\mathbf{Y} = \mathbf{A\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$. Our goal is then to reconstruct $\mathbf{A}$ as efficiently as possible, using only the information in the set $\{\mathbf{Y}, \mathbf{\Omega}, \mathbf{Z}, \mathbf{\Psi}\}$, in a particular rank-structured matrix representation.

Recently in [35], Levitt and Martinsson introduced the first fully black-box linear-complexity randomized algorithm to compress (weakly admissible) HBS matrices, inspired by the "peeling algorithm" of [38] as well as its improvement in [36]. The "block nullification" algorithm of [35] requires only $\mathcal{O}(k)$ matrix-vector products (with modest pre-factor) and $\mathcal{O}(k^2N)$ floating point operations to compress an $N \times N$ HBS matrix. However, to compress flat rank structure formats, linear sampling complexity is not attainable, and in the setting of strong admissibility, the larger pre-factor in computing basis matrices with block nullification presents a significant drawback.

In this manuscript, we present a modification of the algorithm of [35] for strongly admissible uniform BLR matrices. We then introduce a new randomized compression algorithm for uniform BLR matrices based on a method we refer to as *tagging*, in which null spaces of small submatrices are computed to exclude contributions from inadmissible blocks in the random sketches $\mathbf{Y}$ and $\mathbf{Z}$. As in the block nullification method, the strategic exclusion of inadmissible blocks permits the same sample matrices $\mathbf{Y}$ and $\mathbf{Z}$ to be used to compress every admissible block-row and block-column of the input matrix, which can be straightforwardly parallelized for optimized performance. Additionally, our tagging method has a smaller asymptotic prefactor than block nullification, and its sampling complexity for basis matrix computation is independent of the problem size even in flat formats, improving on the performance of block nullification without significantly impacting the accuracy of the approximation.

Contributions We propose two new randomized compression schemes for strongly admissible flat rank-structured matrices with shared bases, termed uniform BLR. The first scheme is based on our extension of the block nullification method in [35] to (hierarchical or flat) rank-structured matrices with shared bases under a strong admissibility criterion. The second randomized compression scheme is our main contribution, which introduces the novel method of tagging for basis matrix computation in the compression of uniform BLR matrices. We draw a theoretical connection between the tagging matrices in our method and Plücker coordinates in projective space that would guarantee optimal performance, before presenting a practical alternative to compute tagging matrices that is more computationally efficient and works well in practice. We provide detailed analysis of the asymptotic complexities of both schemes, and we empirically compare their performances in compressing strongly admissible uniform BLR matrices that arise in discretizations of boundary integral equations and sparse direct solvers to demonstrate the superior computational efficiency of our method.

Outline This manuscript is structured as follows. Section 2 covers the necessary linear algebra preliminaries for our work, as well as background on the uniform BLR matrix format used to illustrate our methods. Section 3 illustrates our modification of the block nullification method of [35] for basis matrix computations and analyzes the associated asymptotic complexity. Section 4 describes the new method of tagging for basis matrix computations including detailed complexity analysis, and Section 5 outlines the theoretical connection between tagging and projective varieties, as well as computational strategies to generate good quality random sketches with tagging in practice. Finally, Section 6 finishes the compression procedure for uniform BLR matrices.

2 Preliminaries

In this section, we briefly summarize the necessary background for randomized compression of the rank-structured matrices considered in our work. We follow the presentation of [40] for the requisite linear algebra material, and we provide a synopsis of the block low-rank matrix format introduced in [6].

2.1 Notation

A vector $\mathbf{x} \in \mathbb{R}^n$ is measured by the Euclidean norm $\|\mathbf{x}\| = \left(\sum_i |x_i|^2\right)^{\frac{1}{2}}$, and a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ is equipped with the corresponding operator norm $\|\mathbf{A}\| = \sup_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\|$. We let $[m]$ denote the integers $1, 2, \ldots, m$. We adopt the notation of Golub and Van Loan [20] to reference submatrices; namely, if $\mathbf{A}$ is an $m \times n$ matrix, and $I = [i_1, i_2, \ldots, i_k] \subset [m]$ and $J = [j_1, j_2, \ldots, j_l] \subset [n]$ are (row and column, resp.) index sets, then $\mathbf{A}(I, J)$ denotes the $k \times l$ matrix

$$\mathbf{A}(I,J) = \begin{bmatrix} \mathbf{A}(i_1,j_1) & \mathbf{A}(i_1,j_2) & \ldots & \mathbf{A}(i_1,j_l) \\ \mathbf{A}(i_2,j_1) & \mathbf{A}(i_2,j_2) & \ldots & \mathbf{A}(i_2,j_l) \\ \vdots & \vdots & & \vdots \\ \mathbf{A}(i_k,j_1) & \mathbf{A}(i_k,j_2) & \ldots & \mathbf{A}(i_k,j_l) \end{bmatrix}.$$

The abbreviation $\mathbf{A}(I, :)$ is used to designate the submatrix $\mathbf{A}(I, [n])$, and $\mathbf{A}(:, J)$ is defined analogously. The (Hermitian) transpose of $\mathbf{A}$ is given by $\mathbf{A}^*$, and a matrix $\mathbf{U}$ is said to be *orthonormal* if its columns are orthonormal, $\mathbf{U}^*\mathbf{U} = \mathbf{I}$.

2.2 The QR factorization

Every $m \times n$ matrix $\mathbf{B}$ has a (full) QR factorization of the form

$$\underset{m\times n}{\mathbf{B}} = \underset{m\times m}{\mathbf{Q}} \quad \underset{m\times n}{\mathbf{R}}, \tag{1}$$

where $\mathbf{Q}$ is orthonormal and $\mathbf{R}$ is upper-triangular. If $\mathbf{B}$ has rank k and its first k columns are linearly independent, then it has a rank-k partial QR factorization given by

$$\underset{m\times n}{\mathbf{B}_k} = \underset{m\times k}{\mathbf{Q}_k} \quad \underset{k\times n}{\mathbf{R}_k},$$

where $\mathbf{Q}_k$ is orthonormal and $\mathbf{R}_k$ is upper-triangular.

2.3 Functions for orthonormal bases

For a matrix $\mathbf{B}$ of rank at most k, we denote a function that returns a matrix $\mathbf{Q}$ with k orthonormal columns spanning the column space of $\mathbf{B}$ by

$$\mathbf{Q} = \texttt{col}(\mathbf{B}, k),$$

which can be implemented by truncating a full QR factorization to obtain a rank-k partial QR factorization.

For a matrix $\mathbf{B}$ with a null space of at least dimension k, we denote a function that returns a matrix $\mathbf{Z}$ with k orthonormal columns in the null space of $\mathbf{B}$ by

$$\mathbf{Z} = \texttt{null}(\mathbf{B}, k),$$

which can be implemented by selecting the last k columns of the factor $\mathbf{Q}$ in the full QR factorization of $\mathbf{B}^*$.

Remark 2.1 *If the first k columns of a matrix $\mathbf{B}$ have smaller rank than* $\text{rank}(\mathbf{B})$*, then the first k columns of $\mathbf{Q}$ produced by an unpivoted QR factorization algorithm might not span the column space of $\mathbf{B}$. There is a similar concern for the linear dependence of rows when computing a basis for the null space. Because we only apply* `col` *and* `null` *to matrices $\mathbf{B}$ that are random matrices or products involving random matrices, any subset of k rows or columns will have the same rank as $\mathbf{B}$ as long as* $\text{rank}(\mathbf{B}) \leq k$*; thus, we can can safely rely on unpivoted QR factorizations in the functions* `col` *and* `null`*.*

2.4 Randomized range-finding

Let $\mathbf{B}$ is an $m \times n$ matrix that can be accurately approximated by a rank-k matrix, and suppose we seek a matrix whose columns form an approximate orthornomal basis (ON-basis) for the column space of $\mathbf{B}$. Often referred to as range-finding, we want to determine an orthonormal matrix $\mathbf{Q}$ such that $\|\mathbf{B} - \mathbf{Q}\mathbf{Q}^*\mathbf{B}\|$ is small. We can accomplish this task efficiently through *randomized sketching*, where the column space of $\mathbf{B}$ is approximated by analyzing how $\mathbf{B}$ acts on matrices drawn from random matrix distributions [33, 31, 26, 51]. In general, the randomized range-finding algorithm proceeds as follows:

1. Choose a small integer p representing how much "oversampling" is done ($p = 10$ is often sufficient).
2. Draw an $n \times (k+p)$ random matrix $\mathbf{G}$ (e.g. Gaussian [33, 26], randomized Fourier transform [10, 37]).
3. Form the $m \times (k+p)$ random sketch $\mathbf{Y} = \mathbf{B}\mathbf{G}$.
4. Compute $\mathbf{Q} = \texttt{col}(\mathbf{Y}, k)$.

We note that each column of $\mathbf{Y}$ is a random linear combination of the columns of $\mathbf{B}$, and the probability of obtaining an accurate column space approximation of $\mathbf{B}$ with the column space of $\mathbf{Y}$ approaches 1 rapidly as p increases; notably, this probability depends only on p (not on m or n, or any other properties of $\mathbf{B}$); cf. [26] and [42, Section 11].

2.5 Block Low-Rank (BLR) Matrices

This manuscript focuses on the randomized compression of $N \times N$ matrices $\mathbf{A}$ that admit a *block low-rank (BLR)* format. BLR matrices are tessellated into b row and b column blocks according to a "flat" (vs. hierarchical) rank structure. An example of a BLR matrix is illustrated in Figure 1.

Typically, a strong admissibility condition is assumed for BLR matrices, as opposed to weak admissibility where every off-diagonal block is treated as low-rank, since the asymptotic complexity is the same as it is with a weak admissibility condition for flat formats [6]. Under a strong admissibility condition, the matrix blocks that correspond to the $\leq 3^d$ neighbors of a given box in a d-dimensional geometry are also treated as full rank.

Figure 1: Tessellation of a strongly admissible BLR matrix with $b = 8$ block-rows and block-columns. Low-rank blocks are shown in gray. The blocks that are not treated as low-rank are shown in red.

2.5.1 Uniform BLR matrices

In hierarchically rank-structured formats where all levels of the index tree are considered in compression, the computation of orthonormal matrices $\mathbf{U}$ and $\mathbf{V}$, whose columns form approximate bases of the column and row spaces of the input matrix, respectively, has been accelerated by nested or shared basis assumptions.

To illustrate, consider an $\mathcal{H}^1$-matrix $\mathbf{A}$ [25, 38], a hierarchical matrix which is characterized by each admissible block $\mathbf{A}_{\ell,m} := \mathbf{A}(I_\ell, I_m)$ having its own basis matrices $\mathbf{U}_{\ell,m}$ and $\mathbf{V}_{\ell,m}$ in a rank-k representation:

$$\underbrace{\mathbf{A}_{\ell,m}}_{m\times m} = \underbrace{\mathbf{U}_{\ell,m}}_{m\times k}\underbrace{\widetilde{\mathbf{A}}_{\ell,m}}_{k\times k}\underbrace{\mathbf{V}_{\ell,m}}_{k\times m}. \tag{2}$$

The matrices $\mathbf{U}_{\ell,m}$ and $\mathbf{V}_{\ell,m}$ in (2) can be computed via

$$\begin{aligned} \mathbf{U}_{\ell,m} &= \texttt{col}\left(\mathbf{A}_{\ell,m}, k\right), \\ \mathbf{V}_{\ell,m} &= \texttt{col}\left(\mathbf{A}^*_{\ell,m}, k\right), \end{aligned} \tag{3}$$

resulting in a total of $\mathcal{O}(b^2)$ basis matrix computations for $\mathcal{O}(b^2)$ admissible blocks.

In contrast, uniform $\mathcal{H}^1$-matrices [36, 38] have the property that low-rank blocks in the same block-row or block-column share basis matrices (cf. Figure 2), resulting in a total of $\mathcal{O}(b)$ basis matrix computations for $\mathcal{O}(b^2)$ admissible blocks. However, compressing the flat analog of uniform $\mathcal{H}^1$-matrices has not been thoroughly investigated since the recent introduction of this format by Ashcraft et al. in [6]. To this end, throughout this work we refer to BLR matrices with shared bases as *uniform BLR* matrices.

2.5.2 Obtaining compressed representations of uniform BLR matrices

More formally, a uniform BLR matrix $\mathbf{A}$ is a flatly tessellated rank-structured matrix for which low-rank blocks within the same block-row or block-column share the same bases of their row or column spaces. Figure 2 illustrates this property which characterizes uniform BLR matrix, using the BLR matrix from Figure 1.

To compress (strongly admissible) uniform BLR matrices, we compute basis matrices $\mathbf{U}_\ell$ and $\mathbf{V}_m$ for

low-rank block $\mathbf{A}_{\ell,m}$ such that

$$\underbrace{\mathbf{A}_{\ell,m}}_{m\times m} = \underbrace{\mathbf{U}_\ell}_{m\times k}\underbrace{\widetilde{\mathbf{A}}_{\ell,m}}_{k\times k}\underbrace{\mathbf{V}_m^*}_{k\times m}, \tag{4}$$

where

$$\begin{aligned}\mathbf{U}_\ell &= \texttt{col}\left(\mathbf{A}(I_\ell, I_{i\in\mathcal{F}_\ell}), k\right),\\ \mathbf{V}_m &= \texttt{col}\left(\mathbf{A}^*(I_{i\in\mathcal{F}_m}, I_m), k\right).\end{aligned} \tag{5}$$

Here, if $\mathcal{N}_\ell$ denotes the set of neighbors of box ℓ (so $|\mathcal{N}_\ell| \leq 3^d$), let $\mathcal{F}_\ell = [b] \setminus \mathcal{N}_\ell$ denote the set complement of $\mathcal{N}_\ell$ in $[b]$, called the far-field of box ℓ, and similarly for $\mathcal{F}_m$. As before, the shared bases assumption used in (5) reduces the number of basis matrices required for compression, as compared to (3), from $\mathcal{O}(b^2)$ to $\mathcal{O}(b)$ for a $b \times b$ flat tessellation.

When (4) holds, we obtain a block factorization of an $N \times N$ uniform BLR matrix $\mathbf{A}$ with b blocks in each block-row and block-column, each block of size $m \times m$ (letting $N = bm$ for notational convenience):

$$\underset{bm\times bm}{\mathbf{A}} = \underset{bm\times bk}{\mathbf{U}}\ \underset{bk\times bk}{\widetilde{\mathbf{A}}}\ \underset{bk\times bm}{\mathbf{V}}^* + \underset{bm\times bm}{\mathbf{B}}, \tag{6}$$

where

$$\begin{aligned}\mathbf{U} &= \operatorname{diag}(\mathbf{U}_1, \ldots, \mathbf{U}_b),\\ \mathbf{V} &= \operatorname{diag}(\mathbf{V}_1, \ldots, \mathbf{V}_b),\end{aligned} \tag{7}$$

$$\widetilde{\mathbf{A}} = \mathbf{U}^*\mathbf{A}\mathbf{V}, \tag{8}$$

and $\mathbf{B}$ is a block-sparse matrix defined (for strongly admissible $\mathbf{A}$) as

$$\mathbf{B}_{i,j} = \begin{cases}\mathbf{A}_{i,j} - \mathbf{U}_i\widetilde{\mathbf{A}}_{i,j}\mathbf{V}_j^*, & i \in \mathcal{N}_j\\ \mathbf{0}, & \text{otherwise.}\end{cases} \tag{9}$$

The matrix $\mathbf{B}$ given by (9) represents a discrepancy term, corresponding to the "remainder" of the inadmissible blocks of $\mathbf{A}$ after their components spanned by the basis matrices have been peeled off [38, 35].

In general, compression of a uniform BLR matrix can be accomplished through the following steps[1]:

(I) Compute basis matrices $\mathbf{U}$ and $\mathbf{V}$.

(II) Compute matrix $\tilde{\mathbf{A}} = \mathbf{U}^*\mathbf{A}\mathbf{V}$.

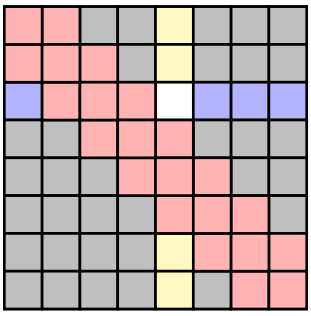
(III) Compute discrepancy matrix $\mathbf{B} = \mathbf{A} - \mathbf{U}\widetilde{\mathbf{A}}\mathbf{V}^*$.

The primary focus of this work is step (I): we develop and compare randomized algorithms for computing basis matrices of uniform BLR matrices under strong admissibility conditions. However, for completeness, we outline in Section 6 how the random sketches used for step (I) can be recycled for steps (II) and (III). First, we describe the two algorithms that we use to compute basis matrices: the existing method of block nullification in Section 3 and our new method of tagging in Section 4.

3 Block Nullification in Uniform BLR Matrix Compression

In this section, we present a modification of the previous work of [35] to develop a linear randomized compression algorithm for hierarchically block-separable (HBS) rank-structured matrices. This compression algorithm utilizes "block nullification" to form random sketches of admissible matrix blocks; these sketches are then used to compute basis matrices in an HBS representation according to the randomized rangefinding procedure of Section 2.4. The algorithm is also fully black-box, so that steps (I)-(III) above can be

[1]Steps (II) and (III) are interchangeable depending on the chosen compression algorithm; see Section 6 for details.



For example, to compute $\mathbf{A}_{3,5} = \mathbf{U}_3 \widetilde{\mathbf{A}}_{3,5} \mathbf{V}_5^*$ *(white)*:

$$\mathbf{U}_3 = \texttt{col}\left(\left[\ \mathbf{A}_{3,1}\ ,\ \mathbf{A}_{3,5}\ ,\ \mathbf{A}_{3,6}\ ,\ \mathbf{A}_{3,7}\ ,\ \mathbf{A}_{3,8}\ \right], k\right)$$

$$\mathbf{V}_5 = \texttt{col}\left(\left[\ \mathbf{A}_{1,5}\ ;\ \mathbf{A}_{2,5}\ ;\ \mathbf{A}_{3,5}\ ;\ \mathbf{A}_{7,5}\ ;\ \mathbf{A}_{8,5}\ \right]^*, k\right)$$

$$\Rightarrow \widetilde{\mathbf{A}}_{3,5} = \mathbf{U}_3^* \mathbf{A}_{3,5} \mathbf{V}_5.$$

Figure 2: Computing basis matrices for a strongly-admissible uniform BLR matrix using $\mathbf{A}$ from Figure 1.

accomplished without access to individual matrix entries. Rather, it assumes access to fast black-box matrix multiplication, so that sample matrices $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\boldsymbol{\Psi}$ can be formed efficiently given tall thin random test matrices $\boldsymbol{\Omega}, \boldsymbol{\Psi} \in \mathbb{R}^{N \times s}$ for $s = \mathcal{O}(k)$ for block-rank k.

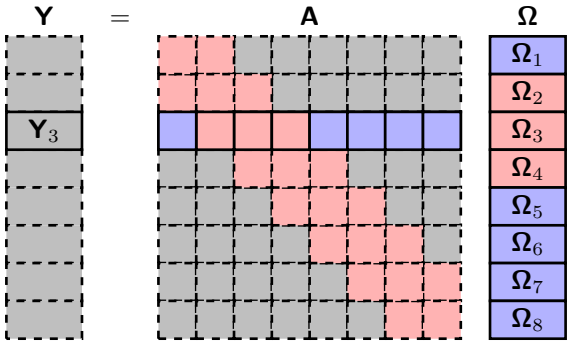
The goal is then to "reconstruct" $\mathbf{A}$ via steps (I)-(III) using only the matrices $\mathbf{Y}, \mathbf{Z}, \boldsymbol{\Omega}$, and $\boldsymbol{\Psi}$ (computed a priori) by the randomized rangefinder procedure in Section 2.4. However, each row of $\mathbf{Y}$, for instance, is a random linear combination of all columns of $\mathbf{A}$ within a given block-row, including the columns belonging to inadmissible blocks. Block nullification yields "clean" random sketches from $\mathbf{Y}$ and $\mathbf{Z}$ by excluding contributions from inadmissible blocks, without repeatedly applying $\mathbf{A}$ or $\mathbf{A}^*$ to tailored random test matrices that individually sample admissible blocks in each block-row or block-column.

While block nullification is suitable for flat or hierarchical rank-structured formats, its performance has only been investigated for hierarchically block-separable (HBS) matrices. As such, we first modify the block nullification procedure in this section to accommodate uniform BLR matrices under a strong admissibility condition. We then discuss its asymptotic complexity to emphasize that block nullification yields a larger pre-factor than our proposed technique in the next section.

3.1 Block nullification for strongly-admissible uniform BLR matrices

We begin with an illustrative example of the block nullification technique applied to the strongly-admissible uniform BLR matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ from Figure 1, flatly tessellated into b block-rows and block-columns each of size $m \times m$, with admissible blocks of rank k. Let $r = k + p$ for a small oversampling parameter (e.g. $p = 10$), and let $\boldsymbol{\Omega}, \boldsymbol{\Psi} \in \mathbb{R}^{N \times s}$ be Gaussian test matrices with $s \geq \max(r + 3m, 3r)$.

Suppose that we want to compute the basis matrix $\mathbf{U}_3 \in \mathbb{R}^{m \times k}$ for the block-row of $\mathbf{A}$ as in Figure 2, now using the sketch $\mathbf{Y}$ as in the randomized rangefinder procedure. Consider a random sketch of the form

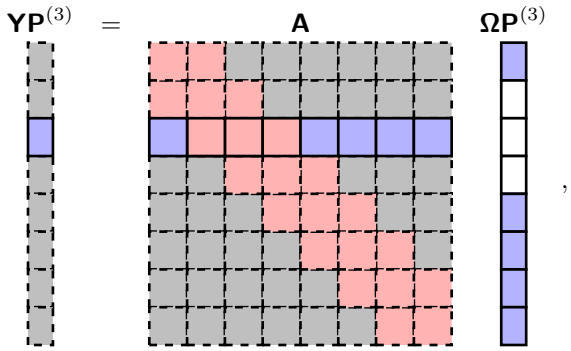


where $\mathbf{Y}_3 = \mathbf{Y}(I_3, :)$, and the test matrix blocks $\boldsymbol{\Omega}_i = \boldsymbol{\Omega}(I_i, :)$ for $i = 1, \ldots, 8$ are color-coded according to their respective block-factors in the block-row $\mathbf{A}_3 = \mathbf{A}(I_3, :)$. As before, the blue blocks of $\mathbf{A}$ are the admissible blocks in $\mathbf{A}_3$ whose columns will be approximately spanned by $\mathbf{U}_3$ computed with $\mathbf{Y}_3$, whereas the red blocks of $\mathbf{A}$ are inadmissible; our goal is to exclude their contributions from the randomized sample of $\mathbf{A}_3$ held in $\mathbf{Y}_3$.

Since $\mathbf{\Omega}^{(3)} := \mathbf{\Omega}([I_2, I_3, I_4], :) = [\mathbf{\Omega}_2|\mathbf{\Omega}_3|\mathbf{\Omega}_4]^*$ is of size $3m \times s$, it has a nullspace of dimension at least $s - 3m \geq r$. We then compute a set of r orthonormal vectors in its null space:

$$\underset{s\times r}{\mathbf{P}^{(3)}} = \texttt{null}(\mathbf{\Omega}^{(3)}, r) := \texttt{null}(\begin{bmatrix}\mathbf{\Omega}_2\\ \mathbf{\Omega}_3\\ \mathbf{\Omega}_4\end{bmatrix}, r) \quad \Rightarrow \quad \mathbf{\Omega}^{(3)}\mathbf{P}^{(3)} = \begin{bmatrix}\mathbf{\Omega}_2\\ \mathbf{\Omega}_3\\ \mathbf{\Omega}_4\end{bmatrix}\mathbf{P}^{(3)} = \underset{3m\times r}{\mathbf{0}}. \tag{10}$$

Thus, we can obtain the desired sample of $\mathbf{A}_3$ inexpensively from $\mathbf{Y}$ via $\mathbf{Y}_3\mathbf{P}^{(3)}$, shown below in blue:



noting that the white blocks of $\mathbf{\Omega}\mathbf{P}^{(3)}$ are filled with zeros. We also note that the blue blocks of $\mathbf{\Omega}\mathbf{P}^{(3)}$ contain standard Gaussian entries because (1) the distribution of Gaussian matrices is invariant under unitary transformations and (2) the matrix $\mathbf{P}^{(3)}$ is computed independently of the blue blocks of $\mathbf{\Omega}$. The desired basis matrix can then be computed via

$$\mathbf{U}_3 = \texttt{col}(\mathbf{Y}_3\mathbf{P}^{(3)}, k)$$

with the usual probabilistic guarantees (cf. Section 2.4 and [26, 42]).

In general, the method of block nullification computes basis matrices $\mathbf{U}, \mathbf{V}$ according to Algorithm 1 to accomplish step (I) of the randomized compression of uniform BLR matrices as in (6). Quickly summarizing, we first draw independent Gaussian matrices $\mathbf{\Omega}$ and $\mathbf{\Psi}$ to form random sketches $\mathbf{Y}$ and $\mathbf{Z}$ (lines 1-3). For any block $i = 1, \ldots, b$, we define $\mathbf{\Omega}^{(i)}$ as the rows of $\mathbf{\Omega}$ indexed by $\{I_j\}_{j\in N_i}$, or all I_j such that j is a neighbor of block i. We then compute r orthonormal vectors in the null space of $\mathbf{\Omega}^{(i)}$, which comprise the columns of $\mathbf{P}^{(i)}$. The matrix $\mathbf{Y}_i = \mathbf{Y}(I_i, :)$ is right-multiplied by $\mathbf{P}^{(i)}$ to compute $\mathbf{U}_i$, whose k columns form an approximate basis for the column space of $\mathbf{A}(I_i, :)$ excluding inadmissible blocks (lines 5-6). Analogously, for each block i, we compute the basis matrix $\mathbf{V}_i$ whose column space approximates the column space of $\mathbf{A}^*(I_i, :)$ excluding inadmissible blocks (lines 7-8). We discuss the asymptotic complexity of Algorithm 1 in the next section.

3.2 Asymptotic complexity of block nullification

We analyze the asymptotic complexity of Algorithm 1 by following its steps and quantifying the computational costs. Let $r = k+p$, where k is the block rank and p is the oversampling parameter, and let $s = 3^d m+r$, with m denoting the block size. For the purpose of generality, we assume that $\mathbf{A}$ is not necessarily self-adjoint. There are savings of a factor of 2 when the matrix is self-adjoint.

- Gaussian matrix generation (lines 2-3). Generating the random test matrices $\mathbf{\Omega}$ and $\mathbf{\Psi}$ requires sampling $2Ns$ values from the standard Gaussian distribution. The cost of this step is $2Ns \times T_{\text{rand}}$, where T_{rand} represents the time to sample one value.
- Matrix-vector products (line 3). Forming the sketches $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$ involves s matrix-vector multiplications for both $\mathbf{A}$ and $\mathbf{A}^*$. This contributes $2s \times T_{\text{mult}}$ to the overall complexity, where T_{mult} is the cost of applying $\mathbf{A}$ or $\mathbf{A}^*$ to a vector.

Algorithm 1 Block Nullification for Basis Construction

Require: Fast matrix-vector multiplication with uniform BLR $\mathbf{A} \in \mathbb{C}^{N\times N}$ and $\mathbf{A}^* \in \mathbb{C}^{N\times N}$, $b\times b$ flat matrix tessellation, maximum block-size m, d-dimensional geometry

Ensure: $\mathbf{U}, \mathbf{V} \in \mathbb{C}^{N\times bk}$ in uniform BLR representation of $\mathbf{A}$ as in (6)

1: Set $r = k + p$ and $s \geq \max(r + 3^d m, 3^d r)$
2: Draw independent Gaussian matrices $\mathbf{\Omega}, \mathbf{\Psi} \in \mathbb{R}^{N\times s}$
3: Sketch $\mathbf{Y} = \mathbf{A\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$
4: **for** blocks $i = 1, \ldots, b$ **do**
5: Compute $\mathbf{P}^{(i)} = \texttt{null}\left(\mathbf{\Omega}^{(i)},\ r\right)$
6: Compute $\mathbf{U}_i = \texttt{col}\left(\mathbf{Y}_i\mathbf{P}^{(i)},\ k\right)$
7: Compute $\mathbf{Q}^{(i)} = \texttt{null}\left(\mathbf{\Psi}^{(i)},\ r\right)$
8: Compute $\mathbf{V}_i = \texttt{col}\left(\mathbf{Z}_i\mathbf{Q}^{(i)},\ k\right)$
9: **end for**
10: Set $\mathbf{U} = \text{diag}\left(\mathbf{U}_1, \ldots, \mathbf{U}_b\right)$ and $\mathbf{V} = \text{diag}\left(\mathbf{V}_1, \ldots, \mathbf{V}_b\right)$

- Null-space basis Computation (lines 5, 7): For each block $i = 1, \ldots, b$, we compute $\mathbf{P}^{(i)} = \texttt{null}(\mathbf{\Omega}^{(i)}, r)$ and $\mathbf{Q}^{(i)} = \texttt{null}(\mathbf{\Psi}^{(i)}, r)$. These computations involve matrices of size at most $3^d m \times s$. Using Householder QR [28, Table C.2] for `null`, the cost for one such computation is approximately: $\mathcal{O}\left(3^{3d}m^3\right)$. Since there are b blocks, this step adds:

$$\mathcal{O}\left(b \times 3^{3d}m^3\right) \times T_{\text{flop}},$$

  where T_{flop} is the cost of one floating-point arithmetic operation.

- Column Basis Extraction (Lines 6, 8): Computing $\mathbf{U}_i = \texttt{col}(\mathbf{Y}_i\mathbf{P}^{(i)}, k)$ and $\mathbf{V}_i = \texttt{col}(\mathbf{Z}_i\mathbf{Q}^{(i)}, k)$ involves matrix multiplications of dimensions $3^d m \times r$ with $r \times k$ for each block. The cost of these multiplications across all blocks is:

$$\mathcal{O}\left(b \times [3^d m r^2]\right) \times T_{\text{flop}}.$$

- Reconstruction: Once the basis matrices $\mathbf{U}$ and $\mathbf{V}$ are computed, determining the matrix $\mathbf{D}$ for the full matrix reconstruction as in (6) involves additional matrix-vector products. The cost of this step is exactly:

$$(3^d m + kb) \times T_{\text{mult}}.$$

Combining the contributions from all steps and using that $N = mb$, we express this as:

$$2N(3^d m + r) \times T_{\text{rand}} + \mathcal{O}\left(N \times 3^{3d}m^2\right) \times T_{\text{flop}} + (3^{(d+1)}m + 2r + kN/m) \times T_{\text{mult}}.$$

4 Tagging in Uniform BLR Matrix Compression

We now describe a new black-box randomized method to compress strongly admissible uniform BLR matrices which we call tagging, the main contribution of our manuscript. As in Section 3, we begin with an illustrative example to introduce the concept before generalizing to tagging for d-dimensional problem geometries and summarizing its asymptotic complexity.

4.1 Tagging for strongly-admissible uniform BLR matrices

Let $\mathbf{A}$ be an $N \times N$ strongly admissible uniform BLR matrix as in Figure 1, tessellated into b blocks of size $m \times m$ with uniform block-rank k, allowing for a small amount of oversampling given by p, and let $r = k + p$. Our aim once again is to construct random test matrices $\mathbf{\Omega}, \mathbf{\Psi} \in \mathbb{R}^{N\times s}$ with $s = \mathcal{O}(k)$ so that the sketches $\mathbf{Y} = \mathbf{A\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$ taken a priori can be used to compress the admissible blocks in every block-row and block-column.

To illustrate, suppose as in Section 3.1 that we want to compute the basis matrix $\mathbf{U}_3 \in \mathbb{R}^{m\times k}$ from Figure 2. We first introduce the 8×4 *tagging matrix*

$$\mathbf{T} = \begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,3} & t_{1,4} \\ t_{2,1} & t_{2,2} & t_{2,3} & t_{2,4} \\ \vdots & \vdots & \vdots & \vdots \\ t_{8,1} & t_{8,2} & t_{8,3} & t_{8,4} \end{bmatrix}, \tag{11}$$

where the entries $t_{i,j}$ will be made explicit in Section 5; for now, we treat them as i.i.d. standard Gaussian entries. We note that the number of rows of $\mathbf{T}$ equals the number of tessellated matrix blocks b. The number of columns of $\mathbf{T}$ is one more than the maximal number of neighbors within a given block-row or block-column, which is 4 for the strongly admissible 1-D matrix in Figure 1. In general, for a d-dimensional problem geometry, the number of columns of $\mathbf{T}$ is $3^d + 1$.

We next define the *extended random test matrix* $\mathbf{\Omega} \in \mathbb{R}^{N\times 4r}$ in terms of random test matrices $\mathbf{\Omega}_j \in \mathbb{R}^{N\times r}$ for $j = 1, \ldots, 4$, given by

$$\underset{N\times 4r}{\mathbf{\Omega}} := \begin{bmatrix} \mathbf{\Omega}_1 & \mathbf{\Omega}_2 & \mathbf{\Omega}_3 & \mathbf{\Omega}_4 \end{bmatrix} = \begin{bmatrix} t_{1,1}\mathbf{G}_1 & t_{1,2}\mathbf{G}_1 & t_{1,3}\mathbf{G}_1 & t_{1,4}\mathbf{G}_1 \\ t_{2,1}\mathbf{G}_2 & t_{2,2}\mathbf{G}_2 & t_{2,3}\mathbf{G}_2 & t_{2,4}\mathbf{G}_2 \\ t_{3,1}\mathbf{G}_3 & t_{3,2}\mathbf{G}_3 & t_{3,3}\mathbf{G}_3 & t_{3,4}\mathbf{G}_3 \\ t_{4,1}\mathbf{G}_4 & t_{4,2}\mathbf{G}_4 & t_{4,3}\mathbf{G}_4 & t_{4,4}\mathbf{G}_4 \\ \vdots & \vdots & \vdots & \vdots \\ t_{8,1}\mathbf{G}_8 & t_{8,2}\mathbf{G}_8 & t_{8,3}\mathbf{G}_8 & t_{8,4}\mathbf{G}_8 \end{bmatrix}, \tag{12}$$

where each $\mathbf{G}_i \in \mathbb{R}^{m\times r}$, $i = 1, \ldots, 8$, is a Gaussian random matrix, weighted by entry $t_{i,j}$ of the tagging matrix $\mathbf{T}$ to form $\mathbf{\Omega}_j$, $j = 1, \ldots, 4$. Note that we again assume $N = bm$ for notational convenience. We form the sketch matrix $\mathbf{Y} = \mathbf{A\Omega} \in \mathbb{R}^{N\times 4r}$, partitioned into $N \times r$ block-columns commensurately with (12) so that

$$\underset{N\times 4r}{\mathbf{Y}} = \begin{bmatrix} \mathbf{Y}_1 & \mathbf{Y}_2 & \mathbf{Y}_3 & \mathbf{Y}_4 \end{bmatrix} = \begin{bmatrix} \mathbf{A\Omega}_1 & \mathbf{A\Omega}_2 & \mathbf{A\Omega}_3 & \mathbf{A\Omega}_4 \end{bmatrix} = \mathbf{A\Omega}.$$

To compute $\mathbf{U}_3 \in \mathbb{R}^{m\times k}$ as in Figure 2, we exclude contributions from inadmissible blocks in the third block-row by computing a (nonzero) vector $\mathbf{z}^{(3)} = \begin{bmatrix} z_1^{(3)} & z_2^{(3)} & z_3^{(3)} & z_4^{(3)} \end{bmatrix}^*$ so that

$$\mathbf{z}^{(3)} = \mathtt{null}\left(\mathbf{T}^{(3)}\right) := \mathtt{null}\left(\begin{bmatrix} t_{2,1} & t_{2,2} & t_{2,3} & t_{2,4} \\ t_{3,1} & t_{3,2} & t_{3,3} & t_{3,4} \\ t_{4,1} & t_{4,2} & t_{4,3} & t_{4,4} \end{bmatrix}\right). \tag{13}$$

Note that this submatrix $\mathbf{T}^{(3)}$ of $\mathbf{T}$ comprises the rows that correspond to the neighbor list $\mathcal{N}_3 = [2:4]$ of block 3. Now consider the weighted sum

$$z_1^{(3)}\mathbf{\Omega}_1 + z_2^{(3)}\mathbf{\Omega}_2 + z_3^{(3)}\mathbf{\Omega}_3 + z_4^{(3)}\mathbf{\Omega}_4 = \begin{bmatrix} (z_1^{(3)}t_{1,1} + z_2^{(3)}t_{1,2} + z_3^{(3)}t_{1,3} + z_4^{(3)}t_{1,4})\mathbf{G}_1 \\ (z_1^{(3)}t_{2,1} + z_2^{(3)}t_{2,2} + z_3^{(3)}t_{2,3} + z_4^{(3)}t_{2,4})\mathbf{G}_2 \\ (z_1^{(3)}t_{3,1} + z_2^{(3)}t_{3,2} + z_3^{(3)}t_{3,3} + z_4^{(3)}t_{3,4})\mathbf{G}_3 \\ (z_1^{(3)}t_{4,1} + z_2^{(3)}t_{4,2} + z_3^{(3)}t_{4,3} + z_4^{(3)}t_{4,4})\mathbf{G}_4 \\ (z_1^{(3)}t_{5,1} + z_2^{(3)}t_{5,2} + z_3^{(3)}t_{5,3} + z_4^{(3)}t_{5,4})\mathbf{G}_5 \\ \vdots \\ (z_1^{(3)}t_{8,1} + z_2^{(3)}t_{8,2} + z_3^{(3)}t_{8,3} + z_4^{(3)}t_{8,4})\mathbf{G}_8 \end{bmatrix}.$$

By construction, it simplifies to

$$z_1^{(3)}\mathbf{\Omega}_1 + z_2^{(3)}\mathbf{\Omega}_2 + z_3^{(3)}\mathbf{\Omega}_3 + z_4^{(3)}\mathbf{\Omega}_4 = \begin{bmatrix} (z_1^{(3)}t_{1,1} + z_2^{(3)}t_{1,2} + z_3^{(3)}t_{1,3} + z_4^{(3)}t_{1,4})\mathbf{G}_1 \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ (z_1^{(3)}t_{5,1} + z_2^{(3)}t_{5,2} + z_3^{(3)}t_{5,3} + z_4^{(3)}t_{5,4})\mathbf{G}_5 \\ \vdots \\ (z_1^{(3)}t_{8,1} + z_2^{(3)}t_{8,2} + z_3^{(3)}t_{8,3} + z_4^{(3)}t_{8,4})\mathbf{G}_8 \end{bmatrix}, \tag{14}$$

so that the rows of $\mathbf{A}(z_1^{(3)}\mathbf{\Omega}_1 + z_2^{(3)}\mathbf{\Omega}_2 + z_3^{(3)}\mathbf{\Omega}_3 + z_4^{(3)}\mathbf{\Omega}_4) = z_1^{(3)}\mathbf{Y}_1 + z_2^{(3)}\mathbf{Y}_2 + z_3^{(3)}\mathbf{Y}_3 + z_4^{(3)}\mathbf{Y}_4$ corresponding to I_3 will contain the desired sample of the admissible blocks for computing $\mathbf{U}_3$, with the contributions from inadmissible blocks now zeroed out. We can then compute $\mathbf{U}_3 = \texttt{col}\left((z_1^{(3)}\mathbf{Y}_1 + z_2^{(3)}\mathbf{Y}_2 + z_3^{(3)}\mathbf{Y}_3 + z_4^{(3)}\mathbf{Y}_4)(I_3, :), k\right)$ as in the randomized rangefinding procedure.

In general, the tagging method computes basis matrices $\mathbf{U}$ and $\mathbf{V}$ according to Algorithm 2, which we quickly summarize. We begin by drawing the entries of the tagging matrix $\mathbf{T} \in \mathbb{R}^{b\times(3^d+1)}$ e.g. from a standard Gaussian distribution (line 1-2). We next form the random sketches

$$\underset{N\times(3^d+1)r}{\mathbf{Y}} = \begin{bmatrix}\mathbf{Y}_1 & \mathbf{Y}_2 & \dots & \mathbf{Y}_{3^d+1}\end{bmatrix} = \begin{bmatrix}\mathbf{A}\mathbf{\Omega}_1 & \mathbf{A}\mathbf{\Omega}_2 & \dots & \mathbf{A}\mathbf{\Omega}_{3^d+1}\end{bmatrix} = \mathbf{A}\mathbf{\Omega}, \tag{15}$$

$$\underset{N\times(3^d+1)r}{\mathbf{Z}} = \begin{bmatrix}\mathbf{Z}_1 & \mathbf{Z}_2 & \dots & \mathbf{Z}_{3^d+1}\end{bmatrix} = \begin{bmatrix}\mathbf{A}^*\mathbf{\Psi}_1 & \mathbf{A}^*\mathbf{\Psi}_2 & \dots & \mathbf{A}^*\mathbf{\Psi}_{3^d+1}\end{bmatrix} = \mathbf{A}^*\mathbf{\Psi}. \tag{16}$$

where each $\mathbf{\Omega}_j$, $j = 1, \dots, 3^d + 1$, comprises b block-rows of independent Gaussian matrices $\mathbf{G}_i \in \mathbb{R}^{m\times r}$, $i = 1, \dots, b$, weighted by tagging entry $t_{i,j}$ as in (12), and similarly for each $\mathbf{\Psi}_j$ with $\mathbf{H}_i$ (lines 3-8). To compute the basis matrix $\mathbf{U}_i$ for any block i, we focus on the submatrix $\mathbf{T}^{(i)}$ of $\mathbf{T}$ comprising rows indexed by $\mathcal{N}_i$, the list of neighbors of block i so that $|\mathcal{N}_i| \le 3^d$. We then compute an orthonormal tagging vector $\mathbf{z}^{(i)} = \begin{bmatrix} z_1^{(i)} z_2^{(i)} \dots z_{3^d+1}^{(i)} \end{bmatrix}^*$ in the nontrivial null space of $\mathbf{T}^{(i)}$ (line 10). Finally, we compute $\mathbf{U}_i, \mathbf{V}_i \in \mathbb{R}^{m\times k}$ (lines 11-12) whose columns are orthonormal basis vectors approximating the column spaces of $\mathbf{A}(I_i, :)$ and $\mathbf{A}^*(I_i, :)$, excluding contributions from the inadmissible blocks in block-row i.

Algorithm 2 Tagging for Basis Construction

Require: Fast matrix-vector multiplication with uniform BLR $\mathbf{A} \in \mathbb{C}^{N\times N}$ and $\mathbf{A}^* \in \mathbb{C}^{N\times N}$, $b\times b$ flat matrix tessellations, maximum block-size m, d-dimensional geometry

Ensure: $\mathbf{U}, \mathbf{V} \in \mathbb{C}^{N\times bk}$ in uniform BLR representation of $\mathbf{A}$ as in (6)

1: Set $r = k + p$ and $\mathbf{\Omega}, \mathbf{\Psi} = [\,]$.
2: Form tagging matrix $\mathbf{T} \in \mathbb{R}^{b\times(3^d+1)}$ ▷ *e.g. Gaussian* $\mathbf{T}$*; see Section 5*
3: **for** blocks $i = 1, \dots, b$ **do**
4: Draw independent Gaussian matrices $\mathbf{G}_i, \mathbf{H}_i \in \mathbb{R}^{m\times r}$
5: Update $\mathbf{\Omega} = \begin{bmatrix}\mathbf{\Omega} & ; & \begin{bmatrix}t_{i,1}\mathbf{G}_i & \dots & t_{i,3^d+1}\mathbf{G}_i\end{bmatrix}\end{bmatrix}$ ▷ *Append* i^{th} *row to* $\mathbf{\Omega}$
6: Update $\mathbf{\Psi} = \begin{bmatrix}\mathbf{\Psi} & ; & \begin{bmatrix}t_{i,1}\mathbf{H}_i & \dots & t_{i,3^d+1}\mathbf{H}_i\end{bmatrix}\end{bmatrix}$ ▷ *Append* i^{th} *row to* $\mathbf{\Psi}$
7: **end for**
8: Sketch $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$
9: **for** blocks $i = 1, \dots, b$ **do**
10: Compute $\mathbf{z}^{(i)} = \texttt{null}\left(\mathbf{T}^{(i)}\right)$
11: Compute $\mathbf{U}_i = \texttt{col}\left(\sum_{j=1}^{3^d+1} z_j^{(i)}\mathbf{Y}_j^{(i)},\ k\right)$
12: Compute $\mathbf{V}_i = \texttt{col}\left(\sum_{j=1}^{3^d+1} z_j^{(i)}\mathbf{Z}_j^{(i)},\ k\right)$
13: **end for**
14: $\mathbf{U} = \text{diag}\,(\mathbf{U}_1, \dots, \mathbf{U}_b)$ and $\mathbf{V} = \text{diag}\,(\mathbf{V}_1, \dots, \mathbf{V}_b)$

4.2 Asymptotic Complexity of Tagging

We derive the asymptotic complexity of Algorithm 2 in terms of the problem size N, block size m, block-rank k, oversampling parameter p, and problem geometry dimension d. Let $r = k + p$ and assume $N = mb$, where b is the number of matrix blocks in each block-row or block-column. The complexity of the algorithm is broken into the following components:

- Formation of the Tagging Matrix (Lines 3-4): The tagging matrix $\mathbf{T} \in \mathbb{R}^{b\times(3^d+1)}$ requires $(3^d+1)N/m$ elements to be sampled from a Gaussian distribution. The total time is:

$$(3^d+1)\frac{N}{m} \times T_{\text{rand}}.$$

- Formation of Random Test Matrices $\mathbf{\Omega}$ and $\mathbf{\Psi}$ (Lines 5–8): Random test matrices $\mathbf{\Omega}, \mathbf{\Psi} \in \mathbb{R}^{N\times(3^d+1)r}$ involves sampling $\mathbf{G}_i, \mathbf{H}_i$ for each subblock and forming the test matrices $\mathbf{\Omega}, \mathbf{\Psi}$ by scaling each block by the relevant tag. Sampling contributes $2Nr \cdot T_{\text{rand}}$, as each Gaussian matrix $\mathbf{G}_i$ and $\mathbf{H}_i$ $(i = 1, \ldots, b)$ is reused. Forming $\mathbf{\Omega}, \mathbf{\Psi}$ requires $2(3^d+1)Nr$ floating-point operations, contributing $2(3^d+1)Nr \cdot T_{\text{flop}}$. The total complexity for $\mathbf{\Omega}$ and $\mathbf{\Psi}$ is:

$$2Nr \times T_{\text{rand}} + \mathcal{O}(3^d Nr) \times T_{\text{flop}}.$$

- Matrix-Vector Products for Sampling (Line 9): Computing the sample matrices $\mathbf{Y} = \mathbf{A\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\mathbf{\Psi}$ involves $2(3^d+1)r$ matrix-vector products, with complexity:

$$2(3^d+1)r \times T_{\text{mult}}.$$

- Null Space Computation (Lines 11–12): For each block, the null space computation involves matrices of size at most $3^d \times (3^d+1)$. Using Householder reflections, the computational cost is:

$$\mathcal{O}\left(3^{3d}b\right) \cdot T_{\text{flop}}.$$

- Basis Matrix Construction (Lines 13–15): Forming the basis matrices $\mathbf{U}$ and $\mathbf{V}$ involves multipling the relevant subblocks by the null space vector, then computing an orthogonal basis. This involves $(3^d+1)mr$ floating-point operations per block to scale submatrices of $\mathbf{Y}$ and $\mathbf{Z}$ and $\mathcal{O}(rmk + k^3)$ floating-point operations per block for $\mathbf{U}_i$ and $\mathbf{V}_i$. The total complexity is:

$$\mathcal{O}(3^d Nr + Nrk + Nk^3/m) \times T_{\text{flop}}.$$

- Reconstruction: Reconstructing the uniform BLR matrix using $\mathbf{U}$ and $\mathbf{V}$ requires additional matrix-vector products and contributes

$$(3^d m + kb) \times T_{\text{mult}}.$$

Combining the above contributions, the overall complexity is:

$$\left((3^d+1)\frac{N}{m} + 2Nr\right) \cdot T_{\text{rand}} + \mathcal{O}\left(3^d Nr + Nrk + \frac{Nk^3}{m} + 3^{3d}\frac{N}{m}\right) \times T_{\text{flop}} + (2 \cdot 3^{d+1}r + 3^d m + kb) \times T_{\text{mult}}.$$

Comparing to block nullification in Section 3.2, far fewer samples are needed to construct the basis matrices $\mathbf{U}$ and $\mathbf{V}$. Block nullification requires $2 \times 3^d m$ samples for basis construction in Section 3.2, whereas tagging only requires $2 \times 3^{d+1}r$ samples. The cost of reconstructing the uniform BLR matrix, however, dominates the asymptotic complexity of T_{mult} for both methods. The key advantage of tagging is the substantially reduced cost of post-processing the test and sketch matrices, with linear post-processing cost, as opposed to block-nullification, which scales linearly in N and quadratically with the block size.

5 Selecting the Tagging Matrix

In its introduction in Section 4.1, we treated the tagging matrix as having standard Gaussian entries. However, standard Gaussian tagging matrix entries do not guarantee standard Gaussian samples of the input matrix. As such, we seek to address the following questions. *Does there exist an optimal tagging matrix that yields Gaussian samples? If so, what is it? If not, how closely can we approximate one?*

To answer these questions, we must first be explicit about what constitutes optimality, so we discuss the criteria in Section 5.1. We then offer a conjecture on the existence of optimal tagging matrices in Section 5.2 which takes an algebraic-geometric perspective on tagging matrix optimality. We finish the section by presenting a highly efficient alternative strategy to determine tagging matrices that perform well empirically despite their sub-optimality.

5.1 On the optimality of tagging matrices: Projected tags and aspect ratios

The main issue that we need to address in tagging matrix selection concerns the *projected tags*, the nonzero non-uniform weights on each Gaussian matrix in (14) from Section 4.1. Recall that for any block $i = 1, \ldots, b$ with neighbor list $\mathcal{N}_i$, we compute $\mathbf{z}^{(i)} = \begin{bmatrix} z_1^{(i)} & z_2^{(i)} & \ldots & z_\ell^{(i)} \end{bmatrix} = \texttt{null}(\mathbf{T}^{(i)})$, letting $\ell = 3^d + 1$, for submatrix $\mathbf{T}^{(i)} = \mathbf{T}(\mathcal{N}_i, :)$. Then consider the linear combination

$$z_1^{(i)}\mathbf{\Omega}_1 + z_2^{(i)}\mathbf{\Omega}_2 + \ldots + z_\ell^{(i)}\mathbf{\Omega}_\ell = \begin{bmatrix} (t_{1,1}z_1^{(i)} + t_{1,2}z_2^{(i)} + \ldots t_{1,\ell}z_\ell^{(i)})\mathbf{G}_1 \\ (t_{2,1}z_1^{(i)} + t_{2,2}z_2^{(i)} + \ldots t_{2,\ell}z_\ell^{(i)})\mathbf{G}_2 \\ \vdots \\ (t_{b,1}z_1^{(i)} + t_{b,2}z_2^{(i)} + \ldots t_{b,\ell}z_\ell^{(i)})\mathbf{G}_b \end{bmatrix}. \tag{17}$$

Note that the coefficients of $\mathbf{G}_j$ (i.e. projected tags) are 0 by construction for any $j \in \mathcal{N}_i$. However, the nonzero projected tags, which correspond to the far-field $\mathcal{F}_i = [b] \setminus \mathcal{N}_i$ of block i, non-uniformly weight each Gaussian matrix, resulting in non-uniformly weighted randomized samples of blocks within the same block-row or block-column of $\mathbf{A}$. Each of the projected tags should ideally be equal in magnitude.

To this end, we define the *aspect ratio* $\rho^{(i)}$ for block-row or block-column $i = 1, \ldots, b$ as the largest-magnitude to the smallest-magnitude nonzero projected tag:

$$\rho^{(i)} = \frac{\max\limits_{j \in \mathcal{F}_i} |t_{j,1}z_1^{(i)} + t_{j,2}z_2^{(i)} + \ldots + t_{j,\ell}z_\ell^{(i)}|}{\min\limits_{j \in \mathcal{F}_i} |t_{j,1}z_1^{(i)} + t_{j,2}z_2^{(i)} + \ldots + t_{j,\ell}z_\ell^{(i)}|} \geq 1. \tag{18}$$

Our goal is then to determine an optimal tagging matrix $\mathbf{T}$ for which the projected tags minimize $\rho^{(i)}$ for each $i = 1, \ldots, b$.

In the following subsections, we examine the tagging matrix optimality problem through two different lenses. The first relies on ideas from algebraic geometry to determine an optimal tagging matrix. The second offers a computational short-cut via null space vectors that minimize aspect ratios through a fast numerical optimization scheme.

5.2 On the existence of optimal tagging matrices: Plücker coordinates

To express our conjecture on optimal tagging matrices, we draw a connection between tagging matrices and projective varieties in algebraic geometry through *Plücker coordinates*. We return to our familiar example of a uniform BLR matrix from Figure 1 for an intuitive introduction to the Plücker embedding that gives rise to Plücker coordinates. We then hypothesize that optimal tagging matrices may be found through a hybrid numeric-symbolic approach based on Plücker coordinates, which is currently out of computational reach.

5.2.1 An illustrative example

Let $\mathbf{A} \in \mathbb{R}^{N \times N}$ be the uniform BLR matrix from Figure 1. Note that for blocks $i = 2, \ldots, 7$, we can apply Cramer's Rule to determine $\mathbf{z}^{(i)} = [z_1^{(i)}, z_2^{(i)}, z_3^{(i)}, z_4^{(i)}]^*$ from $\mathbf{T}^{(i)}$, e.g. the i^{th} coordinate $z_i^{(3)}$ of $\mathbf{z}^{(3)}$ can be

computed as the determinant of $\mathbf{T}^{(3)}$ without the i^{th} column:

$$\begin{aligned} z_1^{(3)} &= \det\left(\begin{bmatrix} t_{2,2} & t_{2,3} & t_{2,4} \\ t_{3,2} & t_{3,3} & t_{3,4} \\ t_{4,2} & t_{4,3} & t_{4,4} \end{bmatrix}\right), \qquad z_2^{(3)} = \det\left(\begin{bmatrix} t_{2,1} & t_{2,3} & t_{2,4} \\ t_{3,1} & t_{3,3} & t_{3,4} \\ t_{4,1} & t_{4,3} & t_{4,4} \end{bmatrix}\right), \\ z_3^{(3)} &= \det\left(\begin{bmatrix} t_{2,1} & t_{2,2} & t_{2,4} \\ t_{3,1} & t_{3,2} & t_{3,4} \\ t_{4,1} & t_{4,2} & t_{4,4} \end{bmatrix}\right), \qquad z_4^{(3)} = \det\left(\begin{bmatrix} t_{2,1} & t_{2,2} & t_{2,3} \\ t_{3,1} & t_{3,2} & t_{3,3} \\ t_{4,1} & t_{4,2} & t_{4,3} \end{bmatrix}\right). \end{aligned} \tag{19}$$

Then notice that the vector of projected tags $\mathbf{T}\mathbf{z}^{(i)}$ contains all 4×4 subdeterminants of $\mathbf{T}$ that can be formed from the three rows of $\mathbf{T}^{(i)}$ plus one remaining row of $\mathbf{T}$, e.g. for $\mathbf{z}^{(3)}$ using (19),

$$\mathbf{T}\mathbf{z}^{(3)} = \begin{bmatrix} z_1^{(3)} t_{1,1} + z_2^{(3)} t_{1,2} + z_3^{(3)} t_{1,3} + z_4^{(3)} t_{1,4} \\ \vdots \\ z_1^{(3)} t_{8,1} + z_2^{(3)} t_{8,2} + z_3^{(3)} t_{8,3} + z_4^{(3)} t_{8,4} \end{bmatrix} = \begin{bmatrix} \det\left(\begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,3} & t_{1,4} \\ t_{2,1} & t_{2,2} & t_{2,3} & t_{2,4} \\ t_{3,1} & t_{3,2} & t_{3,3} & t_{3,4} \\ t_{4,1} & t_{4,2} & t_{4,3} & t_{4,4} \end{bmatrix}\right) \\ \vdots \\ \det\left(\begin{bmatrix} t_{2,1} & t_{2,2} & t_{2,3} & t_{2,4} \\ t_{3,1} & t_{3,2} & t_{3,3} & t_{3,4} \\ t_{4,1} & t_{4,2} & t_{4,3} & t_{4,4} \\ t_{8,1} & t_{8,2} & t_{8,3} & t_{8,4} \end{bmatrix}\right) \end{bmatrix}. \tag{20}$$

When $\mathbf{T}$ has full rank, the 4×4 determinants in each coordinate of the projected tags $\mathbf{T}\mathbf{z}^{(i)}$ for $i = 2, \ldots, 7$ form a subset of the *Plücker relations*[2], the set of all possible 4×4 determinants from the the rows of $\mathbf{T}$. Now, let $\mathcal{L}$ be a 4-dimensional subspace of $\mathbb{R}^8$ with $\mathrm{Col}(\mathbf{T}) = \mathcal{L}$. We define the *Plücker embedding* as the map from $\mathcal{L}$ to the point in real projective space whose coordinates are all 4×4 determinants of $\mathbf{T} \in \mathbb{R}^{8\times 4}$. In algebraic-geometric terms, the Plücker embedding maps the *Grassmannian manifold* $\mathrm{Gr}(4, 8)$ comprising all 4-dimensional subspaces of $\mathbb{R}^8$, to *Plücker coordinates* in $\mathbb{P}^{\binom{8}{4}-1}$, as stated formally below:

Definition 5.1 *The **Plücker embedding** is the map* $\mathrm{Gr}(k, n) \to \mathbb{P}^{\binom{n}{k}-1}$ *that identifies* $\mathcal{V} \in \mathrm{Gr}(k, n)$ *with a unique point in real projective space, whose coordinates are given by all* $k \times k$ *determinants of a matrix* $\mathbf{B} \in \mathbb{R}^{n\times k}$ *satisfying* $\mathcal{V} = \mathrm{Col}(\mathbf{B})$*, called **Plücker coordinates**.*

Definition 5.2 *Let* $\mathcal{V} \in \mathrm{Gr}(k, n)$ *and suppose* $\mathbf{B} \in \mathbb{R}^{n\times k}$ *satisfies* $\mathcal{V} = \mathrm{Col}(\mathbf{B})$*. For any ordered sequence of* k *row indices* $1 \le i_1 < \ldots < i_k \le n$ *of* $\mathbf{B}$*, let* $B_{i_1,\ldots,i_k}$ *denote the determinant of the* $k \times k$ *submatrix* $\mathbf{B}([i_1, \ldots, i_k], :)$*, so that set of all Plücker coordinates may be denoted* $\{B_{i_1,\ldots,i_k}\}$*. Then for any two ordered sequences of row indices*

$$1 \le i_1 < i_2 < \ldots < i_{k-1} \le n, \qquad 1 \le j_1 < j_2 < \ldots < j_{k+1} \le n,$$

*the **Plücker relations** are the following homogeneous quadratic equations that must hold for all Plücker coordinates* $\{B_{i_1,\ldots,i_k}\}$*:*

$$\sum_{\ell=1}^{k+1} (-1)^\ell B_{i_1,\ldots,i_{k-1},j_\ell} B_{j_1,\ldots,\widehat{j_\ell},\ldots,j_{k+1}} = 0, \tag{21}$$

where $j_1, \ldots, \widehat{j_\ell}, \ldots, j_{k+1}$ *is the sequence* $j_1, \ldots, j_{k+1}$ *with the term* j_ℓ *omitted.*

We can now pose the optimality of tagging matrices in algebraic-geometric terms.

[2]We treat the "extremal" blocks $i = 1$ and $i = 8$ (in general, blocks with fewer than the maximal number of neighbors 3^d) at the end of the section.

5.2.2 On the existence of optimal tagging matrices

Let $\mathbf{T} \in \mathbb{R}^{b\times 3^d+1}$ be a matrix of indeterminates $t_{i,j}$ for $1 \le i \le b$ and $1 \le j \le 3^d+1$. For any ordered sequence of 3^d+1 row indices $1 \le i_1 < \ldots < i_{3^d+1} \le b$ of $\mathbf{T}$, let $T_{i_1,\ldots,i_{3^d+1}}$ denote the determinant of the $(3^d+1)\times(3^d+1)$ submatrix $\mathbf{T}([i_1,\ldots,i_{3^d+1}],:)$. For example, we would let $T_{2,3,4,8}$ denote the final 4×4 determinant in (20) of the submatrix comprising $\mathbf{T}^{(3)}$ and the last row of $\mathbf{T}$. Note that each determinant $T_{i_1,\ldots,i_{3^d+1}}$ is a degree-(3^d+1) polynomial in the indeterminates $t_{i,j}$. From the previous section, each of these determinants is a Plücker coordinate that must satisfy the Plücker relations, which are quadratic polynomials in the indeterminates $T_{i_1,\ldots,i_{3^d+1}}$ for all possible ordered sequences $1 \le i_1 < \ldots < i_{3^d+1} \le b$.

To determine tagging matrix entries $t_{i,j}$ that minimize $\rho^{(i)}$ for each block i, we propose the following approach. It is well-known that the set of Plücker relations is not algebraically independent, cf. [46, Chapter 14.2] and [27, Appendix C.7]. Thus, the first step is to determine an algebraically independent generating set of Plücker relations for $\mathbf{T}$. One method is the computation of a Gröbner basis using Buchberger's algorithm[3] for the ideal of the polynomial ring $\mathbb{C}[\{t_{i,j}\}_{i\in[b],j\in[3^d+1]}]$ generated by all Plücker relations, cf. [47].

Remark 5.1 *Another avenue of investigation that bears future consideration involves the so-called "clusters" formed by independent Plücker coordinates [49]. One such cluster is comprised of rectangular Plücker coordinates [34], which correspond to the rectangular partitions of a $k\times(n-k)$ unit rectangle and form a generating set for the coordinate ring. Rectangular Plücker coordinates relate to quantum Schubert calculus on the flag variety [7, 15, 19] and have an associated Laurent polynomial with certain properties [39] that may offer another path to an optimal tagging matrix.*

Now, let $\mathcal{R}$ denote a set of algebraically independent Plücker relations that generate the projective variety defined by all Plücker relations. Recall that for the purposes of tagging, we are only interested in the nonzero projected tags in (17), which correspond to the far-field indices of each block. Let n be the total number of nonzero projected tags, with n_i nonzero projected tags for block i, so that $\sum_{i=1}^{b} n_i = n$. Denote by $\{T_1,\ldots,T_n\}$ these n nonzero projected tags, or n nonzero determinants of $(3^d+1)\times(3^d+1)$ submatrices of $\mathbf{T}$,

$$\left\{ T_{i_1^{(1)},\ldots,i_{3^d+1}^{(1)}},\ T_{i_1^{(2)},\ldots,i_{3^d+1}^{(2)}},\ \ldots,\ T_{i_1^{(n)},\ldots,i_{3^d+1}^{(n)}} \right\} \tag{22}$$

for distinct ordered sequences $1 \le i_1^{(\ell)} < \ldots < i_{3^d+1}^{(\ell)} \le b$ for $\ell = 1,\ldots,n$.

Ideally, for each block i, every nonzero projected tag, or nonzero coordinate of $\mathbf{T}\mathbf{z}^{(i)}$, should be equal (cf. (18)), which we can enforce numerically via

$$\begin{aligned} &\min_{\mathbf{T}\in\mathbb{R}^{b\times 3^d+1}} \sum_{i=1}^{b}\sum_{\ell_i=1}^{n_i} (T_{\ell_i}-\overline{T})^2 \\ &\text{subject to } \ \mathcal{R} \end{aligned} \tag{23}$$

where $\overline{T}$ denotes the mean of the n_i nonzero projected tags for block i. We say an arg min $\mathbf{T}^*$ of (23) is numerically optimal if $\max_i(\rho^{(i)}) > 1$.

We now note that the outlined approach only holds for blocks that have a neighbor list of maximal size 3^d. One workaround is to reassign the necessary number of admissible blocks to be inadmissible in the block-rows or block-columns of $\mathbf{A}$ that have fewer than 3^d inadmissible blocks, though this increases the overall cost of reconstruction. More detrimental, though, is the cost of the symbolic computation required. While the optimization problem in (23) is straightforward, the computations for $\mathcal{R}$ are highly nontrivial even for very small values of b, constraints which must hold if the arg min $\mathbf{T}^*$ of (23) satisfies $\mathrm{Col}(\mathbf{T}^*) = \mathcal{V}$ for some (3^d+1)-dimensional subspace $\mathcal{V}$ of $\mathbb{R}^b$. Moreover, if the problem size or underlying geometry were to change, these computations would need to be done anew for different values of b or d. In the next section, we describe a numerical method of minimizing the aspect ratios that performs highly efficiently in practice without sacrificing accuracy.

[3]The computation of a Gröbner basis for the quadratic polynomials under consideration is highly nontrivial for problems of this size due to, e.g., intermediate swell in Buchberger's algorithm; see [14].

5.3 A practical method for numerical optimization

Because of the difficulties in practice of using the algebraic-geometric approach of Section 5.2, we now present an alternative way to minimize the aspect ratios numerically for each block $i = 1, \ldots, b$. This practical approach requires higher-dimensional null spaces of tagging submatrices; as such, we will now consider tagging matrices $\mathbf{T} \in \mathbb{R}^{b \times \ell}$ where $\ell > 3^d + 1$ so that every null space has dimension strictly greater than 1.

To minimize the aspect ratios of projected tags efficiently, we consider the following optimization problem over the $(\ell - 3^d)$-dimensional null space of $\mathbf{T}^{(i)} \in \mathbb{R}^{3^d \times \ell}$, rather than over all possible matrix representations $\mathbf{T}$ of a Grassmannian subspace $\mathcal{V}$ such that $\mathcal{V} = \mathrm{Col}(\mathbf{T})$. We now seek a unit vector $\mathbf{z}^{(i)}$ in the null space of $\mathbf{T}^{(i)}$ for each block $i = 1, \ldots, 8$, which minimizes the ratio of projected tags:

$$\mathbf{z}^{(i)} = \arg \min_{\mathbf{x} \in \mathrm{Null}(\mathbf{T}^{(i)})} \frac{\max\limits_{j \in \mathcal{F}_i} |t_{j,1} x_1^{(i)} + t_{j,2} x_2^{(i)} + \ldots + t_{j,\ell} x_\ell^{(i)}|}{\min\limits_{j \in \mathcal{F}_i} |t_{j,1} x_1^{(i)} + t_{j,2} x_2^{(i)} + \ldots + t_{j,\ell} x_\ell^{(i)}|}. \tag{24}$$

We illustrate the procedure for block-row $i = 3$ as before, adding one extra column to $\mathbf{T}$. Then the tagging submatrix

$$\mathbf{T}^{(3)} = \begin{bmatrix} t_{2,1} & t_{2,2} & t_{2,3} & t_{2,4} & t_{2,5} \\ t_{3,1} & t_{3,2} & t_{3,3} & t_{3,4} & t_{3,5} \\ t_{4,1} & t_{4,2} & t_{4,3} & t_{4,4} & t_{4,5} \end{bmatrix}$$

has a 2-dimensional null space, so let $\begin{bmatrix} \mathbf{x}_1^{(3)} & \mathbf{x}_2^{(3)} \end{bmatrix} = \mathtt{null}(\mathbf{T}^{(3)})$ be an orthonormal basis. Any unit vector $\mathbf{x}$ in the null space may be expressed as

$$\mathbf{x} = \cos(\theta)\mathbf{x}_1^{(3)} + \sin(\theta)\mathbf{x}_2^{(3)}, \qquad \theta \in [0, 2\pi].$$

Thus, we can efficiently determine an optimal null space vector via

$$\theta^* = \arg \min_{\theta \in [0, 2\pi]} \frac{\max\limits_{j \in \mathcal{F}_i} \left| \left(\cos(\theta)\mathbf{x}_1^{(3)} + \sin(\theta)\mathbf{x}_2^{(3)} \right)^* \mathbf{t}^{(j)} \right|}{\min\limits_{j \in \mathcal{F}_i} \left| \left(\cos(\theta)\mathbf{x}_1^{(3)} + \sin(\theta)\mathbf{x}_2^{(3)} \right)^* \mathbf{t}^{(j)} \right|}, \tag{25}$$

where $\mathbf{t}^{(j)} = \begin{bmatrix} t_{j,1} & t_{j,2} & \ldots & t_{j,\ell} \end{bmatrix}^*$ for a more concise representation of the objective function, so that the optimal null space vector is

$$\mathbf{z}^{(3)} = \cos(\theta^*)\mathbf{x}_1^{(3)} + \sin(\theta^*)\mathbf{x}_2^{(3)}.$$

For arbitrary $i = 1, \ldots, b$ with $\mathbf{T} \in \mathbb{R}^{b \times \ell}$ and $\ell > 3^d + 1$, the null space of submatrix $\mathbf{T}^{(i)}$ has dimension at least $s = \ell - 3^d > 1$. We compute

$$\begin{bmatrix} \mathbf{x}_1^{(i)} & \ldots & \mathbf{x}_s^{(i)} \end{bmatrix} = \mathtt{null}\left(\mathbf{T}^{(i)}, s \right),$$

and write any normalized vector in the null space as

$$\mathbf{x} = \alpha_1(\boldsymbol{\theta})\mathbf{x}_1^{(i)} + \ldots + \alpha_s(\boldsymbol{\theta})\mathbf{x}_s^{(i)}$$

for (spherical) coordinates $(\alpha_1(\boldsymbol{\theta}), \ldots, \alpha_s(\boldsymbol{\theta})) \in \mathbb{R}^s$ parameterizing the unit hypersphere over $\boldsymbol{\theta} \in \mathbb{R}^{s-1}$. We then solve the constrained optimization problem

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^{s-1}} \frac{\max\limits_{j \in \mathcal{F}_i} \left| \left(\alpha_1(\boldsymbol{\theta})\mathbf{x}_1^{(i)} + \ldots + \alpha_s(\boldsymbol{\theta})\mathbf{x}_s^{(i)} \right)^* \mathbf{t}^{(j)} \right|}{\min\limits_{j \in \mathcal{F}_i} \left| \left(\alpha_1(\boldsymbol{\theta})\mathbf{x}_1^{(i)} + \ldots + \alpha_s(\boldsymbol{\theta})\mathbf{x}_s^{(i)} \right)^* \mathbf{t}^{(j)} \right|},$$

to write our desired null space vector as

$$\mathbf{z}^{(i)} = \alpha_1(\boldsymbol{\theta}^*)\mathbf{x}_1^{(i)} + \ldots + \alpha_s(\boldsymbol{\theta}^*)\mathbf{x}_s^{(i)},$$

and compute the basis matrix $\mathbf{U}_i$ as in Section 4.

The overall computational cost of this optimization procedure is negligible when it is integrated into Algorithm 2, since the optimization happens over a convex region with generally no more than a 3-dimensional parameterization in practice. The issue of greater concern is that each additional column in the tagging matrix corresponds to $k+p$ additional matvecs with $\mathbf{A}$ and $\mathbf{A}^*$ for $\boldsymbol{\Omega}$ and $\boldsymbol{\Psi}$, though the total is still far fewer matvecs than in block nullification which we will verify numerically in Section 7. First, for completeness, we outline in the next section our method of reconstructing the full uniform BLR representation of (6).

6 Randomized Compression Algorithms for Uniform BLR Matrices

In the previous sections, we focused on step (I) of randomized compression of uniform BLR matrices—the computation of basis matrices $\mathbf{U}, \mathbf{V}$ from random sketches $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}, \mathbf{Z} = \mathbf{A}^*\boldsymbol{\Psi}$. This was, in large part, due to the similarity of compression algorithms after basis matrices have been computed. More precisely, steps (II) and (III) of uniform BLR compression can be executed in a manner that is oblivious to the particular algorithm used to compute $\mathbf{U}$ and $\mathbf{V}$ in step (I), allowing for direct performance comparisons of Algorithms 1 and 2.

In this section, we briefly describe how steps (I)-(III) of uniform BLR compression are conducted in our experiments. Remark 6.1 also summarizes the compression procedure when matrix entries are readily available. First, though, we present in Algorithm 3 the last basis construction algorithm that we consider as a benchmark for step (I), which is equivalent to a blocked version of the randomized SVD done "naively" with $O(bk)$ structured Gaussian test matrices. We then describe a uniform BLR compression procedure that can be performed with basis matrices obtained from any of Algorithms 1-3; we discuss more involved compression algorithms that reuse the sketches from step (I) for steps (II) and (III) in the Appendix.

Remark 6.1 *When matrix entries are readily available, the task of recovering a uniform BLR representation as in (6) can be done with $k+p$ matvecs of $\mathbf{A}-\mathbf{D}$ and $\mathbf{A}^*-\mathbf{D}^*$, where $\mathbf{D}$ is the near-neighbor matrix given by $\mathbf{D}_{i,j} = \mathbf{A}_{i,j}$, for $i = 1, \ldots, b$ and $j \in \mathcal{N}_i$. The recovery of $\widetilde{\mathbf{A}}$ is then done by accessing $O(bk)$ matrix entries if $\mathbf{U}$ and $\mathbf{V}$ are computed as interpolative bases; see [41, Chapter 18] for more details. Note that Sections 3 and 4 are still applicable for the basis computations of step* (I) *in this instance.*

Algorithm 3 Naive RandSVD for Basis Construction

Require: Fast matrix-vector multiplication with uniform BLR $\mathbf{A} \in \mathbb{C}^{N\times N}$ and $\mathbf{A}^* \in \mathbb{C}^{N\times N}$, $b\times b$ flat matrix tessellation, maximum block-size m, d-dimensional geometry

Ensure: $\mathbf{U}, \mathbf{V} \in \mathbb{C}^{N\times bk}$ in uniform BLR representation of $\mathbf{A}$ as in (6)

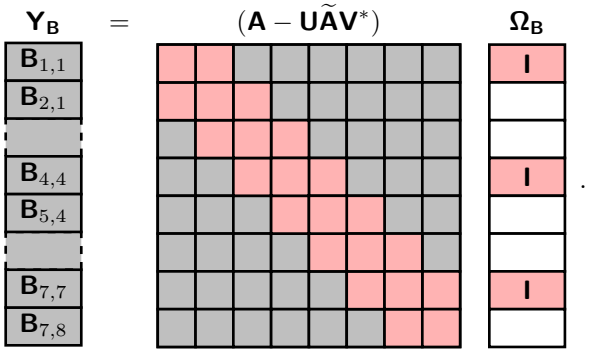
1: Set $r = k + p$
2: **for** blocks $i = 1, \ldots, b$ **do**
3: Draw independent Gaussian test matrices $\boldsymbol{\Omega}, \boldsymbol{\Psi} \in \mathbb{R}^{N\times r}$
4: Set $\boldsymbol{\Omega}(I_j, :) = 0$ and $\boldsymbol{\Psi}(I_j, :) = 0$ for all $j \in \mathcal{N}_i$
5: Form $\mathbf{Y} = \mathbf{A}\boldsymbol{\Omega}$ and $\mathbf{Z} = \mathbf{A}^*\boldsymbol{\Psi}$
6: Compute $\mathbf{U}_i = \texttt{col}\,(\mathbf{Y}(I_i, :),\ k)$
7: Compute $\mathbf{V}_i = \texttt{col}\,(\mathbf{Z}(I_i, :),\ k)$
8: **end for**
9: Set $\mathbf{U} = \text{diag}\,(\mathbf{U}_1, \ldots, \mathbf{U}_b)$ and $\mathbf{V} = \text{diag}\,(\mathbf{V}_1, \ldots, \mathbf{V}_b)$

6.1 Direct evaluation in steps (II) and (III)

We present algorithms to compress uniform BLR matrices that allow for the most direct comparison of our basis construction algorithms. First, we summarize in Algorithm 3 the last basis construction algorithm that

we consider only as a benchmark for step (I), which is equivalent to a blocked version of the randomized SVD done "naively" with $O(bk)$ structured Gaussian test matrices.

The most straightforward way to accomplish steps (II) and (III) for uniform BLR matrices (6) is to evaluate $\widetilde{\mathbf{A}} = \mathbf{U}^*(\mathbf{AV})$ directly, for bk additional matrix-vector multiplications (matvecs) with the basis matrix $\mathbf{V}$ from step (I). Once $\widetilde{\mathbf{A}}$ has been computed, 3^d sparse structured matrices of size $N \times m$, containing the $m \times m$ identity as a submatrix, can be used to extract the nonzero entries of $\mathbf{A} - \mathbf{U}\widetilde{\mathbf{A}}\mathbf{V}^*$ to obtain $\mathbf{B}$, e.g.



Labeled consistently with Algorithms 1-3, these reconstruction algorithms are summarized in Table 1, which we say are of type A; this is in contrast to type B and type C algorithms, which are covered in the appendix. Note that steps 2-3 in Table 1 are identical for all type A algorithms, making them the most ideal for performance comparisons of our basis reconstruction algorithms.

Algorithm A1:	**Algorithm A2:**	**Algorithm A3:**
1. Compute $\mathbf{U}, \mathbf{V}$ with block nullification (Algorithm 1)	1. Compute $\mathbf{U}, \mathbf{V}$ with tagging (Algorithm 2)	1. Compute $\mathbf{U}, \mathbf{V}$ with naive randSVD (Algorithm 3)
2. Form $\widetilde{\mathbf{A}} = \mathbf{U}^*(\mathbf{AV})$	2. Form $\widetilde{\mathbf{A}} = \mathbf{U}^*(\mathbf{AV})$	2. Form $\widetilde{\mathbf{A}} = \mathbf{U}^*(\mathbf{AV})$
3. Form $\mathbf{B} = \mathbf{A} - \mathbf{U}\widetilde{\mathbf{A}}\mathbf{V}^*$ with sparse structured identity matrices	3. Form $\mathbf{B} = \mathbf{A} - \mathbf{U}\widetilde{\mathbf{A}}\mathbf{V}^*$ with sparse structured identity matrices	3. Form $\mathbf{B} = \mathbf{A} - \mathbf{U}\widetilde{\mathbf{A}}\mathbf{V}^*$ with sparse structured identity matrices

Table 1: Randomized compression algorithms of type A for uniform BLR format of (6). Steps 2-3 are identical for each algorithm and contribute an additional bk and $3^d m$ matvecs with $\mathbf{A}$, respectively.

7 Numerical Experiments

In this section, we demonstrate the improved performance of tagging over block nullification in randomized compression of strongly admissible uniform BLR matrices. For several different test problems and problem sizes N, we report the following quantities:

- Accuracy of compressed matrices $\mathbf{A}_{\text{uBLR}}$ of the form (6), using the relative error metric $\frac{\|\mathbf{A}-\mathbf{A}_{\text{uBLR}}\|_2}{\|\mathbf{A}\|_2}$, computed via 20 iterations of the randomized power method [26],
- Total runtime (in seconds) of each compression algorithm, broken down into compression steps (I)-(III), including a separate visualization of the runtime for each of Algorithms 1-3,
- Total number of matvecs with $\mathbf{A}$, $\mathbf{A}^*$ required for compression, broken down into compression steps (I)-(III), including a separate visualization of the matvecs required for Algorithms 1-3.

To investigate the performance of tagging more thoroughly, we also report the aspect ratios incurred when using Gaussian, Haar-distributed, or equidistributed tagging matrices for increasing b; we note that all other experiments were performed with Gaussian tagging matrices. All test problems were implemented in MATLAB 2024a, and all experiments were carried out on a workstation with an Intel(R) Xeon(R) Gold 6254 CPU operating at 3.10GHz with 72 cores and 750 GB of memory.

For each test problem, we use a target block-rank of $k = 30$ with an oversampling parameter of $p = 10$. We also report the number of blocks b and the maximum block size m for each problem size N. To substantiate our choices of b and m, we highlight a key distinction between hierarchical and flat rank-structured matrix formats. Often in hierarchical rank-structured matrix compression, the leaf node size m is chosen such that $m = O(k + p)$ with approximately $2N/m$ total nodes in the index tree; moreover, they can achieve linear complexity by leveraging nested bases, e.g. [35].

By contrast, randomized compression of strongly admissible uniform BLR matrices does not attain linear complexity. The storage requirement in bits of a strongly admissible uniform BLR matrix is

$$M \;\sim\; Nk + b^2k^2 + 3^d\frac{N^2}{b},$$

and the compressed uniform BLR representation $\mathbf{A}_{\text{uBLR}}$ given by (6) can be recovered in no fewer than $N_{\text{matvec}} \sim M/N$ matvecs, since a matrix of size $N \times N_{\text{matvec}}$ holds the *minimum* number of bits needed to store $\mathbf{A}_{\text{uBLR}}$. The dominant storage costs can be attributed to $\widetilde{\mathbf{A}}$ and $\mathbf{B}$, and it is challenging to recover them in an "optimal" number of matvecs; the number of matvecs required for $\widetilde{\mathbf{A}}$ and $\mathbf{B}$ dominates the sample complexity $N_{\text{matvec}} \sim 3^d m + bk$. Thus, we choose b to balance N_{matvec} so that

$$b = \sqrt{\frac{3^d}{k}}\,\sqrt{N} \qquad \Rightarrow \qquad M \sim \sqrt{3^d k}\; N^{3/2}.$$

This choice is reasonable for medium-sized problems, e.g. $20,000 \le N \le 100,000$ in our experiments.

7.1 2D Laplace Kernel

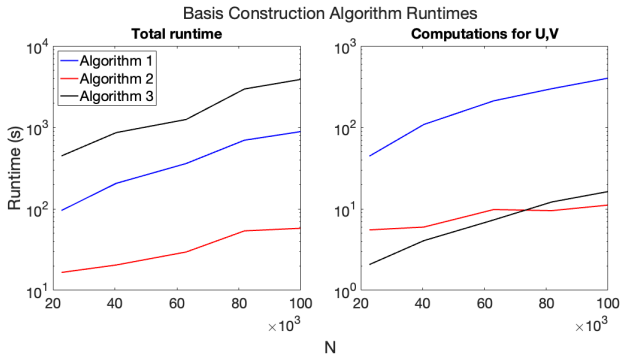
To profile the performance of the method on a benchmark problem, we use the Green's function of the Laplace equation in 2 dimensions for a random distribution of points $\{x_i\}_{i=1}^N$ in the unit square, where

$$\mathbf{A}_{ij} = \log(\|x_i - x_j\|), \text{ for } i \neq j, \tag{26}$$

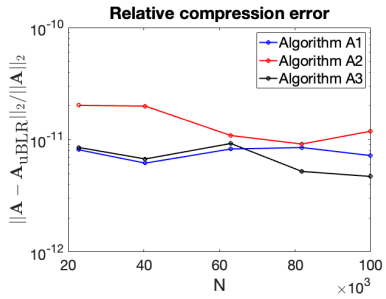
and entries on the diagonal are set to 0. Dense systems of this form commonly arise in the context of integral equations. The matrix entries are straightforward to access and evaluate, and in practice, the method of proxy surfaces is a more fitting approach to approximate basis matrices algebraically [13, 53]. We include the 2D Laplace kernel as a benchmark because the algebraic rank behavior of $\mathbf{A}$ is well-characterized by multipole estimates [22, 23] and exhibits exponential decay.

Figure 3 summarizes the results. Matrix-vector products $\mathbf{x} \to \mathbf{Ax}$ are performed using `FMM2D`, a Fortran implementation of the fast multipole method developed and maintained by the Flatiron Institute. Key observations are as follows:

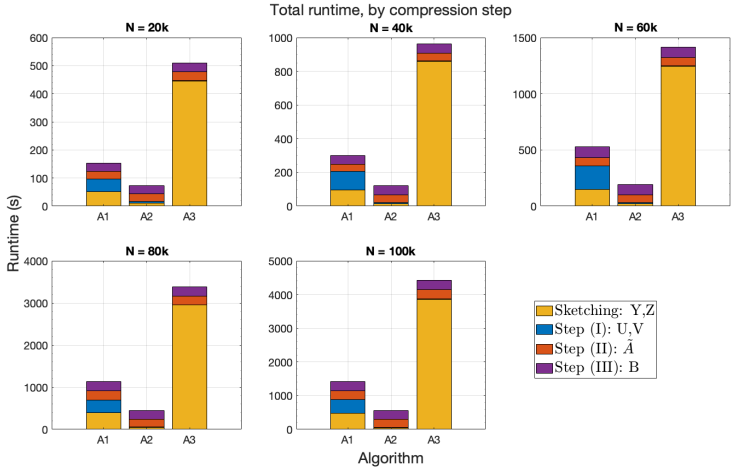
- Reduction in matrix-vector products. Tagging significantly reduces the number of matrix-vector products (N_{matvec}) required for basis construction compared to alternative methods. For Algorithms 1 and 3, N_{matvec} scales with the block size and the number of blocks, respectively—both of which grow with the problem size N in flat formats. In contrast, the number of matvecs for tagging depends on fixed constants, such as the number of neighbors and the rank k.
- Efficiency in basis construction. The reduction in N_{matvec} for basis construction translates into considerable time savings during the sketching of $\mathbf{Y}$ and $\mathbf{Z}$.
- Improved post-processing. Tagging results in substantial post-processing time savings compared to block-nullification.
- Scalability with problem size. Algorithm A2, which uses tagging, achieves an 8.3x reduction in the total number of matvecs for the largest problem size ($N = 100,000$) compared to naive matrix formation.



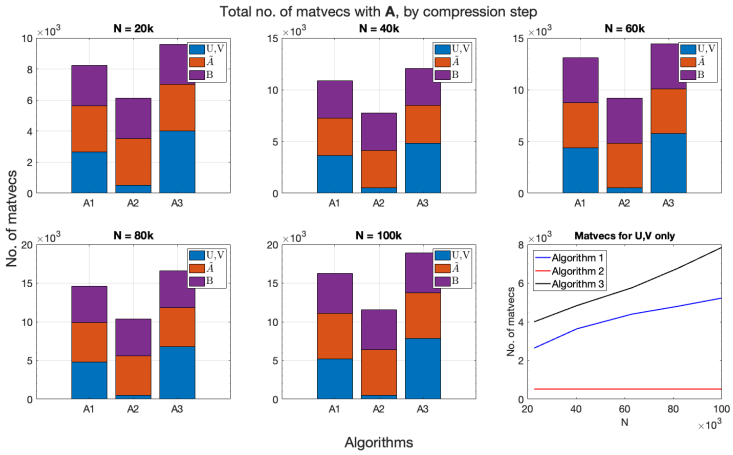
(a) Time for basis construction for algorithmic variants.



(b) Relative error of the approximation.



(c) Total runtime for algorithm variants with a breakdown of algorithm steps.



(d) Total matvecs for algorithm variants with a breakdown of algorithm steps.

Figure 3: The timing and accuracy results for the Laplace 2D FMM example. Figures 3a and 3c report the time for basis reconstruction and total reconstruction, respectively. Figure 3b reports the accuracy of the reconstruction for the algorithmic variants. Figure 3d reports a breakdown of the number of matrix-vector products needed for each stage of the algorithm.

7.2 Sparse Schur Complement for a Thin Slab

In sparse direct solvers for elliptic partial differential equations (PDEs), compressing and factorizing sparse matrices is often necessary. Accessing matrix entries directly is computationally challenging, and randomized sketching techniques are frequently used to accelerate and simplify nested dissection solvers.

Consider solving the constant-coefficient Helmholtz equation with zero body load and prescribed Dirichlet boundary conditions on a domain Ω:

$$\begin{aligned} -\Delta u(x) - \kappa^2 u(x) &= 0, \qquad && x \in \Omega, \\ u(x) &= g(x), && x \in \partial\Omega. \end{aligned} \tag{27}$$

Discretizing with second-order finite differences leads to the linear system $\mathbf{Au} = \mathbf{f}$ to solve. To solve this system efficiently, the domain Ω is partitioned into *thin slabs*, where one dimension is constrained to be electrically small, as demonstrated in [57, 16].

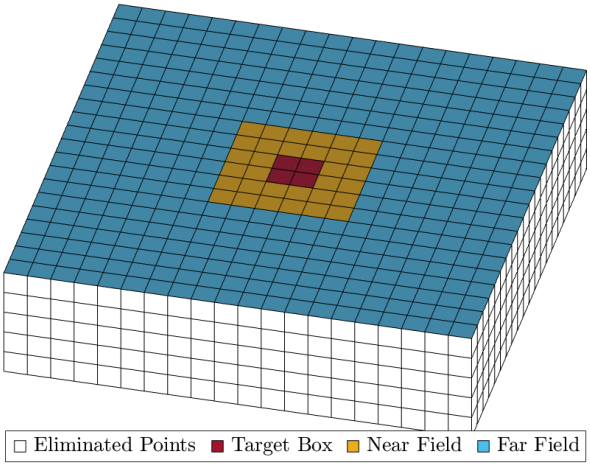


Figure 4: The discretization of a thin slab of size $n \times n \times l$, where $l \ll n$ is fixed to be $l = 10$ in our experiments. The eliminated discretization points are shown in white, and the frontal indices are colored to denote the far field and near field for a target box.

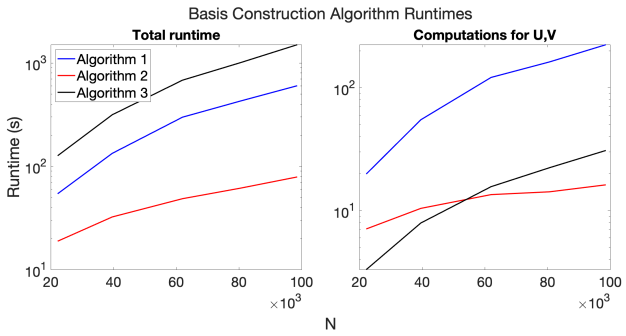
Our experiments explore the use of uBLR matrices for a slab subdomain. For a domain with $n \times n \times l$ discretization points, where l is fixed to be $l = 10$, the front size grows as $N = n^2$. The wavenumber parameter scales with the number of discretization points to maintain 100 points per wavelength. For the largest problem size ($N = 100,000$), the domain measures approximately $3\lambda \times 3\lambda \times 0.1\lambda$, where λ denotes the wavelength.

In the context of domain decomposition, the degrees of freedom are partitioned into frontal nodes and internal nodes, represented by the index vectors $\boldsymbol{J}_f$ and $\boldsymbol{J}_i$, respectively. The Schur complement is the linear algebraic operator that eliminates the internal nodes in a multifrontal solver, resulting in a dense matrix defined on the frontal nodes. Specifically, the Schur complement is given by:

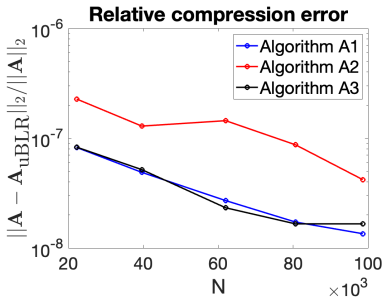
$$\mathbf{T}_{ff} = \mathbf{A}_{ff} - \mathbf{A}_{fi}\mathbf{A}_{ii}^{-1}\mathbf{A}_{if}. \tag{28}$$

While the Schur complement is dense, it can be applied efficiently to vectors by leveraging the sparsity of its components, including the sparse direct solver $\mathbf{A}_{ii}^{-1}$. Since the slab width is fixed, the domain remains pseudo-2D, enabling the efficient factorization of $\mathbf{A}_{ii}$ and fast application of the solver to vectors.

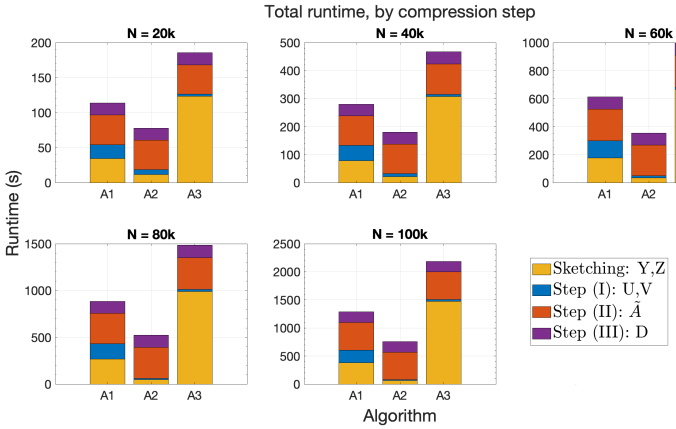
The results, summarized in Figure 5, show that tagging provides excellent scaling in the number of samples needed for basis matrix construction. It is also the most computationally efficient algorithm compared to other variants. Like the experiment of Section 7.1, we have observed that the far-field rank decays exponentially. Since the slab width is fixed, the rank of far-field interactions decays *faster* as the problem size grows, leading to improved approximation accuracy for increasing N and a fixed rank k.



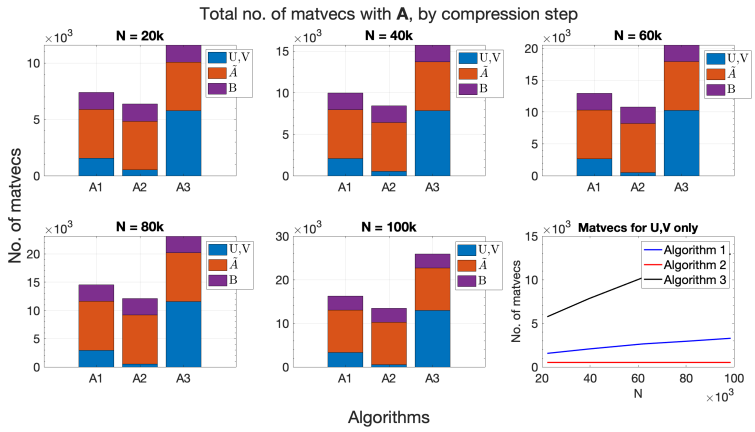
(a) Time for basis construction for algorithmic variants.



(b) Relative error of the approximation.

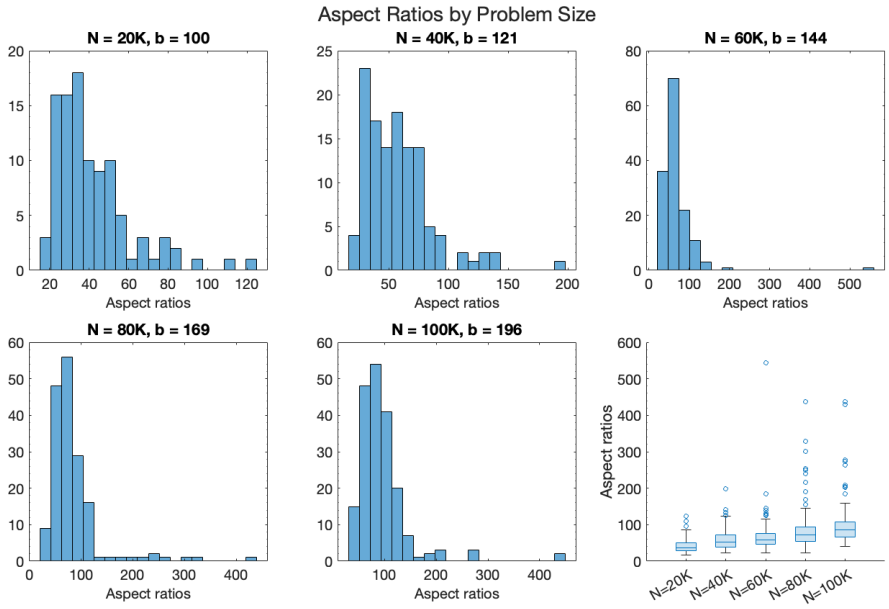


(c) Total runtime for algorithm variants with a breakdown of algorithm steps.

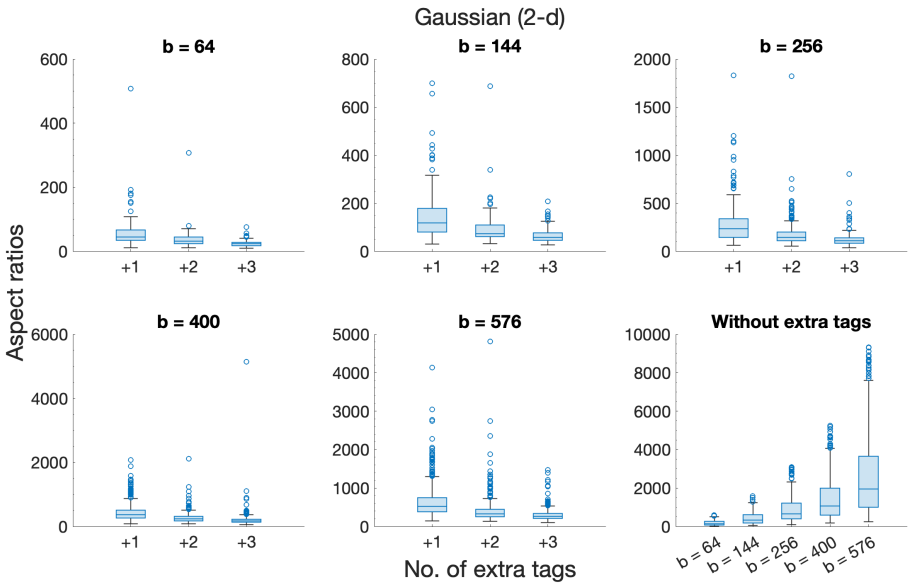


(d) Total matvecs for algorithm variants with a breakdown of algorithm steps.

Figure 5: The timing and accuracy results for the Schur complement of a thin slab, where the Helmholtz equation is discretized to 100 points per wavelength. The slab is a fixed width of 0.1 wavelengths, as the front size increases. Figures 5a and 5c report the time for basis reconstruction and total reconstruction, respectively. Figure 5b reports the accuracy of the reconstruction for the algorithmic variants. Figure 5d reports a breakdown of the number of matrix-vector products needed for each stage of the algorithm.

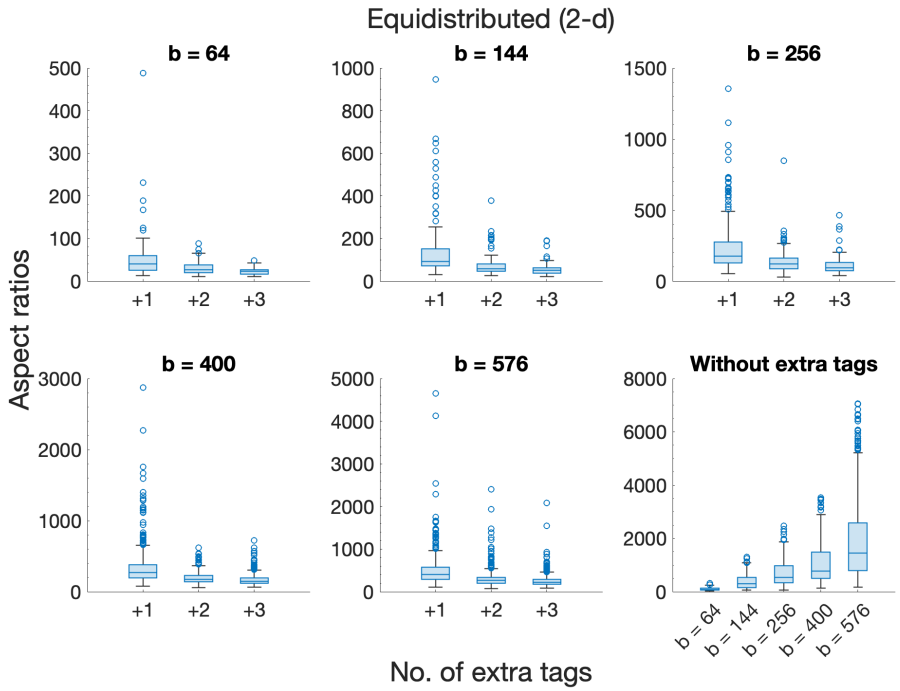


(a) Histogram of the aspect ratios $\rho^{(i)}$ of (18), $i = 1, \ldots, b$ for the problem sizes in the range $20,000 \leq N \leq 100,000$.

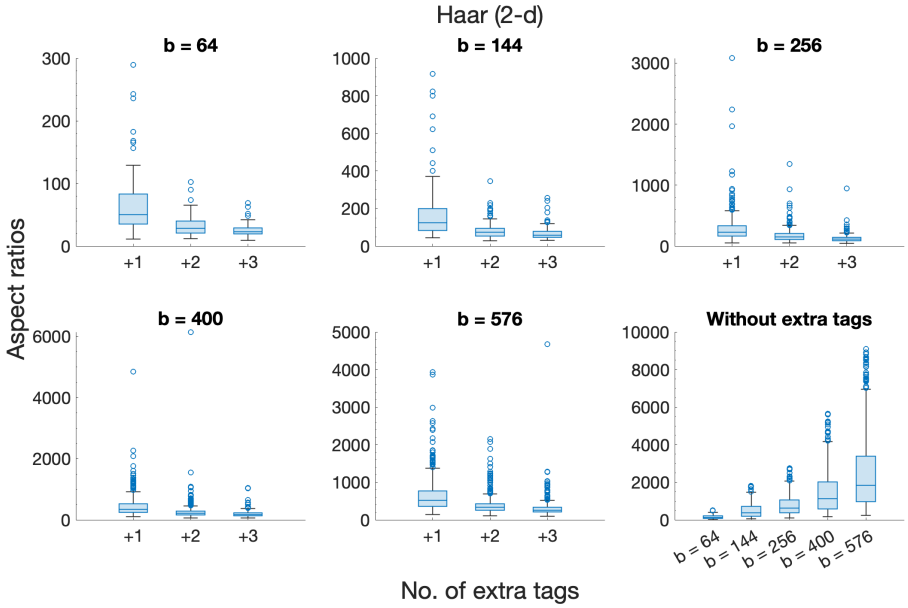


(b) Box plots of the aspect ratios $\rho^{(i)}$ of (18), $i = 1, \ldots, b$, for increasing number of blocks b, using Gaussian tagging matrices with 1, 2, or 3 extra columns. The bottom right figure shows the aspect ratios without any extra tags.

Figure 6: The aspect ratios $\rho^{(i)}$ of (18), $i = 1, \ldots, b$, for Gaussian random tagging matrices. Figure 6a reports the aspect ratios by problem size, for a uniform distribution of points in the 2D square with rank parameter $k = 30$ and oversampling parameter $p = 10$. Figure 6b reports the aspect ratios for an increasing number of blocks and demonstrates an improved performance when using extra tags.



(a) Aspect ratios using equidistributed rows on the unit hypersphere.



(b) Aspect ratios using Haar-distributed tagging matrices.

Figure 7: Box plots of the aspect ratios $\rho^{(i)}$ of (18), $i = 1, \ldots, b$, for increasing number of blocks b, using alternative choices of tagging matrices with 1, 2, or 3 extra columns. Figure 7a uses a random tagging matrices with equidistributed rows on the unit hypersphere, and Figure 7b uses a Haar-distributed tagging matrices. In both subplots, the bottom right figure shows the aspect ratios without any extra tags.

7.3 Performance of Tagging

Tagging significantly reduces the number of matrix-vector products required for basis construction for flat formats. However, unlike the comparison methods in Algorithms 1 and 3, tagging introduces additional sources of error beyond the use of randomized sketching with the rSVD.

Specifically, the process of introducing zeros into the test matrices scales each of the far-field blocks' projected tags, which may vary in magnitude. If the variation in projected tags is too large, some blocks may be scaled disproportionately, potentially affecting the overall accuracy of the computed approximation. To analyze this effect, we conduct detailed experiments on the tagging test matrices and report the aspect ratios $\rho^{(i)}$ from (18), $i = 1, \ldots, b$, for varying numbers of blocks and problem sizes.

The key observation is that the aspect ratios can be effectively controlled by introducing extra tags into the computation. Figure 6 presents a histogram of aspect ratios for increasing problem sizes, corresponding to the experiments shown in Figures 3 and 5. These figures demonstrate minimal loss of approximation accuracy in the overall reconstruction. Additionally, Figure 7 provides box plots of the aspect ratios for projected tags when using alternative tagging matrices, such as a random matrix with equispaced rows on the unit sphere and Haar-distributed matrices.

8 Conclusions and Future Work

In this work, we present a black-box randomized compression algorithm based on our novel method of tagging, which improves on existing randomized compression algorithms for uniform BLR matrices under a strong admissibility condition. To compress an $N \times N$ uniform BLR matrix $\mathbf{A}$, our method only requires $O(k)$ random samples of $\mathbf{A}$ and $\mathbf{A}^*$ for basis computations, versus $O(m + k)$ for block nullification (which increases with N for flat rank-structure formats), where k is the target block-rank and m is the block size. We demonstrate through numerical experiments that compression with tagging achieves comparable accuracy to existing compression algorithms with greatly improved computational efficiency. We also draw a connection between optimality in tagging and Plücker coordinates in algebraic geometry, and we present an alternative numerical method of optimizing tagging matrices that is reliable in practice.

Avenues of future work include the implementation of a hybrid numeric-symbolic computational scheme to generate theoretically optimal tagging matrix entries from their corresponding Plücker relations. Additionally, a high-performance implementation of our randomized compression algorithm for uniform BLR matrices with tagging would be advantageous, given that parallelizing our tagging method can be done straightforwardly. Future work will also investigate an extension of tagging for hierarchical rank-structured formats with shared or nested bases.

References

[1] Kadir Akbudak, Hatem Ltaief, Aleksandr Mikhalev, and David Keyes. Tile Low Rank Cholesky Factorization for Climate/Weather Modeling Applications on Manycore Architectures. In Julian M. Kunkel, Rio Yokota, Pavan Balaji, and David Keyes, editors, *High Performance Computing*, pages 22–40, Cham, 2017. Springer International Publishing.

[2] Noha Al-Harthi, Rabab Alomairy, Kadir Akbudak, Rui Chen, Hatem Ltaief, Hakan Bagci, and David Keyes. Solving Acoustic Boundary Integral Equations Using High Performance Tile Low-Rank LU Factorization. In Ponnuswamy Sadayappan, Bradford L. Chamberlain, Guido Juckeland, and Hatem Ltaief, editors, *High Performance Computing*, pages 209–229, Cham, 2020. Springer International Publishing.

[3] Patrick Amestoy, Cleve Ashcraft, Olivier Boiteau, Alfredo Buttari, Jean-Yves L'Excellent, and Clément Weisbecker. Improving Multifrontal Methods by Means of Block Low-Rank Representations. *SIAM Journal on Scientific Computing*, 37(3):A1451–A1474, 2015.

[4] Patrick Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. On the Complexity of the Block Low-Rank Multifrontal Factorization. *SIAM Journal on Scientific Computing*, 39(4):A1710–A1740, 2017.

[5] Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo A. Mary. Bridging the Gap Between Flat and Hierarchical Low-Rank Matrix Formats: The Multilevel Block Low-Rank Format. *SIAM Journal on Scientific Computing*, 41(3):A1414–A1442, 2019.

[6] Cleve Ashcraft, Alfredo Buttari, and Theo Mary. Block Low-Rank Matrices with Shared Bases: Potential and Limitations of the BLR^2 Format. *SIAM Journal on Matrix Analysis and Applications*, 42(2):990–1010, 2021.

[7] V. V. Batyrev, I. Ciocan-Fontanine, B. Kim, and D. van Straten. Mirror symmetry and toric degenerations of partial flag manifolds. *Acta Mathematica*, 184(1):1–39, 2000.

[8] Mario Bebendorf. *Hierarchical matrices*. Springer, 2008.

[9] Steffen Börm. *Efficient numerical methods for non-local operators*, volume 14 of *EMS Tracts in Mathematics*. European Mathematical Society (EMS), Zürich, 2010. $\mathcal{H}^2$-matrix compression, algorithms and analysis.

[10] Christos Boutsidis and Alex Gittens. Improved matrix algorithms via the subsampled randomized hadamard transform. *SIAM Journal on Matrix Analysis and Applications*, 34(3):1301–1340, 2013.

[11] Qinglei Cao, Yu Pei, Kadir Akbudak, Aleksandr Mikhalev, George Bosilca, Hatem Ltaief, David Keyes, and Jack Dongarra. Extreme-Scale Task-Based Cholesky Factorization Toward Climate and Weather Prediction Applications. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '20, New York, NY, USA, 2020. Association for Computing Machinery.

[12] Shiv Chandrasekaran, Ming Gu, and Timothy Pals. A fast ulv decomposition solver for hierarchically semiseparable representations. *SIAM Journal on Matrix Analysis and Applications*, 28(3):603–622, 2006.

[13] Hongwei Cheng, Zydrunas Gimbutas, Per-Gunnar Martinsson, and Vladimir Rokhlin. On the compression of low rank matrices. *SIAM Journal on Scientific Computing*, 26(4):1389–1404, 2005.

[14] Alexander Demin and Shashi Gowda. Groebner.jl: A package for Gröbner bases computations in Julia. *arXiv preprint*, abs/2304.06935, 2023.

[15] Tohru Eguchi, Kentaro Hori, and Chuan sheng Xiong. Gravitational quantum cohomology. *Int. J. Mod. Phys.*, A12:1743–1782, 1997.

[16] Björn Engquist and Lexing Ying. Sweeping preconditioner for the helmholtz equation: hierarchical matrix representation. *Communications on pure and applied mathematics*, 64(5):697–735, 2011.

[17] Klaus Giebermann. Multilevel approximation of boundary integral operators. *Computing*, 67(3):183–207, 2001.

[18] Adrianna Gillman, Patrick M Young, and Per-Gunnar Martinsson. A direct solver with o (n) complexity for integral equations on one-dimensional domains. *Frontiers of Mathematics in China*, 7(2):217–247, 2012.

[19] A. B. Givental. Stationary phase integrals, quantum toda lattices, flag manifolds and the mirror conjecture. *Topics in singularity theory, American Mathematical Society Translations Ser 2.*, 180, 1997.

[20] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[21] Christopher Gorman, Gustavo Chávez, Pieter Ghysels, Théo Mary, Francois-Henry Rouet, and Xiaoye Sherry Li. Robust and accurate stopping criteria for adaptive randomized sampling in matrix-free hierarchically semiseparable construction. *SIAM Journal on Scientific Computing*, 41(5):S61–S85, 2019.

[22] L Greengard and V Rokhlin. A fast algorithm for particle simulations. *J. Comp. Phys*, 73:325–348, 1987.

[23] Leslie Greengard and Vladimir Rokhlin. A new version of the fast multipole method for the laplace equation in three dimensions. *Acta numerica*, 6:229–269, 1997.

[24] W. Hackbusch, B. Khoromskij, and S. A. Sauter. On H^2-Matrices. In Hans-Joachim Bungartz, Ronald H. W. Hoppe, and Christoph Zenger, editors, *Lectures on Applied Mathematics*, pages 9–29, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[25] Wolfgang Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

[26] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[27] John Harnad and Ferenc Balogh. *Tau Functions and their Applications*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, 2021.

[28] Nicholas J. Higham. *Functions of Matrices: Theory and Computation*. SIAM, 2008.

[29] Nicholas J Higham and Theo Mary. Solving block low-rank linear systems by LU factorization is numerically stable. *IMA Journal of Numerical Analysis*, 42(2):951–980, 04 2021.

[30] Akihiro Ida, Hiroshi Nakashima, and Masatoshi Kawai. Parallel hierarchical matrices with block low-rank representation on distributed memory computer systems. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPCAsia '18, page 232–240, New York, NY, USA, 2018. Association for Computing Machinery.

[31] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 604–613, New York, NY, USA, 1998. Association for Computing Machinery.

[32] Claude-Pierre Jeannerod, Théo Mary, Clément Pernet, and Daniel S. Roche. Improving the Complexity of Block Low-Rank Factorizations with Fast Matrix Arithmetic. *SIAM Journal on Matrix Analysis and Applications*, 40(4):1478–1496, 2019.

[33] William B. Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into hilbert space. *Contemporary mathematics*, 26:189–206, 1984.

[34] Steven N. Karp. Moment curves and cyclic symmetry for positive grassmannians. *Bulletin of the London Mathematical Society*, 51(5):900–916, 2019.

[35] James Levitt and Per-Gunnar Martinsson. Linear-complexity black-box randomized compression of hierarchically block separable matrices. *arXiv preprint arXiv:2205.02990*, 2022.

[36] James Levitt and Per-Gunnar Martinsson. Randomized compression of rank-structured matrices accelerated with graph coloring. *arXiv preprint arXiv:2205.03406*, 2022.

[37] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167–20172, 2007.

[38] Lin Lin, Jianfeng Lu, and Lexing Ying. Fast construction of hierarchical matrix representation from matrix–vector multiplication. *Journal of Computational Physics*, 230(10):4071–4087, 2011.

[39] B.R. Marsh and K. Rietsch. The B-model connection and mirror symmetry for Grassmannians. *Advances in Mathematics*, 366:107027, 2020.

[40] Per-Gunnar Martinsson. Compressing rank-structured matrices via randomized sampling. *SIAM Journal on Scientific Computing*, 38(4):A1959–A1986, 2016.

[41] Per-Gunnar Martinsson. *Fast direct solvers for elliptic PDEs*. SIAM, 2019.

[42] Per-Gunnar Martinsson and Joel A Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 29:403–572, 2020.

[43] P.G. Martinsson. Rapid factorization of structured matrices via randomized sampling, 2008. arXiv:0806.2339.

[44] P.G. Martinsson and V. Rokhlin. A fast direct solver for boundary integral equations in two dimensions. *J. Comp. Phys.*, 205(1):1–23, 2005.

[45] Theo Mary. *Block Low-Rank Multifrontal Solvers: Complexity, Performance, and Scalability.* PhD thesis, Université de Toulouse, Toulouse, France, Nov. 2017.

[46] Ezra Miller and Bernd Sturmfels. Combinatorial commutative algebra. In *Graduate Texts in Mathematics*. Springer New York, 2004.

[47] Chenqi Mou, Qiuye Song, Yutong Zhou, Alicia Dickenstein, Bettina Eick, Kevin Buzzard, Anton Leykin, and Yue Ren. DetGB: A Software Package for Computing Gröbner Bases of Determinantal Ideals. In *Mathematical Software – ICMS 2024*, Lecture Notes in Computer Science, pages 354–364. Springer Nature Switzerland, Cham, 2024.

[48] Grégoire Pichon, Eric Darve, Mathieu Faverge, Pierre Ramet, and Jean Roman. Sparse supernodal solver using block low-rank compression: Design, performance and analysis. *Journal of Computational Science*, 27:255–270, 2018.

[49] Joshua S. Scott. Grassmannians and cluster algebras. *Proceedings of the London Mathematical Society*, 92(2):345–380, 2006.

[50] Marc Sergent, David Goudin, Samuel Thibault, and Olivier Aumage. Controlling the memory subscription of distributed applications with a task-based runtime system. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 318–327, 2016.

[51] Joel A Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. Practical sketching algorithms for low-rank matrix approximation. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1454–1485, 2017.

[52] Jianlin Xia, Shivkumar Chandrasekaran, Ming Gu, and Xiaoye S Li. Superfast multifrontal method for large structured linear systems of equations. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1382–1411, 2010.

[53] Xin Xing and Edmond Chow. Interpolative decomposition via proxy points for kernel matrices. *SIAM Journal on Matrix Analysis and Applications*, 41(1):221–243, 2020.

[54] Anna Yesypenko. *Randomized algorithms for the efficient solution of elliptic PDEs on modern architectures.* PhD thesis, 2023.

[55] Anna Yesypenko, Chao Chen, and Per-Gunnar Martinsson. A simplified fast multipole method based on strong recursive skeletonization. *Journal of Computational Physics*, page 113707, 2024.

[56] Anna Yesypenko and Per-Gunnar Martinsson. Randomized Strong Recursive Skeletonization: Simultaneous compression and factorization of $\mathcal{H}$-matrices in the Black-Box Setting. *arXiv:2311.01451 [math.NA]*, 2023.

[57] Anna Yesypenko and Per-Gunnar Martinsson. SlabLU: a two-level sparse direct solver for elliptic PDEs. *Advances in Computational Mathematics*, 50(4):90, 2024.