Factor Fitting, Rank Allocation, and Partitioning in Multilevel Low Rank Matrices

Tetiana Parshakova Trevor Hastie Eric Darve Stephen Boyd

October 27, 2023

Abstract

We consider multilevel low rank (MLR) matrices, defined as a row and column permutation of a sum of matrices, each one a block diagonal refinement of the previous one, with all blocks low rank given in factored form. MLR matrices extend low rank matrices but share many of their properties, such as the total storage required and complexity of matrix-vector multiplication. We address three problems that arise in fitting a given matrix by an MLR matrix in the Frobenius norm. The first problem is factor fitting, where we adjust the factors of the MLR matrix. The second is rank allocation, where we choose the ranks of the blocks in each level, subject to the total rank having a given value, which preserves the total storage needed for the MLR matrix. The final problem is to choose the hierarchical partition of rows and columns, along with the ranks and factors. This paper is accompanied by an open source package that implements the proposed methods.

Contents

1 Introduction **3**
- 1.1 Prior work . . . 3
- 1.2 Our contributions . . . 6
- 1.3 Outline . . . 7

2 Multilevel low rank matrices **7**
- 2.1 Contiguous multilevel low rank matrices . . . 7
- 2.2 Two-matrix forms . . . 9
- 2.3 Multilevel low rank matrices . . . 10
- 2.4 Variations . . . 10
- 2.5 Properties . . . 11

3 MLR fitting problem **12**
- 3.1 General MLR fitting . . . 12
- 3.2 Factor fitting and rank allocation . . . 13
- 3.3 Special case: Low rank matrix . . . 13

4 Factor fitting methods **15**
- 4.1 Alternating least squares . . . 15
- 4.2 Block coordinate descent . . . 16
- 4.3 Comparison . . . 17

5 Rank allocation **19**
- 5.1 Incrementing and decrementing level rank . . . 19
- 5.2 Rank exchange algorithm . . . 19

6 A method for general MLR fitting **21**
- 6.1 Symmetric row and column dissection . . . 21
- 6.2 Non-symmetric row and column dissection . . . 23

7 Extensions and variations **24**
- 7.1 Rank allocation . . . 24
- 7.2 Sparsity refinement . . . 24
- 7.3 Generalized multilevel low rank matrix . . . 24

8 Numerical examples **26**
- 8.1 Asset covariance matrix . . . 26
- 8.2 Fiedler matrix . . . 30
- 8.3 Discrete Gauss transform matrix . . . 32
- 8.4 Distance matrix . . . 34
- 8.5 Multiscale inverse polynomial kernel matrix . . . 36

1 Introduction

Multilevel low rank (MLR) matrices are an extension of low rank matrices and factor models of covariance matrices. They generalize low rank matrices in the sense of giving a substantial reduction in storage and speed up in computing matrix-vector products, compared to a full dense matrix. MLR matrices are closely related to hierarchical matrices, which have been studied since at least the late 1980s. There are different definitions of hierarchical matrices, and our MLR notion is close to, but not the same as, others that have been proposed. Our focus is on three problems associated with fitting a given full matrix with an MLR matrix with respect to the Frobenius norm.

The first problem we address is factor fitting, which involves finding the factors of the MLR matrix to best fit a given matrix in Frobenius norm. We present two complementary block coordinate descent algorithms for this fitting problem. One alternates between updating the left and the right factors, each of these done by solving a structured least squares problem. The other method cycles over optimizing the matrices at each level of the hierarchy, which can be done via the singular value decomposition (SVD).

The second problem is rank allocation, where we divide up a fixed total rank across the levels of the hierarchy, which preserves the storage required. We propose a heuristic based on the SVD of the matrices at each level of the hierarchy, which estimates the change in fitting objective when the rank allocated to each level is either incremented or decremented, and then chooses a candidate exchange of rank across the levels.

The third problem is general MLR fitting, where only the matrix to be approximated and a total rank are given. This problem involves finding the hierarchical partitioning of rows and columns, as well as rank allocation and fitting the factors. We propose a simple nested dissection of the rows and columns using spectral bi-clustering.

Except for very simple cases (such as a low rank matrix) the methods we propose are heuristic, and can and do converge to non-global local optima. The methods we propose are related to prior work (as described below in more detail), but we believe that our methods for factor fitting and rank allocation are novel.

1.1 Prior work

Low rank approximation. The problem of finding a low rank approximation to a given matrix is readily solved via the singular value decomposition (SVD) [Dat10, DW21]. The solution was originally proposed in 1907 [Sch07] in the context of integral operators and rediscovered for matrices in 1936 [EY36]. Since then the problem has been extensively studied [NS17]. Numerous algorithms have been devised to solve the problem approximately with a reduced computational cost, including QR decomposition with pivoting [Ste98], rank revealing QR factorization [Cha87], randomized algorithms [Ach03, DV06] and cross/skeleton decomposition [Beb11]. By storing the left and right factors, low rank matrices require far less storage than a dense matrix with the same dimensions. Various operations can be carried out more efficiently, *e.g.*, matrix-vector multiply.

Hierarchical matrices. Hierarchical matrices ($\mathcal{H}$-matrices) have been around since at least the late 1980s [GR87, Tyr96, Hac99], with multiple books dedicated to them [Beb08, Hac15]. These matrices arise naturally from numerical solutions to integral equations [GKK85, Rok85, GR87], and discretization of kernel functions and finite element matrices [Hac15, §10, 11].

Various specific forms for hierarchical matrices have been proposed previously, among which the $\mathcal{H}$-matrix is the most general one [Hac99, BGH03, GH03, Beb08, Hac15]. A special case of $\mathcal{H}$-matrix is the hierarchically off-diagonal low-rank (HODLR) matrix [AAD16], where the off-diagonal blocks on multiple levels are low rank. $\mathcal{H}^2$-matrices [HB02, BGH03] are a subcategory of $\mathcal{H}$-matrices with nested bases, and hierarchical semiseparable (HSS) matrices [CGP06, XCGL10a, AD13] are a special case of HODLR matrices with nested bases. The fast multipole method (FMM) [GR87, Dar00, Nis02, YBZ04], which is considered one of the top 10 algorithms of 20th century [Cip00], can be interpreted as relying on $\mathcal{H}^2$-matrices.

All of these types of hierarchical matrices provide storage reduction and enable fast matrix-vector multiplication, just like low rank matrices. Approximate LU factorization, and solving linear equations, can be carried out much faster than for a dense matrix of the same dimensions [Hac15, §7]. This is especially useful since many matrices that arise in scientific and engineering contexts are, or can be approximated by, hierarchical matrices [Kal63, Bon99, Buh03, Hsi06].

Direct solvers [ADLK01, HRR02, CDHR08] become computationally impractical for large-scale dense linear systems. Hence, iterative methods such as conjugate gradient [HS+52], MINRES [PS75], or GMRES [SS86] should be considered. In practice, however, iterative methods need to be accompanied by preconditioners [Wat15]. Efficient preconditioners can be developed using fast approximate hierarchical matrix factorization. These hierarchical preconditioners include $\mathcal{H}$-matrices [Hac99, GH03, SY14, GHK08], HODLR [KBR11, AAD16, PCD17], HSS [CGP06, CDG+07, Mar09, XCGL10b, GYM12, Xia13] and others [CPD17, KD20, KCH+22].

While we also consider hierarchical diagonal block structure as in HODLR, our MLR matrices are a bit different from HODLR. We decompose our matrix into a sum of block diagonal matrices for each level of the hierarchy with each block having a low rank. When representing MLR using HODLR format, an extra logarithmic factor (in the size of the matrix) is introduced in the storage. Similarly, an extra logarithmic factor appears when representing HODLR using MLR format. We present a comparison between MLR and HODLR in §8.

Block low rank matrices. Block low rank (BLR) matrix format [AAB+15] is a flat non-hierarchical block matrix structure. Specifically, the matrix is partitioned into blocks of the same size with dense diagonal blocks and low rank off-diagonal blocks. BLR matrix format has been successfully employed to accelerate sparse direct methods [PDF+18, ABLM19], multifrontal method [AAB+15, Mar17], Cholesky factorization [ALMK17, CPA+20], solving boundary integral equations [AHAA+20] and many more [Wei13, SBA17]. Although LU factorization using BLR matrices has shown improvements, it is still less efficient in terms of both memory and operations compared to the ones based on $\mathcal{H}$-matrices [AAB+15].

Our MLR matrices have similarities with BLR matrices, but they are not the same. Specifically, blocks in an MLR matrix have contributions from all levels in the hierarchy, and therefore need not be low rank.

Butterfly matrices. Butterfly matrices [Par95] are a structured set of matrices that encode the recursive divide and conquer fast Fourier transform algorithm [CT65]. Butterfly matrix parameterization [DGE+19] is motivated by the fact that matrices with fast matrix-vector multiplication can be characterized as being factorizable into products of sparse matrices [DSCP+18]. This parameterization captures many structured linear transformations with nearly optimal space and time complexity [DSG+20]. It has been successfully employed for sparse neural network training [DCL+21], specifically using Monarch matrices [DCS+22], which generalize butterfly matrices while being hardware efficient.

MLR and Monarch matrices share certain similarities, yet they are distinct. Monarch matrices are parameterized as products of two block diagonal matrices up to permutation. MLR, on the other hand, is a row and column permutation of a sum of products of two block diagonal matrices. The permutation in Monarch matrices is fixed, turning the first block diagonal factor into a block matrix with diagonal blocks, while the permutation in MLR is general and is applied to factors in a different order. We compare MLR and Monarch matrices empirically in §8.

Hierarchical principal component analysis. Principal component analysis (PCA) is a technique that reduces the dimensionality of a set of observations while preserving as much variance as possible. PCA was presented back in 1901 [Pea01] and since then there have been many developments [JC16], *e.g.*, generalized low rank models [UHZ+16] which impose additional structure on the factors. Another development is so-called multi-block component models, which are used when the data matrix is collected in blocks [SWDJ03].

Hierarchical PCA (HPCA) is a type of multiblock PCA method [WKT96], which is used to increase the interpretability of PC scores and loadings when the number of features is large. HPCA is easier to interpret because the model coefficients are divided into multiple levels, modeling the relationship between and within the blocks. To the best of our knowledge, HPCA is related to MLR matrices only by the fact that both models are hierarchical, *i.e.*, they use multi-level hierarchical partitioning, and, as with all the matrices discussed above, involve low rank approximation of blocks.

Factor models. Factor analysis is a method for modeling observed features in terms of a smaller number of unobserved or latent factors. It was pioneered in psychometrics in 1904 [Spe04]. Factor models represent a covariance matrix as a sum of a low rank matrix (related to the factors), plus a diagonal matrix (sometimes called the idiosyncratic variance), and are a special case of our definition of MLR matrix. There are many methods for fitting factor (*i.e.*, low rank plus diagonal) matrices; see, *e.g.*, [Har67, Cou13, §2, App. A]. Related concepts also include higher-order and hierarchical factor models [YTM99, MNP13] as well as multilevel factor models [GB14, CKKK18]. With these structures, we can characterize variations both between and within blocks, which increases the interpretability of factors.

Spectral clustering. Our method for finding a suitable hierarchical partition of the rows and columns relies on the well-known idea of spectral clustering. There are various clustering techniques including connectivity-based, centroid-based, distribution-based, data stream-based and spectral clustering [XT15]. Spectral methods [VM03], in particular, have been very popular because of their simple and efficient implementation that relies on linear algebra packages [vL07]. Spectral graph partitioning was first introduced in the 1970s [Hal70, DH73, Fie73] and gained popularity in the 1990s [PSL90].

Bi-clustering (or co-clustering) is a clustering technique for simultaneous clustering the rows and columns of a matrix. It was introduced in 1972 [Har72] and later generalized in 2000 [CC00] for simultaneous clustering of genes and conditions. Spectral methods have proved effective in bipartite graph partitioning [Dhi01, ZHD$^+$01] and have been successfully applied to hierarchical matrices as well [Beb08, §1.4.1].

Rank allocation. There are methods for fitting all of the matrix forms described above, including choosing the rank of the submatrices. Previous studies have explored an adaptive rank determination algorithm [Hac15, §6.6] which sets the rank of each block based on a prescribed upper bound on the truncation error. Similarly, in [MRK22] the blocks are recursively partitioned until a small truncation error is achieved given a maximum rank per block. We are not aware of prior work that uses a fixed total rank across the levels, and then adjusts the allocation of rank. To the best of our knowledge our rank allocation algorithm is a novel approach.

1.2 Our contributions

The main contributions of this paper are the following:

1. We introduce a novel definition of multilevel low rank matrix that, while closely related, differs from the traditional definitions of hierarchical matrix.

2. We present two complementary block coordinate descent algorithms for factor fitting.

3. We present an algorithm for rank allocation that is able to re-allocate the rank assigned to each level in the hierarchy, to improve the fitting.

We provide an open-source package that implements these methods, available at

https://github.com/cvxgrp/mlr_fitting.

We also provide a number of examples that illustrate our methods.

1.3 Outline

In §2 we describe our definition of MLR matrices, which is very close to, but not the same as, traditional hierarchical matrices. We describe the MLR fitting problem in §3. In §4 we give two different block coordinate descent method methods for factor fitting, where we choose the coefficients in the MLR matrix. In §5 we describe our rank allocation algorithm, where a given total rank is to be allocated across the different levels of the MLR matrix. We describe a method for the full MLR fitting problem in §6, where only the total rank budget and the matrix to be approximated are given. We present variations and extensions to MLR in §7. Finally we illustrate our methods with numerical examples in §8.

2 Multilevel low rank matrices

In this section we define multilevel low rank (MLR) matrices and set our notation. We also give a few basic properties of MLR matrices, even though our focus is on the problem of fitting or approximating a matrix with an MLR one.

2.1 Contiguous multilevel low rank matrices

We first describe a special case of MLR matrices, one where the index partitions are contiguous, which simplifies the notation. An $m \times n$ contiguous MLR matrix A with L levels has the form

$$A = A^1 + \cdots + A^L, \tag{1}$$

where

$$A^l = \mathbf{blkdiag}(A_{l,1}, \ldots, A_{l,p_l}), \quad l = 1, \ldots, L,$$

where **blkdiag** (or direct sum) concatenates its not necessarily square matrix arguments in both rows and columns. Here p_l is the size of the partition at level l, with $p_1 = 1$. We refer to $A_{l,k}$ as the kth block on level l. The dimensions of the block $A_{l,k}$ are $m_{l,k} \times n_{l,k}$, for $l = 1, \ldots, L$, $k = 1, \ldots, p_l$. For the sum above to make sense, we must have

$$\sum_{k=1}^{p_l} m_{l,k} = m, \quad \sum_{k=1}^{p_l} n_{l,k} = n, \quad l = 1, \ldots, L.$$

The block dimensions on level l partition the row and column indices into p_l groups, which are contiguous. Since $p_1 = 1$, the level 1 row and column partitions each consist of one set,

$$I = \{1, \ldots, m\}, \qquad J = \{1, \ldots, n\}.$$

Less trivially, the level 2 partition of the row indices is the set of p_2 index sets

$$\{1, \ldots, m_{2,1}\},\ \{m_{2,1}+1, \ldots, m_{2,1}+m_{2,2}\},\ \ldots,\ \{m - m_{2,p_2}+1, \ldots, m\}.$$

We require that these partitions be hierarchical, which means that the row or column partition at level l is a refinement of the row or column partition at level $l-1$, for $l = 2, \ldots, L$.

We require that blocks on level l have rank not exceeding r_l, for $l = 1, \dots, L$. We write them in factored form as

$$A_{l,k} = B_{l,k} C_{l,k}^T, \quad B_{l,k} \in \mathbf{R}^{m_{l,k} \times r_l}, \quad C_{l,k} \in \mathbf{R}^{n_{l,k} \times r_l}, \quad l = 1, \dots, L, \quad k = 1, \dots, p_l,$$

and refer to $B_{l,k}$ and $C_{l,k}$ as the left and right factors (of block k on level l). We refer to $r = r_1 + \cdots + r_L$ as the MLR-rank of A. The MLR-rank of A is in general not the same as the rank of A. We refer to $(r_1, \dots, r_L)$ as the rank allocation, *i.e.*, how the total rank r is divided across the levels. The factors $B_{l,k}$ and $C_{l,k}$ are of course not unique; we can replace them with $B_{l,k}E$ and $C_{l,k}E^{-T}$ for any invertible $r_l \times r_l$ matrix E, and we obtain the same matrix A.

Example. To illustrate our notation we give an example with $L = 3$ levels, with the second level partitioned into $p_2 = 2$ groups, and the third level partitioned into $p_3 = 4$ groups. We take $m = 10$ and $n = 8$, with block row dimensions

$$\begin{aligned}
&m_{1,1} = 10 \\
&m_{2,1} = 4, \quad m_{2,2} = 6, \\
&m_{3,1} = 2, \quad m_{3,2} = 2, \quad m_{3,3} = 4, \quad m_{3,4} = 2,
\end{aligned}$$

and block column dimensions

$$\begin{aligned}
&n_{1,1} = 8 \\
&n_{2,1} = 4, \quad n_{2,2} = 4, \\
&n_{3,1} = 2, \quad n_{3,2} = 2, \quad n_{3,3} = 2, \quad n_{3,4} = 2.
\end{aligned}$$

The sparsity patterns of A^2 and A^3 are shown below, with $*$ denoting a possibly nonzero entry, and all other entries zero. (The sparsity pattern of A^1 is full, *i.e.*, all entries are possibly nonzero.) The colors indicate how the partition is being refined when going from level $l = 2$ to level $l = 3$.

$$A^2 = \begin{bmatrix}
\ast & \ast & \ast & \ast & & & & \\
\ast & \ast & \ast & \ast & & & & \\
\ast & \ast & \ast & \ast & & & & \\
\ast & \ast & \ast & \ast & & & & \\
 & & & & \ast & \ast & \ast & \ast \\
 & & & & \ast & \ast & \ast & \ast \\
 & & & & \ast & \ast & \ast & \ast \\
 & & & & \ast & \ast & \ast & \ast \\
 & & & & \ast & \ast & \ast & \ast \\
 & & & & \ast & \ast & \ast & \ast
\end{bmatrix}, \qquad
A^3 = \begin{bmatrix}
\ast & \ast & & & & & & \\
\ast & \ast & & & & & & \\
 & & \ast & \ast & & & & \\
 & & \ast & \ast & & & & \\
 & & & & \ast & \ast & & \\
 & & & & \ast & \ast & & \\
 & & & & \ast & \ast & & \\
 & & & & \ast & \ast & & \\
 & & & & & & \ast & \ast \\
 & & & & & & \ast & \ast
\end{bmatrix}.$$

If we have ranks $r_1 = 2$, $r_2 = 1$, and $r_3 = 1$, the MLR-rank is $r = 4$. This means that A^1 has rank 2, the $p_2 = 2$ blocks in A^2 each have rank 1, and the $p_3 = 4$ blocks in A^3 also have rank 1. (Of course in practice we are interested in much larger matrices.)

2.2 Two-matrix forms

A contiguous MLR matrix is specified by its left and right factors $B_{l,k}$ and $C_{l,k}$, for $l = 1,\dots,L$ and $k = 1,\dots,p_l$. These factors can be arranged into two matrices in several ways.

Factor form. For each level $l = 1,\dots,L$ define

$$\tilde{B}_l = \mathbf{blkdiag}(B_{l,1},\dots,B_{l,p_l}) \in \mathbf{R}^{m\times p_l r_l}, \qquad \tilde{C}_l = \mathbf{blkdiag}(C_{l,1},\dots,C_{l,p_l}) \in \mathbf{R}^{n\times p_l r_l}.$$

Then we have

$$A^l = \tilde{B}_l \tilde{C}_l^T, \quad l = 1,\dots,L.$$

Define

$$\tilde{B} = \left[\begin{array}{ccc} \tilde{B}_1 & \cdots & \tilde{B}_L \end{array}\right] \in \mathbf{R}^{m\times s}, \qquad \tilde{C} = \left[\begin{array}{ccc} \tilde{C}_1 & \cdots & \tilde{C}_L \end{array}\right] \in \mathbf{R}^{n\times s},$$

with $s = \sum_l p_l r_l$. Then we can write A as

$$A = \tilde{B}\tilde{C}^T,$$

exactly the form of a low rank factorization of A. But here $\tilde{B}$ and $\tilde{C}$ have s columns, and a very specific sparsity structure, with column blocks that are block diagonal.

Compressed factor form. We can also arrange the factors into two arrays or dense matrices with dimensions $m \times r$ and $n \times r$. We vertically stack the factors at each level to form matrices

$$B^l = \left[\begin{array}{c} B_{l,1} \\ \vdots \\ B_{l,p_l} \end{array}\right] \in \mathbf{R}^{m\times r_l}, \qquad C^l = \left[\begin{array}{c} C_{l,1} \\ \vdots \\ C_{l,p_l} \end{array}\right] \in \mathbf{R}^{n\times r_l}, \quad l = 1,\dots,L.$$

We horizontally stack these matrices to obtain two matrices

$$B = \left[\begin{array}{ccc} B^1 & \cdots & B^L \end{array}\right] \in \mathbf{R}^{m\times r}, \qquad C = \left[\begin{array}{ccc} C^1 & \cdots & C^L \end{array}\right] \in \mathbf{R}^{n\times r}.$$

All of the coefficients in the factors of a contiguous MLR matrix are contained in these two matrices. To fully specify a contiguous MLR matrix, we need to give the block dimensions $m_{l,k}$ and $n_{l,k}$ for $l = 1,\dots,L$, $k = 1,\dots,p_l$, and the ranks $r_1,\dots,r_L$. We refer to the matrices B and C, together with these dimensions, as the two-matrix form of a contiguous MLR matrix.

For future use, we observe that the matrix A is a bi-linear function of B and C, *i.e.*, it is a linear function of B for fixed C, and vice versa.

2.3 Multilevel low rank matrices

A general $m \times n$ MLR matrix $\tilde{A}$ has the form

$$\tilde{A} = PAQ^T,$$

where A is a contiguous MLR matrix, $P \in \mathbf{R}^{m \times m}$ is the row permutation matrix, and $Q \in \mathbf{R}^{n \times n}$ is the column row permutation matrix. To specify an MLR matrix $\tilde{A}$ we specify the two matrices $B \in \mathbf{R}^{m \times r}$ and $C \in \mathbf{R}^{n \times r}$, the block dimensions, the ranks, and finally, the permutations P and Q.

We can think of an MLR matrix as using a general hierarchical partition of the row and column index sets I and J, with the level l partition containing p_l groups. In contrast, a contiguous MLR matrix uses hierarchical row and column partitions where the groups in the partitions are contiguous ranges.

Low rank matrix. When $L = 1$, an MLR matrix A is simply a matrix with rank no more than r, say $A = BC^T$ with $B \in \mathbf{R}^{m \times r}$ and $C \in \mathbf{R}^{n \times r}$. In this case matrix rank and MLR matrix rank agree. The associated two-matrix form uses B and C. So MLR matrices generalize low rank matrices. We will see that low rank matrices of rank r and MLR matrices of rank r share several attributes. For example both require the storage of $(m + n)r$ real coefficients.

2.4 Variations

We will consider two variations on MLR matrices.

Symmetric MLR matrices. Here we consider the special case where the matrix A is symmetric. In a symmetric MLR matrix, we require that the row and column permutations are the same, *i.e.*, $P = Q$, and all blocks $A_{l,k}$ are symmetric. This implies that $m_{l,k} = n_{l,k}$ for all l and k. The left and right factors can be chosen to be the same, up to a sign. That is we take $C_{l,k} = B_{l,k} S_{l,k}$, where $S_{l,k}$ is a diagonal sign matrix. This means we only store the matrix B (say), together with the signs in $S_{l,k}$. For a symmetric MLR matrix, we need to store only half the number of coefficients as a general square MLR matrix of the same size and rank.

Positive semidefinite MLR matrices. Here we consider a further restriction beyond symmetry: A, and each block $A_{l,k}$, is symmetric positive semidefinite (PSD). In this case we can take $B_{l,k} = C_{l,k}$.

Example: Factor covariance matrix. A common model for an $n \times n$ covariance matrix is the so-called factor model,

$$\Sigma = FF^T + D,$$

where $F \in \mathbf{R}^{n\times p}$ is the factor loading matrix and D is a diagonal PSD matrix. The dimension p is referred to as the number of factors in the model. This common model is exactly a PSD MLR matrix, with $L = 2$, and $n_{2,k} = 1$ for $k = 1,\dots,n$, $r_1 = p$ and $r_2 = 1$, so as an MLR matrix, Σ has rank $r = p+1$. We take $B_{1,1} = F$, and $B_{2,k} = \sqrt{D_{kk}}$, $k = 1,\dots,n$. (With this specific hierarchical partitioning, the permutation P does not matter, since for any permutation we obtain the same form.)

2.5 Properties

Here we briefly describe a few of the nice properties of MLR matrices. But this is not our focus, which is on fitting MLR matrices, considered in the next section.

Matrix-vector multiply. Suppose A is MLR with rank r, and we wish to evaluate Ax. The pre- and post-permutations require no floating point operations, so we can just assume that A is contiguous MLR. To evaluate Ax, we evaluate $A_{l,k}x_{l,k}$ for $l = 1,\dots,L$, $k = 1,\dots,p_l$, where $x_{l,k}$ is the appropriate subvector or slice of x. Each of these matrix-vector multiplies are carried out in an obvious way by first evaluating $z_{l,k} = C_{l,k}^T x_{l,k}$, and then forming $A^{l,k}x_{l,k} = B_{l,k}z_{l,k}$. These require around $2n_{l,k}r_l$ and $2m_{l,k}r_l$ flops each, so the total is around

$$\sum_{l=1}^{L}\sum_{k=1}^{p_l} 2(n_{l,k}+m_{l,k})r_l = \sum_{l=1}^{L} 2(n+m)r_l = 2(n+m)r$$

flops in total. (In addition there is much opportunity for carrying out the block matrix-vector multiplies in parallel.) This is the same flop count for evaluating Ax when $A = BC^T$ is a rank r matrix, again illustrating the idea that MLR matrices are generalizations of low rank matrices.

Evidently we can carry out the adjoint multiply A^Ty with the same complexity. Indeed, if A is MLR, so is A^T. To get the MLR representation of A^T from that A, we swap P and Q, B and C, and the dimensions $m_{l,k}$ and $n_{l,k}$. The ranks r_l are the same, so the rank of A^T (in the MLR sense) is the same as of A.

Solving equations and least squares. Solving systems of linear equations, least squares problems, and equality constrained least squares problems that involve MLR matrices can be done using methods that require only matrix-vector and matrix-transpose-vector multiplies, exploiting the efficiency of those operations. Alternatively, they can be reformulated as solving systems of sparse linear equations.

Consider the system of linear equations $Ax = b$, with A invertible and MLR. Using the factor form $A = \tilde{B}\tilde{C}^T$, and introducing a new variable $z \in \mathbf{R}^s$, we get two new systems of equations

$$\tilde{B}z = b, \qquad \tilde{C}^T x = z. \tag{2}$$

In replacing the system of equations $Ax = b$ with (2), we have introduced a total of $s = \sum_{l=1}^{L} r_l p_l$ new scalar variables. But each system of equations in (2) is sparse, and its sparsity pattern can be exploited by a sparse direct solver [DRSL16].

The solution of the least squares problem of choosing x to minimize $\|Ax - b\|_2^2$, is given by solving a system of normal equations $A^T(Ax - b) = 0$. Now consider the case where A is MLR. We introduce new variables $z_1, z_3 \in \mathbf{R}^s$, $z_2 \in \mathbf{R}^m$, and get an equivalent new system of equations

$$\begin{bmatrix} \tilde{C}^T & -I & 0 & 0 \\ 0 & \tilde{B} & -I & 0 \\ 0 & 0 & \tilde{B}^T & -I \\ 0 & 0 & 0 & \tilde{C} \end{bmatrix} \begin{bmatrix} x \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \tilde{C}\tilde{B}^T b \end{bmatrix}. \tag{3}$$

Note that the system of equations (3) is sparse and thus a sparse direct solver, as previously mentioned [DRSL16], can be employed. This process of expanding a system to a large sparse system is called extended sparsification [CDG⁺07, PCD17].

Lastly, consider the constrained least squares problem

$$\begin{array}{ll} \text{minimize} & \|Ax - b\|_2^2 \\ \text{subject to} & Gx = h, \end{array} \tag{4}$$

where A and G are MLR matrices. The optimality (KKT) conditions for (4) are

$$\begin{bmatrix} 2A^TA & G^T \\ G & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2A^Tb \\ h \end{bmatrix}. \tag{5}$$

Given that both A and G are MLR, the system (5) can be sparsified as in (3) and solved efficiently afterwards.

3 MLR fitting problem

3.1 General MLR fitting

The most general MLR fitting problem is

$$\begin{array}{ll} \text{minimize} & \|A - \hat{A}\|_F^2 \\ \text{subject to} & \hat{A} \text{ is rank } r \text{ MLR}, \end{array} \tag{6}$$

where $A \in \mathbf{R}^{m \times n}$ is the given matrix to be fit, $\hat{A}$ is the variable, and $\|\cdot\|_F^2$ denotes the square of the Frobenius norm, *i.e.*, the sum of squares of the entries. In this general version of the problem the data are A, the matrix to be fit, and r, the rank. We seek permutations P and Q, the number of levels L, the block dimensions $m_{l,k}$ and $n_{l,k}$, $l = 1, \ldots, L$, $k = 1, \ldots, p_l$, the two matrices B and C, and the rank allocation, *i.e.*, r_i for which $r_1 + \cdots + r_L = r$.

Variations. We consider several variations on the general MLR fitting problem (6). In the symmetric MLR fitting problem, we assume that $m = n$, the matrix A is symmetric, and we require that $\hat{A}$ be a symmetric MLR matrix. In the PSD MLR fitting problem, we assume in addition that $\hat{A}$ is a PSD MLR matrix, and require that $\hat{A}$ is a PSD MLR matrix.

3.2 Factor fitting and rank allocation

We also consider variations in which some parameters in the general MLR fitting problem (6) are fixed, which means we optimize over fewer parameters. Each of these variations can also be restricted to symmetric or PSD MLR matrices.

Factor fitting problem. In the factor fitting problem we fix the hierarchical partition, *i.e.*, the permutations P and Q, the number of levels L, and the block dimensions $m_{l,k}$ and $n_{l,k}$, and we also fix the rank allocation, *i.e.*, $r_1, \ldots, r_L$. We optimize over the factors, *i.e.*, the matrices B and C. Roughly speaking, in the factor fitting problem we fix the combinatorial aspects of $\hat{A}$, and only optimize over the (real) coefficients in the factors.

Rank allocation problem. In the rank allocation problem we fix the hierarchical partition but not the ranks. This means we optimize over B and C, and also the ranks $r_1, \ldots, r_L$, subject to $r_1 + \cdots + r_L = r$. The rank allocation problem is a natural one when the hierarchical row and column partitions are given or already known.

Complexity. Aside from the special case of low rank matrices described below, we do not hope to solve the fitting problem exactly, but only develop good heuristic methods for it. In the next section we describe a method for the most constrained problem, factor fitting. In the two following sections we extend that to rank allocation, and the full fitting problem (6).

3.3 Special case: Low rank matrix

We can solve the MLR fitting problem exactly when there is one level, so rank r MLR reduces to rank r. The well-known solution is found from the singular value decomposition (SVD) of A, $A = U\Sigma V^T$. An optimal rank r approximation is $\hat{A} = BC^T$,

$$B = U_r \Sigma_r^{1/2}, \qquad C^T = \Sigma_r^{1/2} V_r,$$

where U_r is the first r columns of U, V_r is the first r columns of V, and Σ_r is the leading $r \times r$ submatrix of Σ. The optimal (minimum) objective value is $\sum_{i=r+1}^{\min\{m,n\}} \sigma_i^2$.

The symmetric and PSD versions of the low rank matrix fitting problem can also be solved exactly. Here we use an eigendecomposition of A, $A = Q\Lambda Q^T$, with Q an orthogonal matrix of eigenvectors and Λ a diagonal matrix of eigenvalues, sorted from largest to smallest in absolute value. In the symmetric case we take $B = Q_r|\Lambda_r|^{1/2}$ and $C = BS$, where Q_r is the leading $n \times r$ submatrix of Q, Λ_r is the leading $r \times r$ submatrix of Λ, and S is a diagonal $r \times r$ sign matrix, with $S_{ii} = \mathbf{sign}\, \Lambda_{ii}$. In the PSD case, we use the eigendecomposition, with the entries of Λ sorted from largest to smallest. We replace Λ_r in the symmetric case with $\max(\Lambda_r, 0)$, elementwise. We will leverage these analytical solutions for the special case of low rank matrices in the methods we propose for solving the general MLR fitting problem.

Partial SVDs. The singular value and eigenvalue decompositions described above can be computed in full, which has a complexity of order $\min\{mn^2, m^2n\}$ flops. In most cases, however, the target rank is substantially smaller than $\min\{m, n\}$, in which case methods for computing only a set of extremal singular values or eigenvalues and their associated singular vectors and eigenvectors can be used [Sor92, GVL96, LSY98]. These methods have complexity that can be far smaller than computing a full SVD or eigendecomposition. (They can also be warm-started, which is an advantage in the context of the methods we describe later.)

Alternating least squares. We mention for future use that the best rank r approximation can also be computed using alternating least squares, applied to the objective

$$\|A - BC^T\|_F^2,$$

with variables $B \in \mathbf{R}^{m\times r}$ and $C \in \mathbf{R}^{n\times r}$. The method alternates between minimizing the objective over B, with C fixed, and then minimizing the objective over C, with B fixed. These are both simple least squares problems with analytical formulas for the solution. Moreover when we minimize over C, the least squares problem splits into separate ones for each row of C, which can be solved in parallel, and similarly for B.

4 Factor fitting methods

In the factor fitting problem we are given A, the matrix to be approximated, and for the MLR approximator $\hat{A}$, we are given the permutations P and Q, the number of levels L, the block dimensions, and the ranks. We are to choose the matrices B and C, *i.e.*, the factors $B_{l,k}$ and $C_{l,k}$ to minimize $\|A - \hat{A}\|_F^2$. Since P and Q are given, we can just as well fit $\tilde{A} = P^T AQ$ with a contiguous MLR matrix $\hat{A}$. We can also have an initial guess of the factors.

4.1 Alternating least squares

Let B and C be the two matrices associated with the MLR-rank r approximation $\hat{A}$. We write this as $\hat{A}(B, C)$, and recall that $\hat{A}$ is bi-linear, *i.e.*, using the factor form we have $\hat{A} = \tilde{B}\tilde{C}^T$. We can use an alternating least squares (ALS) algorithm to minimize

$$\|A - \hat{A}(B, C)\|_F^2 \tag{7}$$

over B, then C, then B, etc.

There are many possible ways to carry out the minimization over C (with fixed B). We can get a closed-form solution for the least squares problems for each row of $\tilde{C}$ while taking into account the sparsity of $\tilde{C}$. Another approach is to use an iterative method such as conjugate gradients (CG) applied to the normal equations associated with minimizing (7). An iterative approach allows us to solve the minimization over B or C approximately, which is appropriate since the least squares problems solved are just one step in an iterative algorithm.

Algorithm 4.1 ALTERNATING LEAST SQUARES FOR FACTOR FITTING

```
given $A \in \mathbf{R}^{m\times n}$, permutations $P$, $Q$, block sizes $m_{l,k}$ and $n_{l,k}$, ranks $r_1, \dots, r_L$,
and initial factors $B_{l,k}$ and $C_{l,k}$, $l = 1, \dots, L$, $k = 1, \dots, p_l$.
Permute to contiguous form. Form $\tilde{A} = P^T AQ$.
for iteration/epoch $t = 1, 2, \dots$
Check stopping criterion. Quit if (9) holds.
Approximately minimize (7) over $B$.
Approximately minimize (7) over $C$.
```

We have experimented with several methods for approximate minimization of (7) and found that taking 10 steps of CG performs well over a wide range of problems. This alternative least squares method converges to B and C that are locally optimal, in the sense that no other choice of B will decrease the objective for the given C, and vice versa. But it need not be a global solution to the factor fitting problem.

4.2 Block coordinate descent

We can use a block coordinate descent (BCD) algorithm, updating the factors in one level in each iteration. As we will see, we can carry out exact minimization over the factors on any one level. To optimize over the factors in level l, we need to choose $B_{l,k}$ and $C_{l,k}$ to minimize

$$\left\| R - \mathbf{blkdiag}(B_{l,1}C_{l,1}^T, \ldots, B_{l,p_l}C_{l,p_l}^T) \right\|_F^2, \tag{8}$$

where

$$R = \tilde{A} - \sum_{j \neq l} \mathbf{blkdiag}(B_{j,1}C_{j,1}^T, \ldots, B_{j,p_j}C_{j,p_j}^T)$$

is the current residual. This problem separates into p_l independent problems, where we minimize $\|R_{l,k} - B_{l,k}C_{l,k}^T\|_F^2$, where $R_{l,k}$ is the submatrix of R corresponding to the l, k block. But this we know how to do exactly, as described above, using the SVD of $R_{l,k}$, or the eigenvalue decomposition for the symmetric and PSD cases. This step can be carried out in parallel for all p_l blocks in level l. Moreover we can use SVD methods that allow us to compute only an appropriate number of the largest singular values and vectors, and not a full SVD [GVL96].

There are many possible choices for the order in which we update the levels. We have found that an effective method is to start from level $l = 1$, proceed in order to level L, and then in reverse order update levels from $l = L - 1$ back to $l = 1$. In many cases one such V-shape sweep, which we refer to as an epoch, suffices; if not, it can be repeated. We stop when the relative fit does not improve much over an epoch, *i.e.*,

$$\frac{\|A - \hat{A}^{\text{prev}}\|_F}{\|A\|_F} - \frac{\|A - \hat{A}\|_F}{\|A\|_F} \leq \epsilon_{\text{rel}} \frac{\|A - \hat{A}^{\text{prev}}\|_F}{\|A\|_F}, \tag{9}$$

where ϵ_{rel} is a positive parameter, $\hat{A}$ is the current approximation, and $\hat{A}^{\text{prev}}$ is the approximation in the previous epoch.

Algorithm 4.2 BLOCK COORDINATE DESCENT FOR FACTOR FITTING

```
given $A \in \mathbf{R}^{m \times n}$, permutations $P$, $Q$, block sizes $m_{l,k}$ and $n_{l,k}$, ranks $r_1, \ldots, r_L$,
and initial factors $B_{l,k}$ and $C_{l,k}$, $l = 1, \ldots, L$, $k = 1, \ldots, p_l$.
Permute to contiguous form. Form $\tilde{A} = P^T A Q$.
for iteration/epoch $t = 1, 2, \ldots$
Check stopping criterion. Quit if (9) holds.
Carry out V-epoch. for $l = 1, \ldots, L-1, L, L-1, \ldots, 1$
    Form current residual. $R = \tilde{A} - \sum_{j \neq l} \mathbf{blkdiag}(B_{j,1}C_{j,1}^T, \ldots, B_{j,p_j}C_{j,p_j}^T)$.
    Update $B_{l,k}$ and $C_{l,k}$ for $k = 1, \ldots, p_l$ by minimizing (8).
```

This is a descent method and converges to at least a local minimum. We have observed that, depending on the initialization, it can converge to different local minima, even with somewhat different final objective values. We have also observed that the BCD method works well when all factors are initialized at 0, when no better initial guess is available.

Example: Factor covariance fitting. The BCD algorithm takes an interesting form when applied to the factor covariance fitting problem. Here A is the empirical covariance matrix and $\hat{A}$ will have the form $\hat{A} = FF^T + \mathbf{diag}(d)$, with $F \in \mathbf{R}^{n \times p}$ and d nonnegative. If we initialize with $F = 0$ and $d = 0$ the first four BCD steps are

1. Set F as the rank-p PSD approximant of A.
2. Set $d = \mathbf{diag}(A) - \mathbf{diag}(FF^T)$.
3. Set F as the rank-p PSD approximant of $A - \mathbf{diag}(d)$.
4. Set $d = \max(\mathbf{diag}(A) - \mathbf{diag}(FF^T), 0)$.

In step 2 we are guaranteed that d is nonnegative, since $A - FF^T$ is PSD. Steps 1 and 2 are the typical method for fitting a factor model to a given covariance matrix A. The choice of d guarantees that A and $\hat{A}$ agree on the diagonal entries, *i.e.*, $\hat{A}$ captures exactly the (marginal) variance of each component. Steps 3 and 4 (and more, if the iteration continues) are not traditional, but can further reduce the Frobenius approximation error.

4.3 Comparison

In this section we show some typical convergence results for the ALS and BCD fitting methods, using as an example a matrix A of size $m = 5000$ and $n = 7000$, fitting with an MLR matrix with $L = 14$ levels and ranks $r_1 = \cdots = r_{14} = 5$. (The matrix is a discrete Gauss transform, described in §8.3.) The matrices B and C are initialized as the matrices resulting from a single sweep of BCD from $l = 1$ to $l = 14$.

Figure 1 shows the relative fitting loss versus iterations. For the ALS method, one iteration is approximately minimizing over B and then over C (using 10 steps of CG). For the BCD method, one iteration is one V-epoch. The convergence of the two methods is quite similar, at least in terms of iterations.

The total time, however, is very much dependent on the implementation, how much parallelism is exploited, and the computing platform used. Our non-optimized implementation uses SciPy [VGO+20] for the partial SVD in BCD, and PyTorch [PGM+19] to compute the gradient of the quadratic loss function (7) during the CG step of ALS. It requires on the order of 10s of seconds for each iteration of ALS or BCD.

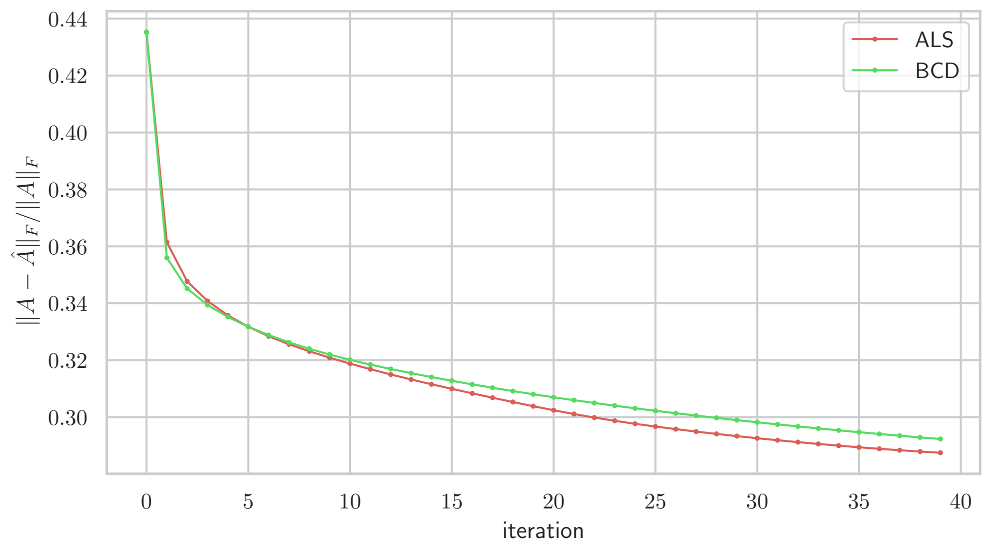

Figure 1: Comparison of BCD and ALS. For ALS, one iteration is approximately minimizing over B and then over C. For BCD, one iteration is one V-epoch.

5 Rank allocation

In this section we describe a method for adjusting the ranks, subject to $r_1 + \cdots + r_L = r$, in order to reduce the Frobenius norm of the approximation error.

5.1 Incrementing and decrementing level rank

We start with any valid allocation of rank, such as $r_1 = r$ and $r_2 = \cdots = r_L = 0$. We use the factor BCD method above to optimize the factors for this rank allocation. (Results with ALS are similar.) We then find an approximation of how much better the approximation would be, if we were to increase the rank allocated to level l by one. We also obtain an approximation of how much worse the approximation would be if we were to decrease the rank allocated to level l by one. (If $r_l = 0$ it is infeasible to reduce the rank allocated to it.)

To do this, recall that the minimum Frobenius norm squared approximation of a matrix by a rank r one is $\sum_{i=r+1}^{\min\{m,n\}} \sigma_i^2$, where σ_i are the singular values of the matrix. If we increase the rank of the approximation by one (assuming $r < \min\{m, n\}$), to $r + 1$, it follows that the Frobenius norm square fitting error decreases by σ_{r+1}^2. If we decrease the rank of the approximation by one (assuming $r > 0$) to $r - 1$, it follows that the fitting error increases by σ_r^2. If we increase the rank allocated to (each block of) level l by 1, the decrease in Frobenius norm squared error is

$$\delta_l^+ = \sum_{k=1}^{p_l} \sigma_{r_l+1}^2(R_{l,k}).$$

If we reduce the rank allocated to level l, the increase in Frobenius norm squared error is

$$\delta_l^- = \sum_{k=1}^{p_l} \sigma_{r_l}^2(R_{l,k}).$$

These numbers predict the decrease or increase when only the level l factors are changed; by using BCD, the Frobenius norm squared error can only decrease. Nevertheless we have found these numbers to give good enough estimates of the changes for our rank allocation algorithm to be effective.

5.2 Rank exchange algorithm

In each iteration of the rank allocation algorithm, we find the level i and j, with $i \neq j$, for which the predicted net decrease is maximized, *i.e.*,

$$i, j = \underset{i \neq j}{\operatorname{argmax}} \left(\delta_i^+ - \delta_j^-\right).$$

We then carry out this suggested rank re-allocation or rank exchange, by increasing r_i by one and decreasing r_j by one. We then use the BCD factor fitting method, in warm start mode, to update the factors to further decrease the fitting error. We reject the update if the fitting error with the new rank allocation is worse than the current one, and quit.

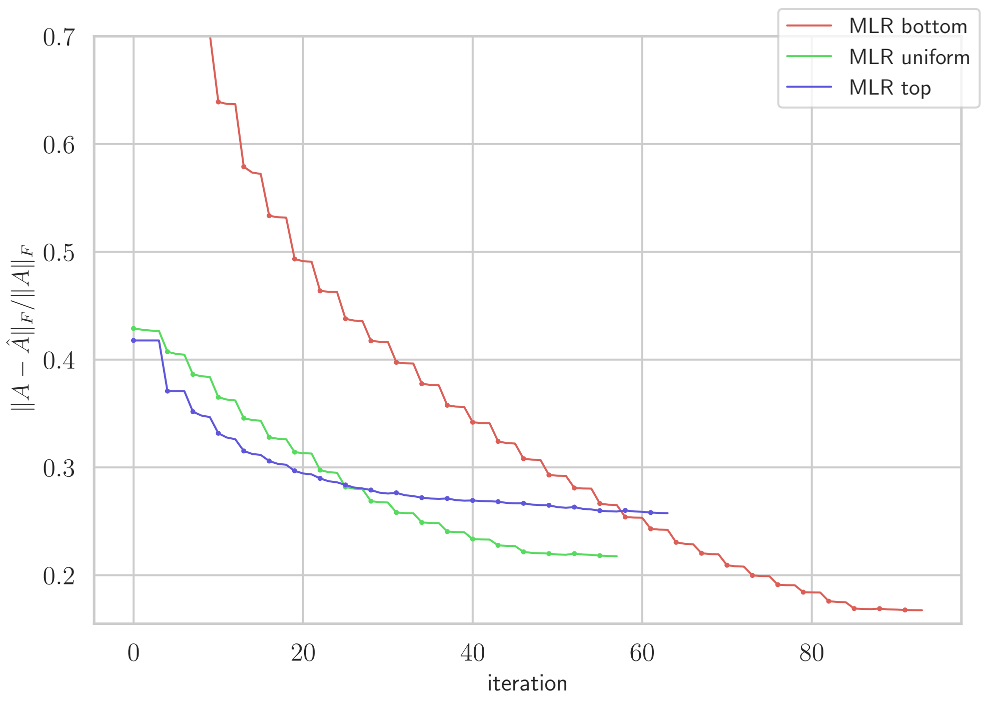

Figure 2: Fitting error during rank allocation, starting from three different initial allocations of rank.

The PSD version of this method tends to converge faster, since it often occurs that $\delta_l^+ = 0$, *i.e.*, allocating more rank to level l would not improve the current approximation error. (This occurs when the current level has a lower rank than is allocated to it.)

Example. Figure 2 shows some typical fitting error trajectories during rank allocation, for a matrix with $m = 5000$ and $n = 7000$, where we allocate a total rank $r = 28$ to $L = 14$ levels. The horizontal axis shows BCD iterations. The point markers on the curves show the iterations in which rank was re-allocated. We show three trajectories, corresponding to different initializations. The one labeled MLR bottom has initial allocation of rank entirely to the bottom level, *i.e.*, $r_{14} = 28$, $r_1 = \cdots = r_{13} = 0$. The one labeled MLR uniform has initial allocation of rank uses $r_1 = \cdots = r_{14} = 2$. The one labeled MLR top has initial allocation of rank $r_1 = 28$, $r_2 = \cdots = r_{14} = 0$. (The particular allocations of rank for these can be seen in figure 7.) We observe that different rank initializations converge to different local minima, with different loss values. For this example, the rank allocation obtained starting from the allocation of rank to the bottom yields the best fitting error.

6 A method for general MLR fitting

In this section we describe a method for the full MLR fitting problem, where only A, the matrix to be approximated, and a rank r, are given. The method constructs the hierarchy one level at a time, starting from the top level $l = 1$. We first describe the general method. For simplicity we describe the specific case when each block on one level is split into two nearly equal sized blocks on the next level, *i.e.*, we use nested dissection on the rows and columns. This means $p_l = 2^{l-1}$, and $L = \lceil \log_2 \min\{m, n\} \rceil + 1$. (The final approximation may have fewer levels than this initial maximum value.) The row dimensions $m_{l,k}$ are approximately $m2^{-(l-1)}$, and the column dimensions $n_{l,k}$ are approximately $n2^{-(l-1)}$. We start with an initial rank allocation that is nearly uniform across levels, *i.e.*, $r_l \approx r/L$.

We first form a rank r_1 approximation of A, and subtract it from A to form a residual matrix R.

The next step is to partition the rows and columns each into two nearly equal sized groups $I_{2,1}$ and $I_{2,2}$, and $J_{2,1}$ and $J_{2,2}$, so as to maximize the sum of the squares of the residuals within the two groups,

$$\sum \left\{ R_{ij}^2 \mid (i,j) \in I_{2,1} \times J_{2,1} \text{ or } (i,j) \in I_{2,2} \times J_{2,2} \right\}.$$

The sum here includes about half the pairs $(i, j) \in I \times J$. We do not need to solve this partitioning problem exactly; we describe below a simple spectral method, but any method can be used. We then permute the rows and columns so the partition is contiguous, *i.e.*, $I_{2,1} = \{1, \ldots, m_{2,1}\}$ and $I_{2,2} = \{m_{2,1} + 1, \ldots, m\}$ and similarly for the columns indices. We can interpret this step as finding row and column permutations that give the best block diagonal Frobenius norm approximation of the residual matrix. Now we use the BCD factor fitting method to update the factors on levels $l = 1$ and $l = 2$.

We now apply the same method to each of the $p_2 = 2$ blocks on level $l = 2$. For each of the two blocks we get the current residual, and then partition the rows and columns within each block to maximize the sum of squares of residuals within the blocks. Then we permute rows and columns within each block so the partition is contiguous. We use the BCD factor fitting method to update the factors on levels $l = 1$, $l = 2$, and $l = 3$.

We continue this way, in each step partitioning the rows and columns within each block so as to maximize the sum of squares of the current residuals within each group of the partition, and then permuting the rows and columns so the partitions are contiguous. Finally, we use BCD factor fitting to update all the factors from the top to the current level.

When we finish with level L, we have our first MLR approximation of A. We can now run rank allocation to further improve the fit.

The same method can be used for the symmetric MLR fitting problem, using the same partitioning of the rows and columns within each block.

6.1 Symmetric row and column dissection

Here we describe a simple heuristic for partitioning the rows and columns of a matrix, each about equally, so as to maximize the sum of squares of a given residual matrix.

We start with the symmetric case, where the method is just spectral partitioning. Spectral partitioning goes back to the early 1970s [Hal70, DH73, Fie73] and was later popularized in the 1990s through seminal contributions of [PSL90, Sim91, BS94]. We review this here, mostly to explain the generalization to the non-symmetric case later.

For simplicity we assume that n is even. Let S be the elementwise square of the (symmetric) residual matrix R. We represent the partition as a vector $x \in \{-1, 1\}^n$, with $x_i = -1$ meaning i is in the first group, and $x_i = 1$ meaning i is in the second. The two groups have equal size provided $\mathbf{1}^T x = 0$. We observe that

$$x^T S x = \sum_{i,j} x_i x_j R_{ij}^2 = \sum_{x_i = x_j} R_{ij}^2 - \sum_{x_i \neq x_j} R_{ij}^2 = 2 \sum_{x_i = x_j} R_{ij}^2 - \|R\|_F^2.$$

The problem we wish to solve is

$$\begin{array}{ll} \text{maximize} & x^T S x \\ \text{subject to} & x_i \in \{-1, 1\}, \quad i = 1, \ldots, n, \quad \mathbf{1}^T x = 0, \end{array}$$

with variable x. Changing the diagonal entries of S only adds a constant to the objective, and so does not change the solution. So we change each diagonal entry of S to be the negative of the sum of the other entries in the row. This results in the negative of a Laplacian matrix we denote as L, with $L\mathbf{1} = 0$. Substituting L for S in the problem above (and minimizing since we have changed the objective sign) yields an equivalent problem,

$$\begin{array}{ll} \text{minimize} & x^T L x \\ \text{subject to} & x_i \in \{-1, 1\}, \quad i = 1, \ldots, n, \quad \mathbf{1}^T x = 0. \end{array}$$

For any x with $x_i \in \{-1, 1\}$ we have $\|x\|_2 = \sqrt{n}$. Now we relax the problem to

$$\begin{array}{ll} \text{minimize} & x^T L x \\ \text{subject to} & \|x\|_2^2 = n, \quad \mathbf{1}^T x = 0, \end{array}$$

with variable $x \in \mathbf{R}^n$. We will assume that $L_{ij} < 0$ for all $i \neq j$ (which we can ensure by subtracting a small number from each entry), which implies that the second smallest eigenvalue of L is positive. With this assumption, the solution of the problem above is $x = \sqrt{n} v$, where v is the eigenvector of the Laplacian associated with its second smallest eigenvalue. Since $L\mathbf{1} = 0$, we see that $\mathbf{1}$ is an eigenvector of L, so we get $\mathbf{1}^T x = 0$ automatically.

The final step is to project or round this x back to our original constraints $x_i \in \{-1, 1\}$ while maintaining $\mathbf{1}^T x = 0$. This is done by sorting x and taking $x_i = -1$ for the first $n/2$ entries and $+1$ for the remaining entries. (The same method works when n is odd; we can arbitrarily choose the group that has one more index than the other.)

It is typically followed with a greedy algorithm, where we consider swapping a pair, one from each group, and accepting the new partition if the objective decreases. Finding the change in cost for all such pairs requires order n^2 flops.

6.2 Non-symmetric row and column dissection

We now describe a generalization of spectral partitioning to the case where we wish to partition both the rows and columns. Again for simplicity we assume that m and n are even. We use $u \in \mathbf{R}^m$ and $v \in \mathbf{R}^n$ to denote the row and column partitions, with $u_i, v_i \in \{-1, 1\}$. The constraints $\mathbf{1}^T u = \mathbf{1}^T v = 0$ guarantee that the partitions of rows and columns have equal size. We observe that

$$u^T S v = \sum_{i,j} u_i v_j R_{ij}^2 = \sum_{u_i = v_j} R_{ij}^2 - \sum_{u_i \neq v_j} R_{ij}^2 = 2 \sum_{u_i = v_j} R_{ij}^2 - \|R\|_F^2 = \|R\|_F^2 - 2 \sum_{u_i \neq v_j} R_{ij}^2.$$

Our goal is to maximize $u^T S v$, subject to $u_i, v_i \in \{-1, 1\}$, $\mathbf{1}^T u = \mathbf{1}^T v = 0$. This is closely related to [Dhi01, ZHD+01] which minimizes the normalized sum of edge weights crossing partitions, while our method does not incorporate normalization.

As with spectral partitioning, we first modify S to $\tilde{S}$ in a way that does not change the solution of the problem, but results in $\tilde{S}\mathbf{1} = 0$ and $\tilde{S}^T \mathbf{1} = 0$. This is analogous to changing the diagonal entries of S in the symmetric case, so that $S\mathbf{1} = 0$. We then will minimize $u^T M v$, with $M = -\tilde{S}$. (The matrix M is analogous to L in the symmetric case, but it is not a Laplacian matrix.) For any $a \in \mathbf{R}^m$ and $b \in \mathbf{R}^n$ we have

$$u^T S v = u^T \tilde{S} v, \qquad \tilde{S} = S - a\mathbf{1}^T - \mathbf{1}b^T,$$

for any u and v with $\mathbf{1}^T u = \mathbf{1}^T v = 0$. The choice

$$a = (1/n)(S\mathbf{1} - (\mathbf{1}^T S \mathbf{1}/2m)\mathbf{1}), \qquad b = (1/m)(S^T\mathbf{1} - (\mathbf{1}^T S\mathbf{1}/2n)\mathbf{1})$$

yields the desired result, $\tilde{S}\mathbf{1} = 0$ and $\tilde{S}^T\mathbf{1} = 0$.

As in spectral partitioning, we form the relaxed problem

$$\begin{array}{ll} \text{minimize} & u^T M v \\ \text{subject to} & \|u\|_2^2 = m, \quad \|v\|_2^2 = n, \quad \mathbf{1}^T u = \mathbf{1}^T v = 0, \end{array}$$

where $M = -\tilde{S}$. The solution of this problem is to choose u and v to be (scaled) left and right singular vectors of M associated with its smallest (positive) singular value. Since $M\mathbf{1} = 0$ and $M^T\mathbf{1} = 0$, these vectors are left and right singular vectors of M associated with singular value zero. Thus we automatically have $\mathbf{1}^T u = \mathbf{1}^T v = 0$.

The final step is to round u_i and v_i to Boolean values. We do that setting the $m/2$ smallest entries of u to be -1, and the remaining ones $+1$, and similarly for v. We can also follow this method with a greedy method in which we alternatively consider swapping each pair of rows, one from each group, and each pair of columns, one from each group.

7 Extensions and variations

In this section we introduce various extensions for MLR and related methods. In particular we present an extension of the rank allocation method, and provide generalizations of MLR matrices that include different sparsity patterns and recursion on the block terms.

7.1 Rank allocation

In §5 we introduce a method for distributing MLR-rank across all levels. This method operates by identifying two distinct levels for which rank exchange maximizes the predicted net decrease. More specifically, it is achieved by exchanging one unit of rank between two levels. The idea can be extended to identify two distinct levels that can swap from 1 to q units of rank to maximize the decrease in Frobenius norm squared error. The corresponding decrease, δ_l^+, and increase, δ_l^-, in error can be computed directly using

$$\delta_l^+ = \sum_{k=1}^{p_l}\sum_{j=1}^{q} \sigma^2_{r_l+j}(R_{l,k}), \qquad \delta_l^- = \sum_{k=1}^{p_l}\sum_{j=1}^{q} \sigma^2_{r_l-j-1}(R_{l,k}).$$

This strategy offers greater flexibility in re-allocating ranks. Lastly, we can retain several top candidates for rank exchange and terminate the rank allocation algorithm if none of them decreases the fitting error.

7.2 Sparsity refinement

In the MLR definition, the partition at each level is hierarchical. This implies that the sparsity pattern of each level's block diagonal matrix A^l refines the sparsity pattern found in the preceding level's block diagonal matrix A^{l-1}. However, we can relax the refinement constraint and only require A^l to be a block diagonal matrix. Given that A decomposes onto a sum of matrices A^l, the BCD algorithm remains directly applicable. Moreover, since A^l maintains its block diagonal form, A can also be expressed in factor form, making the ALS method equally suitable. It is straightforward to see that the rank allocation algorithm can be readily applied in this context as well.

7.3 Generalized multilevel low rank matrix

We further generalize the definition of MLR as follows. Define a tail sum as $\bar{A}^l = \sum_{l'=l}^{L} A^{l'}$. The definition of A in §2.1 can then be alternatively presented as a recursion

$$\bar{A}^l = \mathbf{blkdiag}(\bar{A}_{l,1}, \dots, \bar{A}_{l,p_l}) = \mathbf{blkdiag}(B_{l,1}C_{l,1}^T, \dots, B_{l,p_l}C_{l,p_l}^T) + \bar{A}^{l+1}, \tag{10}$$

where $B_{l,k} \in \mathbf{R}^{m_{l,k}\times r_l}$ and $C_{l,k} \in \mathbf{R}^{n_{l,k}\times r_l}$, $\bar{A}_{l,k}$ is $m_{l,k} \times n_{l,k}$ block of $\bar{A}^l$ restricted to rows $\sum_{k'=1}^{k-1} m_{l,k'}, \dots, \sum_{k'=1}^{k} m_{l,k'}$ and to columns $\sum_{k'=1}^{k-1} n_{l,k'}, \dots, \sum_{k'=1}^{k} n_{l,k'}$. Using the recursion in (10), we can write

$$A = \bar{A}^1 = B_{1,1}C_{1,1}^T + \mathbf{blkdiag}(\bar{A}_{2,1}, \dots, \bar{A}_{2,p_2}).$$

The recursion process in (10) occurs within the block diagonal term. This process can be further generalized to include a recursion for each block derived from row and column partitioning,

$$\bar{A}^l = \begin{bmatrix} \bar{A}_{l,1,1} & & \bar{A}_{l,1,p_l} \\ & \ddots & \\ \bar{A}_{l,p_l,1} & & \bar{A}_{l,p_l,p_l} \end{bmatrix} = \begin{bmatrix} B_{l,1,1}C_{l,1,1}^T & & B_{l,1,p_l}C_{l,1,p_l}^T \\ & \ddots & \\ B_{l,p_l,1}C_{l,p_l,1}^T & & B_{l,p_l,p_l}C_{l,p_l,p_l}^T \end{bmatrix} + \bar{A}^{l+1}, \tag{11}$$

where $B_{l,k_1,k_2} \in \mathbf{R}^{m_{l,k_1} \times r_l}$ and $C_{l,k_1,k_2} \in \mathbf{R}^{n_{l,k_2} \times r_l}$, $\bar{A}_{l,k_1,k_2}$ is $m_{l,k_1} \times n_{l,k_2}$ block of $\bar{A}^l$ restricted to rows $\sum_{k'=1}^{k_1-1} m_{l,k'}, \ldots, \sum_{k'=1}^{k_1} m_{l,k'}$ and to columns $\sum_{k'=1}^{k_2-1} n_{l,k'}, \ldots, \sum_{k'=1}^{k_2} n_{l,k'}$. Similarly, we have

$$A = \bar{A}^1 = B_{l,1,1}C_{l,1,1}^T + \begin{bmatrix} \bar{A}_{2,1,1} & & \bar{A}_{2,1,p_2} \\ & \ddots & \\ \bar{A}_{2,p_2,1} & & \bar{A}_{2,p_2,p_2} \end{bmatrix}.$$

We refer to the format in (11) as the generalized multilevel low rank matrix (GMLR).

Since A decomposes into a sum of block matrices at each level with each block of a given rank, the BCD algorithm remains applicable. Similarly, A is bilinear in B_{l,k_1,k_2} and C_{l,k_1,k_2}, and therefore ALS scheme can still be applied to GMLR. Furthermore, given the total rank and matrix partitioning at each level, the rank allocation method directly extends to this setting. In particular, estimating the increase δ_l^+ (or decrease δ_l^-) in fitting error now involves singular values of all blocks on a given level, rather than just the block diagonal blocks as in MLR. Hence, while the rank allocation method in §5 was developed for MLR matrices, it can be readily applied to more general settings, such as GMLR.

We can further relax the rank constraint to allow blocks at the same level to have different ranks. Consequently, the GMLR without the rank constraint contains a class of hierarchical matrices $\mathcal{H}$ as well as BLR and multilevel BLR.

8 Numerical examples

In this section we illustrate our methods with several numerical examples. We report the relative fitting error $\|A - \hat{A}\|_F / \|A\|_F$ and rank allocation evolution versus the number of iterations. We use three different initial allocations of rank, MLR bottom, MLR uniform, and MLR top, as described in §5. We also report the error of the low rank approximation, denoted as LR. For square matrices we report the low rank plus diagonal approximation (denoted as LR+D) or factor model in the symmetric PSD case.

The stopping criterion for factor fitting (9) is $\epsilon_{\text{rel}} = 0.01$ and for rank allocation, $\epsilon_{\text{rel}} = 0.001$. In factor fitting for the low rank plus diagonal model, we use $\epsilon_{\text{rel}} = 10^{-6}$. For each iteration of rank allocation, we carry out 2 V-epochs of BCD during factor fitting. In §8.1 we use a given hierarchical partitioning; in §8.5 the hierarchical partitioning is derived based on recursive spectral bi-clustering of a spatial domain, while in the remaining examples, we employ hierarchical partitioning found by spectral dissection, as detailed in §6. In the process of constructing a hierarchical partitioning, we refine each partitioning of the rows and columns (found by spectral dissection) using a greedy algorithm for at most 5000 pair swaps that decrease the objective.

We compare the performance of MLR with HODLR and Monarch matrices. We report the fitting errors associated with HODLR and Monarch matrices, while matching their storage with MLR. HODLR is specified by the block fitting tolerance and hierarchical partitioning. We conduct a comparison with the HODLR, adjusting its tolerance parameter to approximately match the storage of other formats. In every experiment, we use the same hierarchical partitioning for HODLR that we do for the MLR with the best fitting error. Note that due to the difference in specification of the formats (MLR, LR and LR+D are defined by rank, while HODLR is defined by tolerance), the comparison with HODLR may not be entirely fair.

We employ the official implementation from [DCS+22] to find an analytical solution for fitting a dense matrix using a Monarch matrix. We apply a global permutation to the input matrix given by its hierarchical partitioning, as it results in a smaller fitting error compared to an unpermuted matrix. We search over several parameters like block sizes and rank, and we report the best fitting error among them. Nevertheless, due to the difference in format specification with MLR, this comparison is not entirely fair.

8.1 Asset covariance matrix

We consider the problem of fitting the empirical covariance matrix of daily stock returns. For this example there is a widely used hierarchical partition of the assets, the Global Industry Classification Standard (GICS) [BLO03], based on the company's primary business activity. This hierarchical partition has $L = 6$ levels, shown in figure 3.

In this example the daily returns are found or derived from data from CRSP Daily Stock and CRSP/Compustat Merged Database ©2023 Center for Research in Security Prices (CRSP®), The University of Chicago Booth School of Business. We consider a 300 (trading) day period ending 2022/12/30. We obtain the GICS codes from CRSP/Compustat Merged

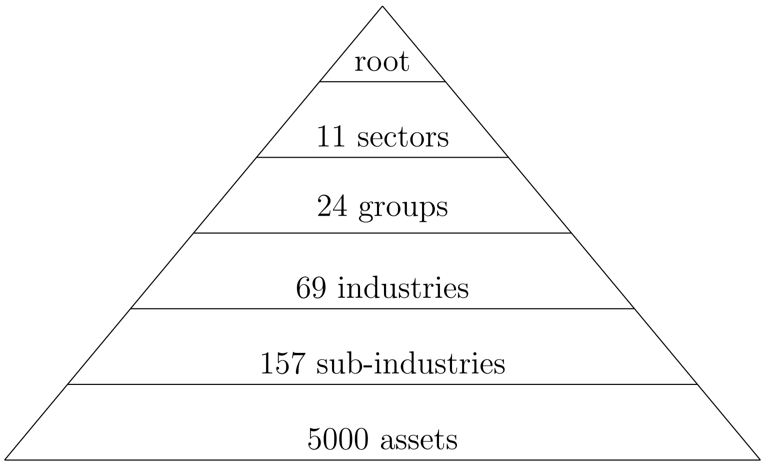

Figure 3: GICS hierarchical partition of 5000 companies.

Database – Security Monthly during 2022/06/30 to 2023/01/31. We joined the two tables using PERMCO (a unique permanent company identification number assigned by CRSP to all companies). Next we filtered out assets with missing GICS codes, and kept the $m = 5000$ assets with the fewest missing return values. (We understand that this introduces survival bias, but our goal here is only to illustrate MLR fitting, and not build a covariance model.) We clip or Winsorize the entries at $\pm 3\sigma$, where σ is the standard deviation of return of each stock over the period. We let Σ be the 5000×5000 empirical covariance matrix of the returns. Since it is based on only 300 days, its rank is 300. We fit Σ with a PSD MLR matrix of total rank $r = 30$.

Results. A factor model is the traditional approach for modeling a covariance matrix of financial returns [Cou13]; we were curious to see if a different MLR approximation might give a better fit to the empirical covariance matrix. Using the GICS hierarchical partition, we ran rank allocation from 3 different initial allocations (see table 1). In all three cases the method converged to the factor model, *i.e.*, $r_1 = 29$, $r_2 = \cdots = r_5 = 0$ and $r_6 = 1$ (see figure 4). In particular, the GICS hierarchy appears to bring no advantage over the standard factor model, at least in terms of the Frobenius norm fitting of the empirical covariance matrix. (Still, the GICS classification is widely used in portfolio construction [BLO03, BS$^+$16] and in finding stable eigenportfolios [Ave19, SA20].)

We also ran our full MLR fitting method, ignoring the GICS hierarchy, fitting an MLR matrix with $L = 14$ levels. For two out of the three initial rank allocations, our method converged again to the factor model; see table 2 and figure 5. We take these results as an endorsement of the commonly used factor model.

The fitting error achieved by HODLR is at minimum twice as large as that of low rank approximation across both hierarchical partitionings. The Monarch fitting error is comparable but worse than that of the LR.

Method	Error (%)	Storage ($\times 10^5$)
LR	16.178	1.500
LR+D	**15.379**	1.500
HODLR	38.837	1.503
Monarch	17.971	1.560
MLR bottom	**15.379**	1.500
MLR uniform	**15.379**	1.500
MLR top	**15.379**	1.500

Table 1: Fitting errors for asset covariance matrix with GICS hierarchical partition.

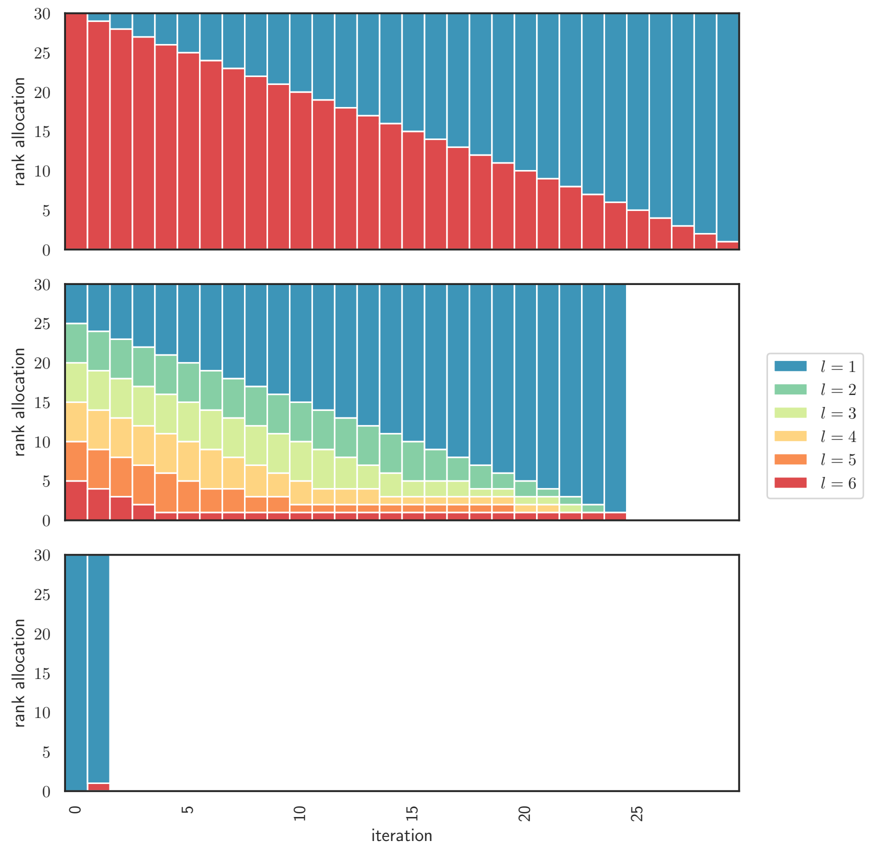

Figure 4: Rank $r = 30$ partitioning across $L = 6$ levels during fitting for asset covariance matrix with GICS hierarchy, starting from the bottom level $l = 6$, uniform and the top level $l = 1$ initial allocation.

Method	Error (%)	Storage ($\times 10^5$)
LR	16.178	1.500
LR+D	**15.379**	1.500
HODLR	34.643	1.509
Monarch	17.980	1.560
MLR bottom	**15.379**	1.500
MLR uniform	15.685	1.500
MLR top	**15.379**	1.500

Table 2: Fitting errors for asset covariance matrix with full hierarchy.

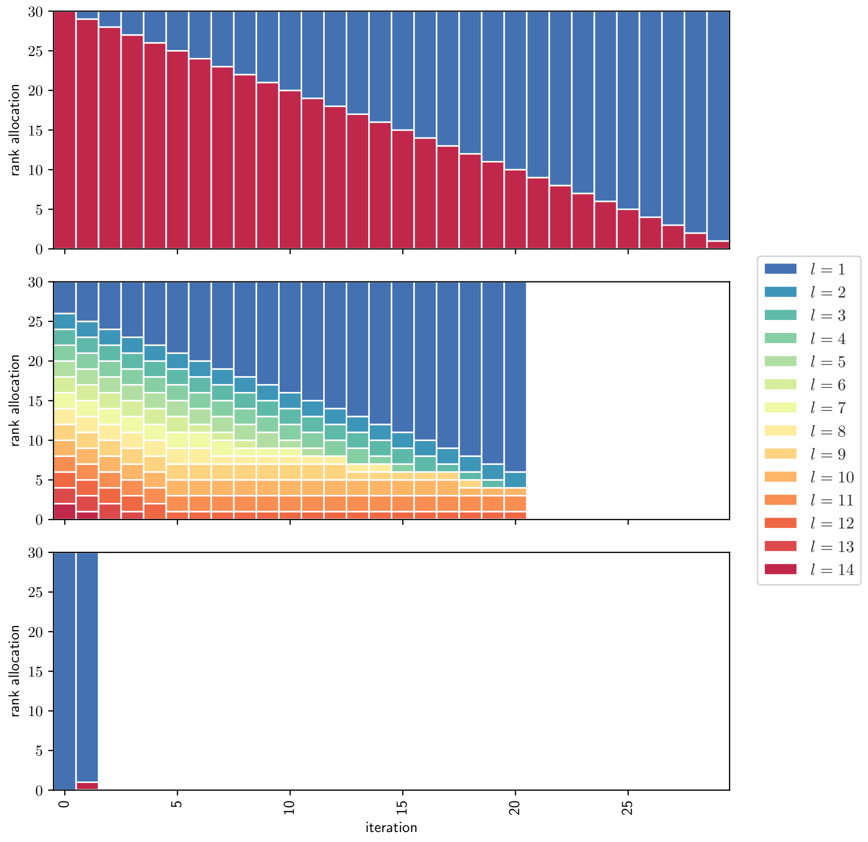

Figure 5: Rank $r = 30$ partitioning across $L = 14$ levels during fitting for asset covariance matrix with full hierarchy, starting from the bottom level $l = 14$, uniform ($r_1 = 4$, $r_2 = \cdots = r_{14} = 2$) and the top level $l = 1$ initial allocation.

Method	Error (%)	Storage ($\times 10^5$)
LR	0.1988	1.400
LR+D	0.2081	1.400
HODLR	**0.0128**	1.373
Monarch	0.1903	1.500
MLR bottom	0.0996	1.400
MLR uniform	**0.0128**	1.400
MLR top	0.0548	1.400

Table 3: Fitting errors for Fielder matrix.

8.2 Fiedler matrix

Here we consider a Fiedler matrix, with entries $A_{ij} = |a_i - a_j|$ for some vector $a \in \mathbf{R}^n$. We set $m = n = 5000$ and sample the entries of a uniformly from $[0, 1]$. We fix the rank as $r = 28$, and apply our fitting method to find symmetric (but not PSD) MLR matrix approximations.

Results. The algorithm converges to three different MLR matrices, all better than a low rank approximation. The MLR matrix found when initialized with a uniform allocation has objective that is almost $20\times$ smaller than the low rank model; see table 3 and figure 6. Its rank allocation is

$$r_1 = r_2 = r_3 = 5, \quad r_4 = 4, \quad r_5 = 3,$$
$$r_6 = r_7 = 2, \quad r_8 = r_9 = 1, \quad r_{10} = \cdots = r_{14} = 0.$$

HODLR achieves the same fitting error as MLR with the uniform initialization while having smaller storage. The error achieved by the Monarch matrix is lower than the LR by 4%.

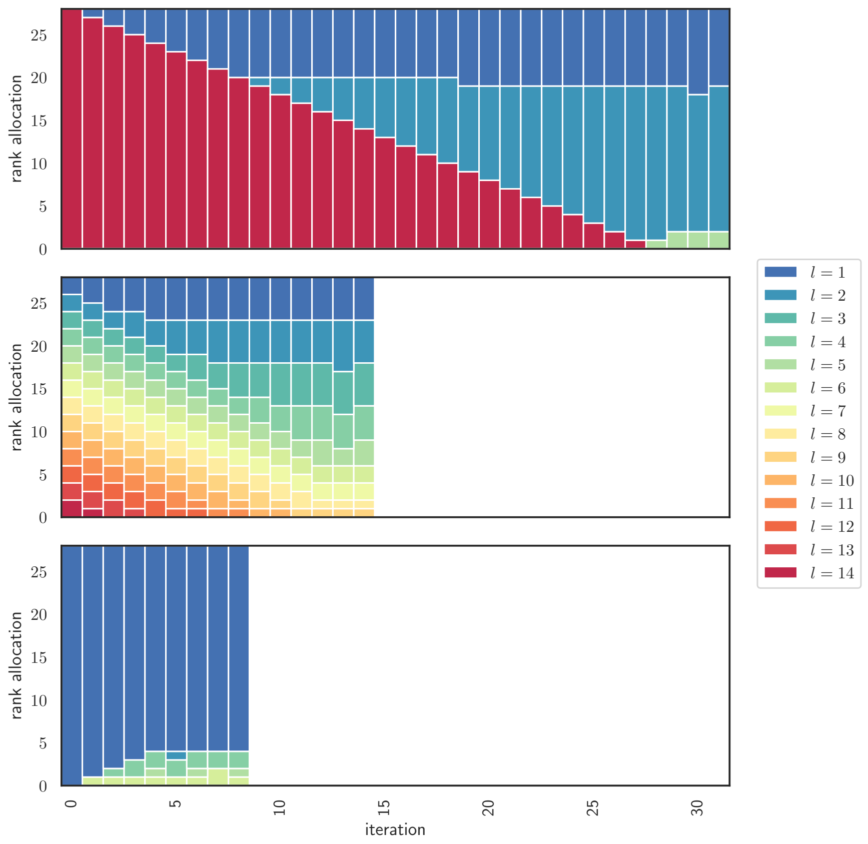

Figure 6: Rank $r = 28$ partitioning across $L = 14$ levels during fitting of the Fiedler matrix, starting from the bottom level $l = 14$, uniform and the top level $l = 1$ initial allocation.

Method	Error (%)	Storage ($\times 10^5$)
LR	41.779	3.360
HODLR	72.549	3.385
Monarch	43.962	3.600
MLR bottom	**16.753**	3.360
MLR uniform	21.766	3.360
MLR top	25.759	3.360

Table 4: Fitting errors for DGT matrix.

8.3 Discrete Gauss transform matrix

The discrete Gauss transform (DGT) matrix [GS91] is given by

$$A_{ij} = e^{-\|t_i - s_j\|_2^2/h^2},$$

where $s_j \in \mathbf{R}^d$ for $j = 1, \dots, n$ and $t_i \in \mathbf{R}^d$ for $i = 1, \dots, m$ are source and target locations respectively and $h > 0$ is the bandwidth. Similarly to the experimental setup in [YDGD03], we set sources and targets to be uniformly distributed in a unit cube $[0, 1]^d$ with $d = 3$, $m = 5000$, $n = 7000$, and $h = 0.2$. We fix the rank as $r = 28$.

Results. Our algorithm converges to three different MLR matrices, each better than the low rank or low rank plus diagonal approximations or HODLR or Monarch; see table 4 and figure 7. The lowest approximation error is achieved with the bottom rank initialization, and achieves error more than a factor 2 smaller than the low rank model. It has rank allocation

$$r_1 = 12, \quad r_2 = 6, \quad r_3 = r_4 = r_5 = 3, \quad r_6 = 0, \quad r_7 = 1, \quad r_8 = \cdots = r_{14} = 0.$$

The HODLR and Monarch objectives surpass that of the MLR by more than a factor of 4 and 2 respectively.

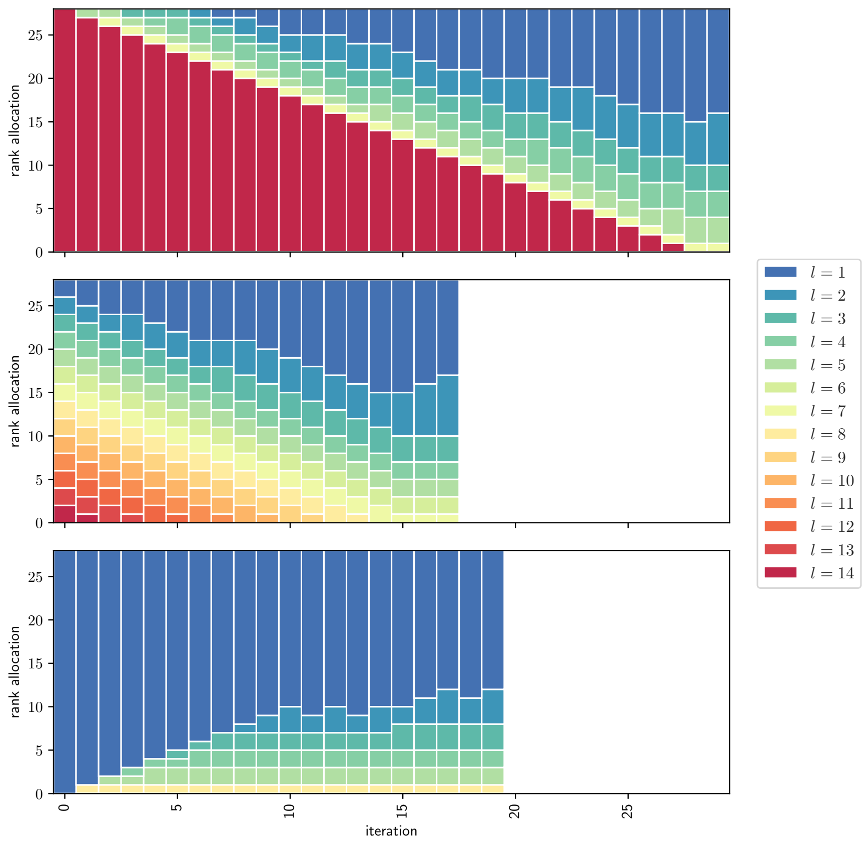

Figure 7: Rank $r = 28$ partitioning across $L = 14$ levels during fitting of DGT matrix, starting from the bottom level $l = 14$, uniform and the top level $l = 1$ initial allocation.

Method	Error (%)	Storage ($\times 10^5$)
LR	0.7197	5.775
LR+D	0.7056	5.775
HODLR	2.4696	5.789
Monarch	0.8692	5.880
MLR bottom	0.6644	5.775
MLR uniform	**0.3702**	5.775
MLR top	0.3929	5.775

Table 5: Fitting errors for distance matrix.

8.4 Distance matrix

Consider a connected weighted graph $G = (V, E, w)$ with $V = \{1, 2, \dots, n\}$ nodes and $E = \{1, 2, \dots, d\}$ edges. Let $D \in \mathbf{R}_+^{n \times n}$ denote the shortest path distance matrix, *i.e.*, D_{ij} is a shortest path distance from node j to node i. We will approximate D by a symmetric MLR matrix.

We consider the drivable street network of Venice, Italy. It contains $n = 5893$ nodes and 12026 edges. We use the OSMnx Python package [Boe17] to access the data in OpenStreetMap [Ope17]. This graph is directed; to obtain an undirected graph, we ignore the direction and take the weight to be the average of the forward and reverse directions. This results in a connected graph with 12098 nonzero entries in the adjacency matrix. We use our MLR fitting method to obtain MLR matrices with $L = 14$ levels, with a total rank to be allocated of $r = 98$.

Results. Table 5 shows the error obtained from three initial rank allocations. All are better than the low rank approximation, with the one obtained from a uniform allocation obtaining an error about $2\times$ smaller. The lowest fitting error is attained with uniform allocation, where

$$r_1 = 44, \quad r_2 = 13, \quad r_3 = 9, \quad r_4 = r_5 = r_6 = 7,$$
$$r_7 = 5, \quad r_8 = 3, \quad r_9 = 2, r_{10} = 1, \quad r_{11} = \cdots = r_{14} = 0.$$

The rank allocations versus iterations are shown in figure 8. The fitting error of MLR is more than $6\times$ less than the HODLR approximation error and $2\times$ less than the Monarch approximation error.

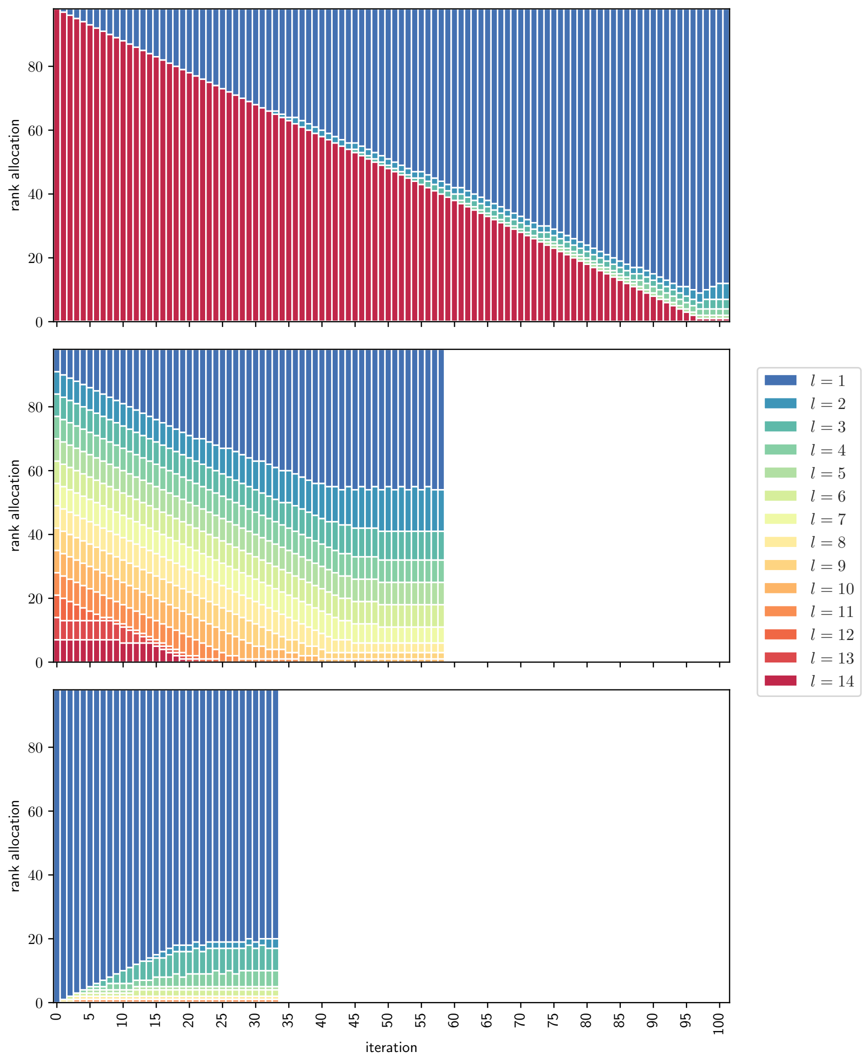

Figure 8: Rank $r = 98$ partitioning across $L = 14$ levels during fitting of the distance matrix, starting from the bottom level $l = 14$, uniform and the top level $l = 1$ initial allocation.

Method	Error (%)	Storage ($\times 10^5$)
LR	22.767	2.800
LR+D	23.282	2.800
HODLR	40.657	2.813
Monarch	25.682	2.800
MLR bottom	**6.497**	2.800
MLR uniform	7.733	2.800
MLR top	9.400	2.800

Table 6: Fitting errors for multiscale inverse polynomial kernel matrix.

8.5 Multiscale inverse polynomial kernel matrix

The multiscale inverse polynomial kernel matrix is given by

$$A_{ij} = \sum_{l=0}^{L_A-1} \left(1 + \left(\frac{\|t_i - s_j\|_2}{\sigma/2^l}\right)^2\right)^{-2},$$

where $s_j \in \mathbf{R}^d$ for $j = 1, \dots, n$ and $t_i \in \mathbf{R}^d$ for $i = 1, \dots, m$ are source and target locations respectively. To induce different scaling, we decrease the kernel radius $\sigma/2^l$ with every level increment $l = 0, \dots, L_A - 1$. We set sources and targets to be uniformly distributed on a unit sphere in $\mathbf{R}^d$ with $d = 3$, $m = n = 5000$, $L_A = 3$, and $\sigma = 0.9$. We derive the hierarchical partitioning based on the recursive spectral bi-clustering of source and target points w.r.t. the Euclidean distances $\|t_i - s_j\|_2$, and use it for all MLR, HODLR and Monarch. We fix the rank as $r = 28$.

Results. The algorithm converges to three different MLR matrices, each better than the low rank or HODLR approximations; see table 6 and figure 9. The lowest approximation error is achieved with the bottom rank initialization, with error more than a factor 3 smaller than the low rank model. It has rank allocation

$$r_1 = 11, \quad r_2 = 5, \quad r_3 = 3, \quad r_4 = 5,$$
$$r_5 = 0, \quad r_6 = r_7 = 2, \quad r_8 = \cdots = r_{14} = 0.$$

The approximation errors of the HODLR and Monarch are respectively $6\times$ and $4\times$ greater than the fitting error produced by the MLR.

Acknowledgments

The authors thank Parth Nobel, Mykel Kochenderfer, Emmanuel Candes, and Dylan Rueter for useful discussions and helpful suggestions.

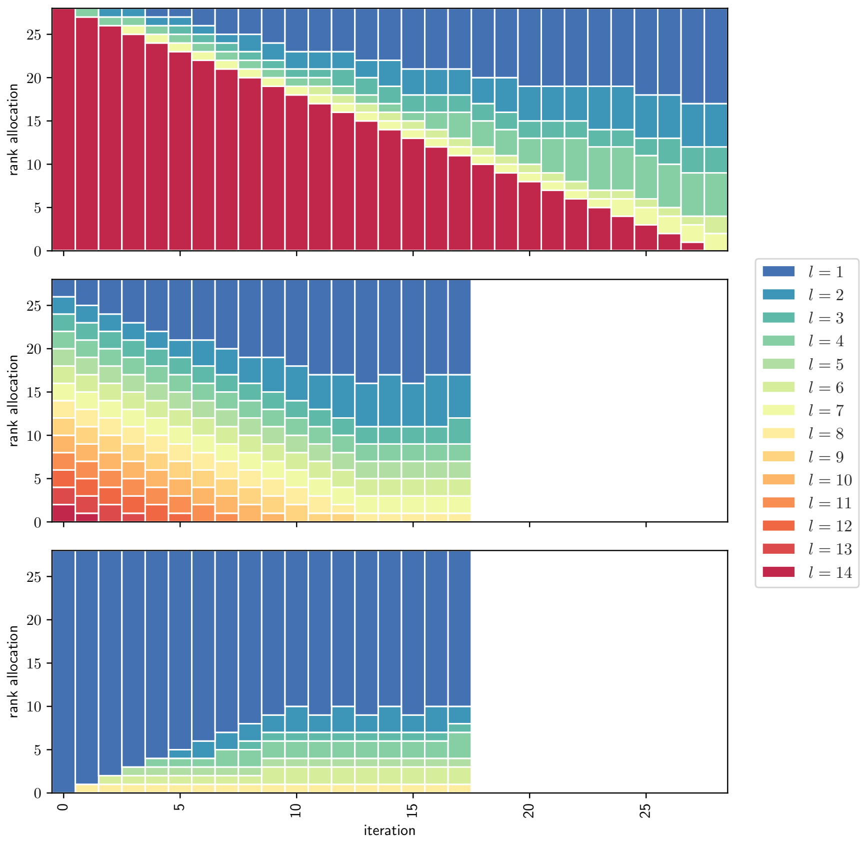

Figure 9: Rank $r = 28$ partitioning across $L = 14$ levels during the fitting of multiscale inverse polynomial kernel matrix, starting from the bottom level $l = 14$, uniform and the top level $l = 1$ initial allocation.

References

[AAB$^+$15] 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.

[AAD16] Amirhossein Aminfar, Sivaram Ambikasaran, and Eric Darve. A fast block low-rank dense solver with applications to finite-element matrices. *Journal of Computational Physics*, 304:170–188, 2016.

[ABLM19] Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'excellent, and Theo Mary. Performance and scalability of the block low-rank multifrontal factorization on multicore architectures. *ACM Transactions on Mathematical Software (TOMS)*, 45(1):1–26, 2019.

[Ach03] Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003. Special Issue on PODS 2001.

[AD13] Sivaram Ambikasaran and Eric Darve. An $\mathcal{O}(n \log n)$ fast direct solver for partial hierarchically semi-separable matrices: With application to radial basis function interpolation. *Journal of Scientific Computing*, 57:477–501, 2013.

[ADLK01] Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. MUMPS: A general purpose distributed memory sparse solver. In *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia: 5th International Workshop*, pages 121–130. Springer, 2001.

[AHAA$^+$20] Noha Al-Harthi, Rabab Alomairy, Kadir Akbudak, Rui Chen, et al. Solving acoustic boundary integral equations using high performance tile low-rank LU factorization. In *High Performance Computing: 35th International Conference, ISC High Performance*, pages 209–229. Springer, 2020.

[ALMK17] Kadir Akbudak, Hatem Ltaief, Aleksandr Mikhalev, and David Keyes. Tile low rank Cholesky factorization for climate/weather modeling applications on manycore architectures. In *High Performance Computing: 32nd International Conference*, pages 22–40. Springer, 2017.

[Ave19] Marco Avellaneda. Hierarchical PCA and applications to portfolio management. *Capital Markets: Asset Pricing & Valuation eJournal*, 2019.

[Beb08] Mario Bebendorf. *Hierarchical matrices: A means to efficiently solve elliptic boundary value problems*. Springer, 1st edition, 2008.

[Beb11] Mario Bebendorf. Adaptive cross approximation of multivariate functions. *Constructive Approximation*, 34(2):149–179, 2011.

[BGH03] Steffen Börm, Lars Grasedyck, and Wolfgang Hackbusch. Introduction to hierarchical matrices with applications. *Engineering Analysis with Boundary Elements*, 27(5):405–422, 2003.

[BLO03] Sanjeev Bhojraj, Charles M. C. Lee, and Derek K. Oler. What's my line? A comparison of industry classification schemes for capital market research. *Journal of Accounting Research*, 41(5):745–774, 2003.

[Boe17] Geoff Boeing. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

[Bon99] Marc Bonnet. Boundary integral equation methods for solids and fluids. *Meccanica*, 34:301–302, 1999.

[BS94] Stephen T. Barnard and Horst D. Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, 1994.

[BS$^+$16] Marie Brière, Ariane Szafarz, et al. Factor-based vs. industry-based asset allocation: The contest. Technical report, Amundi Working Paper, 2016.

[Buh03] Martin D. Buhmann. *Radial basis functions: Theory and implementations*, volume 12. Cambridge University Press, 2003.

[CC00] Yizong Cheng and George M. Church. Biclustering of expression data. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, page 93–103. AAAI Press, 2000.

[CDG$^+$07] Shiv Chandrasekaran, Patrick Dewilde, Ming Gu, William Lyons, and Timothy Pals. A fast solver for HSS representations via sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 29(1):67–81, 2007.

[CDHR08] Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software*, 35(3), 2008.

[CGP06] 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.

[Cha87] Tony F. Chan. Rank revealing QR factorizations. *Linear Algebra and its Applications*, 88-89:67–82, 1987.

[Cip00] Barry A. Cipra. The best of the 20th century: Editors name top 10 algorithms. *SIAM News*, 33(4):1–2, 2000.

[CKKK18] In Choi, Dukpa Kim, Yun Jung Kim, and Noh-Sun Kwark. A multilevel factor model: Identification, asymptotic theory and applications. *Journal of Applied Econometrics*, 33(3):355–377, 2018.

[Cou13] Kevin B. Coughlin. *An analysis of factor extraction strategies: A comparison of the relative strengths of principal axis, ordinary least squares, and maximum likelihood in research contexts that include both categorical and continuous variables.* USF Tampa Graduate Theses and Dissertations, 2013.

[CPA$^+$20] Qinglei Cao, Yu Pei, Kadir Akbudak, Aleksandr Mikhalev, et al. Extreme-scale task-based Cholesky factorization toward climate and weather prediction applications. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, pages 1–11, 2020.

[CPD17] Pieter Coulier, Hadi Pouransari, and Eric Darve. The inverse fast multipole method: Using a fast approximate direct solver as a preconditioner for dense linear systems. *SIAM Journal on Scientific Computing*, 39(3):A761–A796, 2017.

[CT65] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[Dar00] Eric Darve. The fast multipole method: Numerical implementation. *Journal of Computational Physics*, 160(1):195–240, 2000.

[Dat10] Biswa N. Datta. *Numerical linear algebra and applications*, volume 116. Society for Industrial and Applied Mathematics, 2010.

[DCL$^+$21] Tri Dao, Beidi Chen, Kaizhao Liang, Jiaming Yang, Zhao Song, Atri Rudra, and Christopher Re. Pixelated butterfly: Simple and efficient sparse training for neural network models. *arXiv preprint arXiv:2112.00029*, 2021.

[DCS$^+$22] Tri Dao, Beidi Chen, Nimit S. Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pages 4690–4721. PMLR, 2022.

[DGE$^+$19] Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. In *International Conference on Machine Learning*, pages 1517–1527. PMLR, 2019.

[DH73] William E. Donath and Alan J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5):420–425, 1973.

[Dhi01] Inderjit S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the Seventh ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, page 269–274. Association for Computing Machinery, 2001.

[DRSL16] Timothy A. Davis, Sivasankaran Rajamanickam, and Wissam M. Sid-Lakhdar. A survey of direct methods for sparse linear systems. *Acta Numerica*, 25:383–566, 2016.

[DSCP$^+$18] Christopher De Sa, Albert Cu, Rohan Puttagunta, Christopher Ré, and Atri Rudra. A two-pronged progress in structured dense matrix vector multiplication. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1060–1079. SIAM, 2018.

[DSG$^+$20] Tri Dao, Nimit S. Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. *arXiv preprint arXiv:2012.14966*, 2020.

[DV06] Amit Deshpande and Santosh Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 292–303. Springer, 2006.

[DW21] Eric Darve and Mary Wootters. *Numerical linear algebra with Julia*, volume 172. SIAM, 2021.

[EY36] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[Fie73] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(2):298–305, 1973.

[GB14] Harvey Goldstein and William Browne. Multilevel factor analysis modelling using Markov chain Monte Carlo estimation. In *Latent variable and latent structure models*, pages 237–256. Psychology Press, 2014.

[GH03] Lars Grasedyck and Wolfgang Hackbusch. Construction and arithmetics of $\mathcal{H}$-matrices. *Computing*, 70:295–334, 2003.

[GHK08] Lars Grasedyck, Wolfgang Hackbusch, and Ronald Kriemann. Performance of $\mathcal{H}$-LU preconditioning for sparse matrices. *Computational Methods in Applied Mathematics*, 8(4):336–349, 2008.

[GKK85] Israel Gohberg, Thomas Kailath, and Israel Koltracht. Linear complexity algorithms for semiseparable matrices. *Integral Equations and Operator Theory*, 8(6):780–804, 1985.

[GR87] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348, 1987.

[GS91] Leslie Greengard and John Strain. The fast Gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.

[GVL96] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins University Press, USA, 1996.

[GYM12] Adrianna Gillman, Patrick M. Young, and Gunnar Martinsson. A direct solver with $\mathcal{O}(n)$ complexity for integral equations on one-dimensional domains. *Frontiers of Mathematics in China*, 7:217–247, 2012.

[Hac99] Wolfgang Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

[Hac15] Wolfgang Hackbusch. *Hierarchical matrices: Algorithms and analysis*, volume 49. Springer, 2015.

[Hal70] Kenneth M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17(3):219–229, 1970.

[Har67] Harry H. Harman. *Modern factor analysis*. University of Chicago Press, 1967.

[Har72] John A. Hartigan. Direct clustering of a data matrix. *Journal of the American Statistical Association*, 67(337):123–129, 1972.

[HB02] Wolfgang Hackbusch and Steffen Börm. Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. *Computing*, 69(1):1–35, sep 2002.

[HRR02] Pascal Hénon, Pierre Ramet, and Jean Roman. Pastix: A high-performance parallel direct solver for sparse symmetric positive definite systems. *Parallel Computing*, 28(2):301–321, 2002.

[HS$^+$52] Magnus R. Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

[Hsi06] George C. Hsiao. Boundary element methods — An overview. *Applied Numerical Mathematics*, 56(10-11):1356–1369, 2006.

[JC16] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: A review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[Kal63] Rudolf E. Kalman. Mathematical description of linear dynamical systems. *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, 1(2):152–192, 1963.

[KBR11] Wai Y. Kong, James Bremer, and Vladimir Rokhlin. An adaptive fast direct solver for boundary integral equations in two dimensions. *Applied and Computational Harmonic Analysis*, 31(3):346–369, 2011.

[KCH$^+$22] Bazyli Klockiewicz, Léopold Cambier, Ryan Humble, Hamdi Tchelepi, and Eric Darve. Second-order accurate hierarchical approximate factorizations for solving sparse linear systems. *International Journal for Numerical Methods in Engineering*, 123(22):5473–5499, 2022.

[KD20] Bazyli Klockiewicz and Eric Darve. Sparse hierarchical preconditioners using piecewise smooth approximations of eigenvectors. *SIAM Journal on Scientific Computing*, 42(6):A3907–A3931, 2020.

[LSY98] Richard B. Lehoucq, Danny C. Sorensen, and Chao Yang. *ARPACK users' guide: Solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. SIAM, 1998.

[Mar09] Per-Gunnar Martinsson. A fast direct solver for a class of elliptic partial differential equations. *Journal of Scientific Computing*, 38(3):316–330, 2009.

[Mar17] Théo Mary. *Block low-rank multifrontal solvers: Complexity, performance, and scalability*. PhD thesis, Université Paul Sabatier-Toulouse III, 2017.

[MNP13] Emanuel Moench, Serena Ng, and Simon Potter. Dynamic hierarchical factor models. *Review of Economics and Statistics*, 95(5):1811–1817, 2013.

[MRK22] Stefano Massei, Leonardo Robol, and Daniel Kressner. Hierarchical adaptive low-rank format with applications to discretized partial differential equations. *Numerical Linear Algebra with Applications*, 29(6):e2448, 2022.

[Nis02] Naoshi Nishimura. Fast multipole accelerated boundary integral equation methods. *Applied Mechanics Reviews*, 55(4):299–324, 2002.

[NS17] Kishore K. Naraparaju and Jan Schneider. Literature survey on low rank approximation of matrices. *Linear and Multilinear Algebra*, 65(11):2212–2244, 2017.

[Ope17] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org, 2017.

[Par95] Douglass S. Parker. *Random butterfly transformations with applications in computational linear algebra*. UCLA Computer Science Department, 1995.

[PCD17] Hadi Pouransari, Pieter Coulier, and Eric Darve. Fast hierarchical solvers for sparse matrices using extended sparsification and low-rank approximation. *SIAM Journal on Scientific Computing*, 39(3):A797–A830, 2017.

[PDF$^+$18] 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.

[Pea01] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[PGM$^+$19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

[PS75] Christopher C. Paige and Michael A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, 1975.

[PSL90] Alex Pothen, Horst D. Simon, and Kang-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Applications*, 11(3):430–452, 1990.

[Rok85] Vladimir Rokhlin. Rapid solution of integral equations of classical potential theory. *Journal of Computational Physics*, 60(2):187–207, 1985.

[SA20] Juan A. Serur and Marco Avellaneda. Hierarchical PCA and modeling asset correlations. *Econometric Modeling: International Financial Markets - Emerging Markets eJournal*, 2020.

[SBA17] Thomas Sterling, Maciej Brodowicz, and Matthew Anderson. *High performance computing: Modern systems and practices*. Morgan Kaufmann, 2017.

[Sch07] Erhard Schmidt. Zur theorie der linearen und nichtlinearen integralgleichungen. i. teil: Entwicklung willkürlicher funktionen nach systemen vorgeschriebener. *Mathematische Annalen*, 63:433–476, 1907.

[Sim91] Horst D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(2-3):135–148, 1991.

[Sor92] Danny C. Sorensen. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM Journal on Matrix Analysis and Applications*, 13(1):357–385, 1992.

[Spe04] Charles Spearman. "General intelligence," objectively determined and measured. *The American Journal of Psychology*, 15(2):201–292, 1904.

[SS86] Youcef Saad and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[Ste98] Gilbert W. Stewart. *Matrix algorithms.* Society for Industrial and Applied Mathematics, 1998.

[SWDJ03] Age K. Smilde, Johan A. Westerhuis, and Sijmen De Jong. A framework for sequential multiblock component methods. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 17(6):323–337, 2003.

[SY14] Phillip G. Schmitz and Lexing Ying. A fast nested dissection solver for Cartesian 3d elliptic problems using hierarchical matrices. *Journal of Computational Physics*, 258:227–245, 2014.

[Tyr96] Eugene Tyrtyshnikov. Mosaic-skeleton approximations. *Calcolo*, 33(1):47–57, 1996.

[UHZ$^+$16] Madeleine Udell, Corinne Horn, Reza Zadeh, Stephen Boyd, et al. Generalized low rank models. *Foundations and Trends® in Machine Learning*, 9(1):1–118, 2016.

[VGO$^+$20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, et al. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17:261–272, 2020.

[vL07] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[VM03] Deepak Verma and Marina Meila. A comparison of spectral clustering algorithms. *University of Washington Tech Rep UWCSE030501*, 1:1–18, 2003.

[Wat15] Andrew J. Wathen. Preconditioning. *Acta Numerica*, 24:329–376, 2015.

[Wei13] Clément Weisbecker. *Improving multifrontal solvers by means of algebraic block low-rank representations.* PhD thesis, Institut National Polytechnique de Toulouse-INPT, 2013.

[WKT96] Svante Wold, Nouna Kettaneh, and Kjell Tjessem. Hierarchical multiblock PLS and PC models for easier model interpretation and as an alternative to variable selection. *Journal of Chemometrics*, 10(5-6):463–482, 1996.

[XCGL10a] Jianlin Xia, Shivkumar Chandrasekaran, Ming Gu, and Xiaoye S. Li. Fast algorithms for hierarchically semiseparable matrices. *Numerical Linear Algebra with Applications*, 17(6):953–976, 2010.

[XCGL10b] 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.

[Xia13] Jianlin Xia. Randomized sparse direct solvers. *SIAM Journal on Matrix Analysis and Applications*, 34(1):197–227, 2013.

[XT15] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

[YBZ04] Lexing Ying, George Biros, and Denis Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics*, 196(2):591–626, 2004.

[YDGD03] Changjiang Yang, Ramani Duraiswami, Nail A. Gumerov, and Larry Davis. Improved fast Gauss transform and efficient kernel density estimation. In *Proceedings of the Ninth IEEE International Conference on Computer Vision*, volume 2. IEEE Computer Society, 2003.

[YTM99] Yiu-Fai Yung, David Thissen, and Lori D. McLeod. On the relationship between the higher-order factor model and the hierarchical factor model. *Psychometrika*, 64:113–128, 1999.

[ZHD$^+$01] Hongyuan Zha, Xiaofeng He, Chris Ding, Horst Simon, and Ming Gu. Bipartite graph partitioning and data clustering. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, pages 25–32. Association for Computing Machinery, 2001.