SIMPLE NON-EXTENSIVE SPARSIFICATION OF THE HIERARCHICAL MATRICES[§]

DARIA A. SUSHNIKOVA[¶] AND IVAN V. OSELEDETS^{‡¶}

Abstract.

In this paper, we consider the matrices approximated in \mathcal{H}^2 format. The direct solution, as well as the preconditioning, of systems with such matrices is a challenging problem. We propose a non-extensive sparse factorization of the \mathcal{H}^2 matrix that allows to substitute direct \mathcal{H}^2 solution with the solution of the system with an equivalent sparse matrix of the same size. The sparse factorization is constructed of parameters of the \mathcal{H}^2 matrix. In the numerical experiments, we show the consistency of this approach in comparison to the other approximate block low-rank hierarchical solvers, such as HODLR[3], H2Lib[5], and IFMM[11].

Key words. \mathscr{H}^2 matrix, sparse factorization, preconditioning

1. Introduction. Problems arising in the discretization of boundary integral equations (and a number of other problems with approximately separable kernels) lead to matrices that can be well-approximated by hierarchical block low-rank (\mathscr{H} [15, 17], mosaic skeleton[27]) matrices. These are the matrices hierarchically divided into blocks, some of which has low-rank. The development of the \mathscr{H} matrices is the \mathscr{H}^2 [16, 6] matrices, which are the hierarchical block low-rank matrices with nested bases. The nested basis property leads to the additional improvement in terms of storage and complexity of different operations such as matrix-vector products. Approximate solution and preconditioning of systems with \mathscr{H}^2 matrices is a rapidly developed area[9, 3, 11, 19], however, construction of the accurate, time and memory efficient factorization that leads to approximate solution is still a challenging problem. In this paper, we propose a new representation of \mathscr{H}^2 matrices. Namely, we show that \mathscr{H}^2 factorization of matrix $A \in \mathbb{R}^{N \times N}$ is equivalent to the factorization

$$A = USV^{\top}, \tag{1.1}$$

where $S \in \mathbb{R}^{N \times N}$ is a sparse matrix. Note that the size of matrix *S* matches the size of matrix *A*. $U \in \mathbb{R}^{N \times N}$ and $V \in \mathbb{R}^{N \times N}$ are orthogonal matrices that are products of block-diagonal and permutation matrices. Once the factorization (1.1) is built, we can substitute a solution of the system

$$Ax = b$$

by a solution of the system with the sparse matrix:

$$Sy = U^{\top}b, \tag{1.2}$$

where x = Vy. The system (1.2) can be easily solved using standard sparse tools. In this paper we propose:

• Sparse **non-extensive**¹ factorization for \mathcal{H}^2 matrix that leads to the solver and the preconditioner.

[‡]Skolkovo Institute of Science and Technology, Nobel St. 3, Skolkovo Innovation Center, Moscow, 143025 Moscow Region, Russia (i.oseledets@skolkovotech.ru)

[¶]Institute of Numerical Mathematics Russian Academy of Sciences, Gubkina St. 8, 119333 Moscow, Russia

[§]Section 2 is supported by Russian Foundation for Basic Research grant 17-01-00854, Sections 3, 4 are supported by Russian Foundation for Basic Research grant 16-31-60095, Sectiom 5 is supported by Russian Science Foundation grant 15-11-00033.

¹The term **non-extensive** means that the sizes of the factors *S*, *U* and *V* are equal to the size of the matrix *A* (in opposition to extensive[2, 11, 26] sparse factorizations of \mathcal{H}^2 matrix).

- The algorithm that allows to construct factors U, S and V from parameters of \mathscr{H}^2 matrix.
- Numerical comparison of the proposed method with HODLR[1], IFMM[11] and H2Lib[5] packages.

The main idea of the sparsification algorithm is the compression of the low-rank blocks (the very close idea of the compression of the fill-in blocks during the block Cholesky factorization of a sparse matrix is presented in works[25, 29]). The main difference between the presented sparse factorization and the other methods of sparsification[2, 11, 26] is a size of factors. In the other methods, the hierarchical matrix is factorized into the sparse matrices of **larger sizes**. The characteristic inflating coefficient is k = 5 (it depends on the number of levels in the cluster tree[2, 26]). The matrix extension is a major drawback since it increases the complexity of matrix computations. We propose the factorization that takes \mathcal{H}^2 matrix and returns the sparse factors of the **same size**. Another drawback of the extended sparse factorizations is that resulting sparse matrix may lose a positive definiteness of the original \mathcal{H}^2 matrix.

2. Compression algorithm. Consider the dense matrix $A \in \mathbb{R}^{N \times N}$ that can be approximated in \mathscr{H}^2 format (has corresponding low-rank blocks). Formal definition of the \mathscr{H}^2 matrix is presented in Section 4.1, here we give basic facts that are used in the current section. Matrix A is a block matrix with following properties. It consists of two non-intersecting "close" and "far" matrices:

$$A = C_k + F_k, \quad k \in 0, \dots, L$$

block size of zero level is B, block size of k-th level B_k is

$$B_k=2^kB,$$

 $C_k \in \mathbb{R}^{N \times N}$ is block-sparse matrix of full-rank close blocks, and $F_k \in \mathbb{R}^{N \times N}$ is block matrix of far blocks, see Figure 2.1b. The matrix F_k has low-rank block rows and block columns. Moreover, the nested basis property holds: basis rows for block rows and columns on level l are a subset of basis rows at level (l-1). This is used in the multilevel computations, which are described in Section 2.2.



Fig. 2.1: Close and far blocks of matrix A at level l = 0

2.1. Compression at zero level. First consider the compression procedure at zero block level (l = 0). Assume that the number of block rows and columns at zero level is M. Nonzero block $F_{ij} \in \mathbb{R}^{B \times B}$ of far matrix F_0 has low rank:

$$F_{ij} \approx \widetilde{U}_i \widetilde{F}_{ij} \widetilde{V}_j^{\top}, \quad \forall i, j \in 1, \dots, M$$

where $\widetilde{F}_{ij} \in \mathbb{R}^{N \times N}$ is the compressed far block with the following structure:

$$\widetilde{F}_{ij} = \begin{bmatrix} \dot{F}_{ij} & 0\\ 0 & 0 \end{bmatrix},$$

where $\dot{F}_{ij} \in \mathbb{R}^{r \times r}$. Matrices $\widetilde{U}_i \in \mathbb{R}^{N \times N}$ and $\widetilde{V}_j \in \mathbb{R}^{N \times N}$ are orthogonal. The blocks in *i*-th row have the same left orthogonal compression factor \widetilde{U}_i and all blocks in *j*-th column have the same right factor \widetilde{V}_i^{\top} .

The goal of the compression procedure is to sparsify the matrix A by obtaining the compressed blocks \tilde{F}_{ij} instead of original blocks F_{ij} . One can achieve this by finding \tilde{U}_i and \tilde{V}_j compression matrices and applying matrix \tilde{U}_i^{\top} to *i*-th row and \tilde{V}_j to *j*-th column. We introduce the block-diagonal orthogonal compression matrix

$$U_0^{\top} = \begin{bmatrix} \tilde{U}_1^{\top} & 0 & 0\\ 0 & \ddots & 0\\ 0 & 0 & \tilde{U}_M^{\top} \end{bmatrix}.$$
 (2.1)

Similarly, for block columns we obtain the block-diagonal orthogonal compression matrix

$$V_0 = \begin{bmatrix} \widetilde{V}_1 & 0 & 0\\ 0 & \ddots & 0\\ 0 & 0 & \widetilde{V}_M \end{bmatrix}.$$
 (2.2)

Applying matrices U_0 and V_0 to the matrix A we obtain the matrix A_1 with *compressed* far matrix:

$$A_1 = U_0^\top A V_0.$$

The process is illustrated in Figure 2.2.



Fig. 2.2: Zero level compression

Finally, we obtain:

$$A_1 = U_0^\top (C_0 + F_0) V_0 = U_0^\top C_0 V_0 + \widetilde{F}_1,$$

where \widetilde{F}_1 is a compressed far matrix which consists of blocks \widetilde{F}_{ij} . Note that the matrix C_0 is available as one of the parameters of the \mathscr{H}^2 format, it is a so-called "close matrix".

2.2. Compression at the first level (l = 1). For each block row in A_1 we denote the rows with zero far blocks by "*non-basis*", and the other rows by "*first level basis*". Assume that each block row (column) has *r* basis rows (columns) and (B - r) non-basis. Introduce the permutation P_{r1} that puts non-basis block rows before the basis ones preserving the row order and permutation P_{c1} that does the same for columns, see Figure 2.4a. For the permuted matrix

$$\widetilde{A}_1 = P_{r1}A_1P_{c1}$$

we obtain

$$\widetilde{A}_1 = \begin{bmatrix} A_{\mathbf{n_1}\mathbf{n_1}} & A_{\mathbf{n_1}\mathbf{b_1}} \\ A_{\mathbf{b_1}\mathbf{n_1}} & A_{\mathbf{b_1}\mathbf{b_1}} \end{bmatrix}$$

where $A_{\mathbf{n_1n_1}} \in \mathbb{R}^{M(B-r) \times M(B-r)}$ is a submatrix on the intersection of non-basis rows and non-basis columns, $A_{\mathbf{b_1n_1}} \in \mathbb{R}^{Mr \times M(B-r)}$ is on the intersection of basis rows and non-basis columns and so on, see Figure 2.4a. Denote the permuted far matrix:

$$\widehat{F}_1 = (P_{r1}\widetilde{F}_1P_{c1}).$$

Note that permutations P_{r1} and P_{c1} concentrate all nonzero blocks of compressed far zone F_1 inside of the submatrix $A_{b_1b_1}$. Denote permuted close matrix:

$$\widehat{C}_1 = (P_{r1}U_0^\top C_0 V_0 P_{c1}).$$
(2.3)

Consider the submatrix $A_{\mathbf{b_1b_1}} \in \mathbb{R}^{Mr \times Mr}$, note that this matrix has exactly the same close and far block structure as the matrix A, but the block size in $A_{\mathbf{b_1b_1}}$ is r. Now we join block rows and columns of the matrix $A_{\mathbf{b_1b_1}}$ by groups of J blocks (e.g. J = 2 in Figure 2.3a). Assume that Jr = B.

We will call the grouped blocks "big blocks". Among these blocks, the big block that consists only of far sub-blocks will be called far, the big block that contains at least one close small block will be referred to as close. Denote blocks of the far matrix \hat{F}_1 that become close after grouping by \hat{F}_{ml1} , see Figure 2.3. We also introduce a new close matrix with big blocks by

$$C_1 = \widehat{C}_1 + \widehat{F}_{\mathbf{ml}1}.\tag{2.4}$$

Denote far matrix with big blocks by F_1 . Consider this joining for the block $A_{b_1b_1}$:

$$A_{\mathbf{b}_{1}\mathbf{b}_{1}} = (\widehat{C}_{1})_{\mathbf{b}_{1}\mathbf{b}_{1}} + \widehat{F}_{1} = (\widehat{C}_{1})_{\mathbf{b}_{1}\mathbf{b}_{1}} + \widehat{F}_{\mathbf{m}\mathbf{l}1} + F_{1} = (C_{1})_{\mathbf{b}_{1}\mathbf{b}_{1}} + F_{1}.$$



Fig. 2.3: Small (*r*-size) and big (*B*-size) far and close blocks of matrix $A_{b_1b_1}$

Similarly, for the matrix \widetilde{A}_1 :

$$\widetilde{A}_1 = \widehat{C}_1 + \widehat{F}_1 = \widehat{C}_1 + \widehat{F}_{ml1} + F_1 = C_1 + F_1.$$

It can be shown that block rows and columns of the matrix F_1 have low-rank by the properties of the \mathscr{H}^2 matrix A. Similarly to (2.1) compute orthogonal block-diagonal matrices $U_{\mathbf{b}_1}, V_{\mathbf{b}_1} \in \mathbb{R}^{Mr \times Mr}$ that compress matrix F_1 . Multiplication of matrix F_1 by matrices $U_{\mathbf{b}_1}$ and $V_{\mathbf{b}_1}$ leads to compression:

$$\widetilde{F}_2 = U_{\mathbf{b}_1}^\top F_1 V_{\mathbf{b}_1},\tag{2.5}$$

where the matrix \widetilde{F}_2 consists of compressed blocks.

Now we introduce extended matrices $U_{\mathbf{b}_1}$ and $V_{\mathbf{b}_1}$ that can be applied to matrix \widehat{A}_1 :

$$U_{1} = \begin{bmatrix} I_{(N-Mr)\times(N-Mr)} & 0\\ 0 & U_{\mathbf{b}_{1}} \end{bmatrix}, \quad V_{1} = \begin{bmatrix} I_{(N-Mr)\times(N-Mr)} & 0\\ 0 & V_{\mathbf{b}_{1}} \end{bmatrix}.$$
 (2.6)

Applying matrices U_1 and V_1 to matrix \widetilde{A}_1 we obtain the matrix with compressed first level:

$$A_2 = U_1^\top \widetilde{A}_1 V_1.$$

The process of the first level compression is shown in Figure 2.4b.



Fig. 2.4: The first level compression

For the first level we obtain

$$A_2 = U_1^{\top} (C_1 + F_1) V_1 = U_1^{\top} C_1 V_1 + \widehat{F}_2.$$

2.3. Compression at all levels. We apply permutation and repeat this procedure *L* times and obtain:

$$A_{1} = U_{0}^{\top} A V_{0} = U_{0}^{\top} C_{0} V_{0} + \widehat{F}_{1}$$

$$A_{2} = U_{1}^{\top} U_{0}^{\top} A V_{0} V_{1} = U_{1}^{\top} C_{1} V_{1} + \widehat{F}_{2}$$

$$\vdots$$

$$A_{L} = \left(\prod_{k=0}^{L} U_{k}^{\top}\right) A\left(\prod_{k=L}^{0} V_{k}\right) = U_{L-1}^{\top} C_{L-1} V_{L-1} + \widehat{F}_{L} = S,$$
(2.7)

thus

$$A = \left(\prod_{k=L}^{0} U_k\right) S\left(\prod_{k=0}^{L} V_k^{\top}\right).$$

If we denote

$$U = \prod_{k=0}^{L} U_k, \quad V = \prod_{k=0}^{L} V_k,$$
(2.8)

then the final result of the algorithm is a sparse approximate factorization

$$A = USV^{\top}, \tag{2.9}$$

where S is a sparse matrix of the same size as matrix A, U and V are orthogonal matrices that are products of permutation and block-diagonal orthogonal matrices.

REMARK 2.1. If matrix A is approximated into \mathcal{H}^2 format, then matrices S, U and V can be constructed from parameters of the \mathcal{H}^2 representation, see details in Section 4.

REMARK 2.2. Sparsity of the matrix S is proven in Section 3.

PROPOSITION 2.1. If the matrix A is symmetric and positive definite, then the factors U and V are equal and the matrix S is symmetric and positive definite.

Proof. If the matrix *A* is symmetric and positive definite, then from the steps of compression algorithm, compression matrices U_i and V_i , $i \in 0, ..., L$ are equal, thus, by equations (2.8), U = V. Since $S = U^{\top}AV$, U = V and *A* is symmetric and positive definite, then *S* is symmetric and positive definite matrix. \Box

REMARK 2.3. The proposed sparsification algorithm is applicable to the special cases of \mathscr{H}^2 matrices such as HSS (Hierarchically Semiseparable), HOLDR (Hierarchical Off-Diagonal Low-Rank).

2.4. Pseudo code of the compression algorithm.

 Algorithm 1: Compression algorithm

 Input:

 $A \in \mathbb{R}^{N \times N}$ - matrix with \mathscr{H}^2 structure

 L - number of levels

 Compression:

 for $k = 0, \dots, L$ do

 M_k - number of blocks on level M_k
 P_{rk}, P_{ck} basis-non-basis permutations

 $\widetilde{A}_k = P_{rk}A_1P_{ck}, (P_{r0} = I, P_{c0} = I)$

 for $i = 1, \dots, M_k$ do

 Compute U_{ki} (using SVD of *i*-th block row of $A_{b_k b_k}$)

 Compute V_{ki} (using SVD of *i*-th block column of $A_{b_k b_k}$)

 $U_k = P_{rk} \operatorname{diag}(U_{k1}, \dots, U_{kM_k})$
 $V_k = P_{ck} \operatorname{diag}(V_{k1}, \dots, V_{kM_k})$
 $A_{j+1} = U_j A_j V_j^{\top}$

 Output: Factorization $A \approx USV^{\top}$
 $U = (\prod_{k=0}^L U_k),$
 $V = (\prod_{k=0}^L U_k),$
 $S = A_L = (\prod_{k=0}^L U_k^{\top}) A (\prod_{k=L}^0 V_k)$

3. Sparsity of the matrix S. First, define the block sparsity pattern of a block sparse matrix. For a matrix A with M_1 block columns, M_2 block rows and block size B define $bsp(A)_{B\times B}^{M_1\times M_2}$ (block sparsity pattern) as a function

$$\mathbf{bsp}: \mathbb{R}^{M_1B \times M_2B} \to \mathbb{B}^{M_1 \times M_2},$$

where $\mathbb{B} = \{0,1\}$. The function takes block matrix $A \in \mathbb{R}^{M_1B \times M_2B}$ as input and returns as output the matrix $R = \mathbf{bsp}(A)_{B \times B}^{M_1 \times M_2} \in \mathbb{B}^{M_1 \times M_2}$ such that

$$\begin{cases} R_{ij} = 1, & \text{if } A_{ij} \in \mathbb{R}^{B \times B} & \text{is nonzero block,} \\ R_{ij} = 0, & \text{if } A_{ij} \in \mathbb{R}^{B \times B} & \text{is zero block.} \end{cases}$$

By $\#bsp(A)_{B\times B}^{M_1\times M_2}$ define the number of nonzero blocks of matrix A and the number of ones in matrix R.

PROPOSITION 3.1. If the matrix A has \mathscr{H}^2 structure, the compression Algorithm 1 has L levels, the block size on each level is B, the matrix A has zero level close matrix C, and the far blocks are compressed with rank r = B/2, then the compression algorithm for the matrix A leads to the factorization:

$$A = U^{\top}SV$$

where U and V are orthogonal matrices equal to the multiplication of block-diagonal compression and permutation matrices

$$U = \left(\prod_{j=0}^{K} U_j P_j^{\top}\right), \quad V = \left(\prod_{j=0}^{K} V_j P_j^{\top}\right),$$

S is a sparse matrix that has

$$\#S \leqslant \left(4L + 6\left(\frac{1}{2^L} - 1\right)\right) \#bsp(C)_{B \times B}^{M \times M}$$

nonzero blocks² of size $(r \times r)$.

Proof. Consider the matrix S from (2.9). Let matrix S_{ij} correspond to *i*-th level non-basis hyper row and *j*-th level non-basis hyper column. Thanks to basis-non-basis row and column permutations P_{ri} and P_{ci} matrix S is separated into blocks S_{ij} , where $i, j \in 0, ..., L$.

The number of nonzero blocks in S is equal to sum of nonzero blocks in S_{ij} :

$$\#S = \sum_{i=0}^{L} \sum_{j=0}^{L} \#S_{ij}.$$

Let us compute the number of nonzero blocks in block S_{ij} . Since on each level we join block rows by groups of J blocks, we obtain:

$$bsp(S_{ij})_{r\times r}^{M/2^i\times M/2^j} = bsp(C)_{2^iB\times 2^jB}^{M/2^i\times M/2^j}$$

where $i, j \in 0, \ldots, L$.

Note that

$$bsp(C)_{2^{i}B \times 2^{j}B}^{M/2^{i} \times M/2^{j}} \leqslant \frac{\#bsp(C)_{B \times B}^{M \times M}}{2^{\min(i,j)}}$$

Thus

$$\begin{split} \#S \leqslant \sum_{i=0}^{L} \sum_{j=0}^{L} \#bsp(C)_{2^{i}B \times 2^{j}B}^{M/2^{i} \times M/2^{j}} &= \sum_{i=0}^{L} \left(\frac{2(L-i)-1}{2^{i}}\right) \#bsp(C)_{B \times B}^{M \times M} = \\ &= \left(4L + 6(\frac{1}{2^{L}} - 1)\right) \#bsp(C)_{B \times B}^{M \times M}. \end{split}$$

Obtain

$$\#S \leqslant \left(4L + 6\left(\frac{1}{2^L} - 1\right)\right) \#bsp(C)_{B \times B}^{M \times M}.$$

Thus, the number of nonzero blocks in matrix S is less than the number of nonzero blocks in close matrix C multiplied by constant $(4L+6(\frac{1}{2^L}-1))$, if matrix C is block-sparse, and all proposition conditions are met, then matrix S is also sparse. \Box

4. Building sparse factorization from \mathcal{H}^2 coefficients.

4.1. Definition of \mathscr{H}^2 **matrix.** In this section we consider the matrix A approximated in \mathscr{H}^2 format. There exists a number of efficient ways to build this approximation[22, 6]. In this paper, we do not consider the process of building the \mathscr{H}^2 matrix and assume that it is given. Let us explain in details how to construct factors in the decomposition (2.9) from parameters of \mathscr{H}^2 matrix A. First, we give the definition of \mathscr{H}^2 matrix, the more detailed definition can be found in[6].

DEFINITION 4.1 (Row and column cluster trees). Cluster trees of rows and columns \mathcal{T}_r and \mathcal{T}_c define the hierarchical division of block rows and columns. At each level of the row

²The symbol # before the matrix means the number of nonzero $(r \times r)$ blocks in this matrix.

cluster tree \mathcal{T}_r , each node corresponds to a block row of the matrix A, child nodes correspond to the subrows of this row. Same for the column cluster tree \mathcal{T}_c .

DEFINITION 4.2 (Block cluster tree). Let \mathcal{T}_{rc} be a tree. \mathcal{T}_{rc} is a block cluster tree for \mathcal{T}_r and \mathcal{T}_c if it satisfies the following conditions:

- $root(\mathscr{T}_{rc}) = (root(\mathscr{T}_r), root(\mathscr{T}_c)).$
- *Each node* $b \in \mathscr{T}_{rc}$ *has the form* b = (t,s) *for* $t \in \mathscr{T}_r$ *and* $s \in \mathscr{T}_c$.
- Let $b = (t,s) \in \mathscr{T}_{rc}$. If $sons(b) \neq \emptyset$, then

$$sons(b) = \begin{cases} \{t\} \times sons(s) & \text{if } sons(t) \neq \emptyset, sons(s) \neq \emptyset, \\ sons(t) \times \{s\} & \text{if } sons(t) \neq \emptyset, sons(s) \neq \emptyset, \\ sons(t) \times sons(s) & \text{otherwise.} \end{cases}$$

DEFINITION 4.3 (Admissibility condition). Let \mathcal{T}_r and \mathcal{T}_r be a row and column cluster trees. A predicate

$$\mathscr{A} = \mathscr{T}_r \times \mathscr{T}_c \to \{ \mathit{True}, \mathit{False} \}$$

is an admissibility condition for \mathcal{T}_r and \mathcal{T}_r if

$$\mathscr{A}(t,s) \Longrightarrow \mathscr{A}(t',s) \quad holds for all \quad t \in \mathscr{T}_r, s \in \mathscr{T}_c, t' \in sons(t)$$

and

$$\mathscr{A}(t,s) \Longrightarrow \mathscr{A}(t,s')$$
 holds for all $t \in \mathscr{T}_r, s \in \mathscr{T}_c, s' \in sons(s)$.

If $\mathscr{A}(t,s)$, the pair (t,s) is called admissible.

DEFINITION 4.4 (Admissibility block cluster tree). Let \mathscr{T}_{rc} be a block cluster tree for \mathscr{T}_r and \mathscr{T}_c , let \mathscr{A} be an admissibility condition. If for each $(t,s) \in \mathscr{T}_{rc}$ either sons $(t) \neq \varnothing \neq$ sons(s) or $\mathscr{A}(t,s) =$ True holds, the block cluster tree \mathscr{T}_{rc} called \mathscr{A} -Admissible.

DEFINITION 4.5 (Farfield and nearfield). Let \mathcal{T}_{rc} be a block cluster tree for \mathcal{T}_r and \mathcal{T}_c , let \mathscr{A} be an admissibility condition. The index set

$$\mathscr{I}_{N\times N}^{+} := \{(t,s) \in \mathscr{I}_{N\times N} \quad : \mathscr{A}(t,s) = True\}$$

is called set of farfield blocks. The index set

$$\mathscr{I}_{N\times N}^{-} := \{ (t,s) \in \mathscr{I}_{N\times N} \quad : \mathscr{A}(t,s) = False \}$$

is called set of nearfield blocks.

DEFINITION 4.6 (Cut-off matrices). Let \mathscr{T}_r be a row cluster tree. For all tree nodes $t \in \mathscr{T}_r$ the cut-off matrix $\chi_t \in \mathbb{R}^{N \times N}$ corresponding to t is defined by

$$(\chi_t)_{ij} = \begin{cases} 1 & \text{if } i = j \in t, \\ 0 & \text{otherwise,} \end{cases} \quad \text{for all } i, j \in N.$$

DEFINITION 4.7 (Cluster basis). Let $K = (K_t)_{t \in \mathcal{T}_r}$ be a family of finite index sets (rank distribution for \mathcal{T}_r). Let $R = (R_t)_{t \in \mathcal{T}_r}$ be a family of matrices satisfying $R_t \in \mathbb{R}_i^{N \times K_t}$ for all $t \in \mathcal{T}_r$. Then R is called row cluster basis and the matrices R_t are called row cluster basis matrices. Analogically for column cluster basis E and column cluster basis matrices E_s , $s \in \mathcal{T}_c$.

DEFINITION 4.8 (Close matrix). The matrix $C \in \mathbb{R}^{N \times N}$ is close if

$$C = \sum_{\substack{b = (s,t) \in \mathscr{I}_{N \times N}^{-} \\ 9}} \chi_{t} A \chi_{s}$$

DEFINITION 4.9 (\mathscr{H}^2 matrix). The matrix $A \in \mathbb{R}^{N \times N}$ is approximated in \mathscr{H}^2 format if \mathscr{T}_c and \mathscr{T}_r are block cluster trees of columns and rows of matrix A, if there exist a row cluster basis R (row transition matrices), the column cluster basis E (column transition matrices), a family $D = (D_b)_{b \in \mathscr{I}_{N \times N}^+}$ of matrices satisfying $D_b \in \mathbb{R}^{K_t \times L_s}$ for all $b = (s,t) \in \mathscr{I}_{N \times N}^+$ (interaction list), and close matrix C, and if

$$A = C + \sum_{b=(s,t)\in\mathscr{I}_{N\times N}^+} R_t D_b E_s^*.$$

4.2. Construction of matrices U and V from the coefficients of \mathcal{H}^2 matrix. Let us first construct orthogonal matrices U and V from the factorization (2.9). According to equation (2.8):

$$U=\prod_{k=0}^L P_{rk}U_k,$$

and

$$V = \prod_{k=0}^{L} P_{ck} V_k$$

Note that matrices U_k and V_k , $k \in 0, ..., L$, are very close in their meaning to cluster basis matrices R_t and E_s both are level compression matrices. The difference between these matrices is that the diagonal blocks of matrices U_k and V_k are square orthogonal blocks, and diagonal blocks of matrices R_t and E_s are rectangular non-orthogonal blocks.

Thus we can take matrices R_t and E_s , orthogonalize blocks, complete each block to square orthogonal block and obtain the matrices U_k and V_k . The algorithm that orthogonalizes blocks of matrices R_t and E_s is known as the \mathcal{H}^2 compression algorithm, it can be found in[6, 7]. Compression of blocks can be done by QR decomposition of blocks with the square Q factor. Permutations P_{rk} and P_{ck} can be constructed from the cluster trees.

4.3. Construction of the matrix *S* from the coefficients of \mathscr{H}^2 matrix. According to equations (2.7), equation (2.3) and equation (2.4):

$$S = U_{L-1}^\top C_{L-1} V_{L-1} + \widehat{F}_L =$$

$$= U_{L-1}^{\top} (\dots (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{L-1} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{ml2} (U_1^{\top} (U_1^{\top} (U_0^{\top} C_0 V_0 + \widehat{F}_{ml1}) V_1 + \widehat{F}_{ml2}) \dots) V_{L-1} + \widehat{F}_{ml2} (U_1^{\top} (U_1$$

Construction of matrices U_i and V_i is shown in the previous subsection, matrix C_0 is stored in the \mathcal{H}^2 matrix explicitly as close matrix C, matrices \hat{F}_{mli} are exactly matrices D_i from interaction list, matrix \hat{F}_L is matrix D_L . Thus, matrix S can be easily computed from the coefficients of the \mathcal{H}^2 matrix.

5. Numerical experiments. Sparsification algorithm is implemented in the Python programming language. For the \mathscr{H}^2 matrix implementation we use the h2tools[22] library. All computations are performed on MacBook Air with a 1.3GHz Intel Core i5 processor and 4 GB 1600 MHz DDR3 RAM.

First, we numerically show that the matrix S in factorization (2.9) is indeed sparse. Then we give the timing and storage requirements of the sparse factorization. Thereafter we consider the sparse factorization of the \mathcal{H}^2 matrix combined with a sparse direct solver as a

direct solver for the system with the \mathscr{H}^2 matrix and compare this approach with HODLR direct solver and \mathscr{H}^2 -LU solver from \mathscr{H} 2Lib library. Finally, we consider sparse factorization with the matrix *S* factorized by the ILUt method as a preconditioner for GMRES solver.

We want to note that the sparsification algorithm **does not worsen** the accuracy of the \mathcal{H}^2 approximation. It follows from the algorithm and it is confirmed in the experiments. Therefore, in the experiments below, we omit the accuracy of the sparse factorization and show only the \mathcal{H}^2 approximation accuracy to avoid redundancy.

5. Numerical experiments. In Section 3 we have studied the sparsity of the factor S from the factorization (2.9) analytically. Here we present the numerical illustration that matrix S is indeed sparse. Tests are performed for two \mathcal{H}^2 matrices.

EXAMPLE 5.1. \mathscr{H}^2 approximation of the matrix

$$A_{ij} = \begin{cases} \frac{1}{|r_i - r_j|} & \text{if } i \neq j \\ 0, & \text{if } i = j \end{cases},$$
(5.1)

where $r_i \in \mathbb{R}^2$ or $r_i \in \mathbb{R}^3$ is the position of the *i*-th element. Elements are randomly distributed in identity square $\Omega = [0,1] \times [0,1]$ in \mathbb{R}^2 case and in identity cube $\Omega = [0,1] \times [0,1] \times [0,1]$ in \mathbb{R}^3 case. The \mathscr{H}^2 approximation accuracy is $\varepsilon = 10^{-6}$. In Figure 5.1 the factor S of the sparse factorization of the \mathscr{H}^2 matrix with the core (5.1) is marked by "inv".

EXAMPLE 5.2. \mathscr{H}^2 approximation of the matrix

$$A_{ij} = 2\delta_{ij} + \exp(-||r_i - r_j||^2), \qquad (5.2)$$

where $r_i \in \mathbb{R}^2$ or $r_i \in \mathbb{R}^3$ is the position of the *i*-th element. Elements are randomly distributed in identity square $\Omega = [0,1] \times [0,1]$ in \mathbb{R}^2 case and in identity cube $\Omega = [0,1] \times [0,1] \times [0,1]$ in \mathbb{R}^3 case. The \mathscr{H}^2 approximation accuracy is $\varepsilon = 10^{-6}$, In Figure 5.1 the factor S of the sparse factorization of the \mathscr{H}^2 matrix with the core (5.2) is marked by "exp".

We consider the sparse factorization (2.9) of the \mathcal{H}^2 matrices from Examples 5.1 and 5.2 in 2D and 3D. In Figure 5.1 we show the number of nonzero elements **per row** in the factor *S*. Since for all considered matrices the number of nonzero elements per row is constant (and does not grow with the matrix size), we conclude that the matrix *S* in factorization (2.9) is indeed sparse.



Fig. 5.1: Number of nonzero elements per row in factor S for \mathscr{H}^2 matrices from Examples 5.1 and 5.2 in 2D and 3D.

5. Numerical experiments. In Figure 5.2 we show the time of the sparse factorization of \mathcal{H}^2 matrices with the cores (5.1) and (5.2) (approximation accuracy is $\varepsilon = 10^{-6}$) denoted by "inv" and "exp".



Fig. 5.2: Timing of sparsification building for matrices from Examples 5.1 and 5.2 in 2D and 3D.

Sparsification time grows almost linearly. In Figure 5.3 we show the memory requirements of the matrix (5.1) in 2D and 3D, also we show the memory requirements of its \mathcal{H}^2 approximation ($\varepsilon = 10^{-6}$), of the sparse factorization (sum of U, S, and V) and of the factor S

separately.



Fig. 5.3: Storage requirements for the original matrix (5.1), its \mathscr{H}^2 approximation and the sparse factorization.

In Figure 5.4 we show the memory requirements of the matrix (5.2) in 2D and 3D, also we show the memory requirements of its \mathscr{H}^2 approximation ($\varepsilon = 10^{-6}$), of the sparse factorization (sum of U, S, and V) and of the factor S separately.



Fig. 5.4: Storage requirements for the original matrix (5.2), its \mathscr{H}^2 approximation and the sparse factorization.

For all examples, the memory requirements of both \mathcal{H}^2 and the sparse factorization grows almost linearly, unlike the memory requirements of the original matrix which scales quadratically.

5.3. Comparison to HODLR. Paper[3] considers the HODLR approximation of the dense matrix and its factorization as an efficient way to compute the determinant of the dense

matrix. We propose the \mathscr{H}^2 approximation of the dense matrix, its sparsification and factorization of the sparse matrix as an alternative. The triangular factorization of the sparse matrix is computed by CHOLMOD[12] package. Tests are performed for 3D data, for matrix

$$A_{ij} = 2\delta_{ij} + \exp(-||r_i - r_j||^2),$$

where $r_i \in \mathbb{R}^3$ is the position of the *i*-th element. Both HODLR and \mathscr{H}^2 approximation accuracy is $\varepsilon = 10^{-6}$. The HODLR factorization accuracy is $\vartheta = 10^{-5}$, the accuracy of the triangular factorization of the sparse matrix is $\vartheta = 10^{-10}$ (which is redundant, but the used package has no accuracy options). In Figure 5.5 we show the time comparison for this two approaches.



Fig. 5.5: Comparison of the \mathscr{H}^2 sparsification approach with HODLR solver in 3D.



Fig. 5.6: Comparison of the \mathscr{H}^2 sparsification approach with HODLR solver in 2D and 3D.

5.4. Comparison to H2Lib liberary. In this subsection we compare the approach proposed in this paper (sparse non-extensive factorization of the \mathcal{H}^2 matrix, triangular factorization of the sparse matrix and then the solution of the system) with the approach based on \mathcal{H}^2 -LU factorization, proposed in the work[9] and implemented in H2lib package[5]. The \mathcal{H}^2 -LU factorization takes the \mathcal{H}^2 matrix and returns the triangular factors *L* and *U* in \mathcal{H}^2 format. The triangular factorization of the sparse matrix is computed by CHOLMOD[12] package. Tests are performed on the following problem.

EXAMPLE 5.3. Consider the Dirichlet boundary value problem for Laplace's equation

$$\begin{cases} -\Delta u(x) = 0 & x \in \Omega, \\ u(x) = f(x) & x \in \Gamma = \partial \Omega, \end{cases}$$
(5.3)

where $\Omega = [-1,1]^3$ is a cube. The standard technic: using the single layer potential we obtain boundary integral formulation of the equation (5.3).

$$\int_{\Gamma} G(x-y)\varphi(y)\,ds_y = f(x),\tag{5.4}$$

where $G(x) = \frac{1}{4\pi} \frac{1}{|x|}$ is the fundamental solution for the Laplace operator. Then we discretize the integral equation (5.4) on the triangular grid on Γ using the Galerkin method. Obtained dense matrix is approximated in \mathcal{H}^2 format with accuracy $\varepsilon = 10^{-6}$. The accuracy of the \mathcal{H}^2 -LU factorization is $\vartheta = 10^{-6}$, the accuracy of the triangular factorization of the sparse matrix is $\vartheta = 10^{-10}$.

In Table 5.1 we show the time comparison of the solution of the system with matrix from Example 5.3, using \mathcal{H}^2 -LU and sparsification approaches. For the \mathcal{H}^2 -LU we show the approximation in \mathcal{H}^2 format and factorization time, for the sparsification we show the approximation in \mathcal{H}^2 format, sparsification and sparse factorization time. In both cases, time of the solution of the system with factorized matrix is negligible, so we do not show it.

Ν	3072	12288	49152	196608
H2Lib approx., sec	6.72	24.26	104.97	487.24
H2Lib factor., sec	13.56	164.00	1712.17	10970.43
Sp. approx., sec	4.8	26.34	110.91	399.43
Sp. sp., sec	2.16	10.12	63.47	323.23
Sp. factor., sec	0.19	1.30	7.83	56.89

Table 5.1: Comparison with H2Lib.

In Figure 5.7 we show the comparison of the total time, required for the system solution.



Fig. 5.7: Total time required for the system solution using \mathscr{H}^2 sparsification and H2lib solver.

The sparsification approach has not only better timing, but also better asymptotics.

5.5. Sparsification method as a preconditioner. \mathscr{H}^2 matrix is an efficient tool to multiply a matrix by a vector. This allows to apply iterative solvers like GMRES to solution of the systems with \mathscr{H}^2 matrix. But the preconditioning is still a challenging problem due to complexity of the factorization of the \mathscr{H}^2 matrix. We propose to use the approximately factored sparsification of the \mathscr{H}^2 matrix as a preconditioner to iterative method. For tests we use randomly distributed 3D data with following interaction matrix.

EXAMPLE 5.4.

$$A_{ij} = \begin{cases} 1 & \text{if } i = j \\ \frac{|r_i - r_j|}{d} & \text{if } 0 < |r_i - r_j| < d \\ \frac{d}{|r_i - r_j|}, & \text{if } |r_i - r_j| \ge d \end{cases}$$

where $r_i \in \mathbb{R}^3$ is the position of the *i*-th element. This example is used for testing of IFMM (Inverse Fast Multipole Method) method as a preconditioning in[11], so we have chosen this example for the convenient comparison.

This matrix is useful for the iterative tests since condition number of this matrix significantly depends on the parameter d: the larger d is, the larger condition number is.

Example with well-conditioned matrix. First, consider the matrix from Example 5.4 with $d = 10^{-3}$, condition number cond(A) = 10. We solve this system using GMRES iterative solver for the matrix approximated in \mathcal{H}^2 format with accuracy $\varepsilon = 10^{-9}$ and as a preconditioner we use \mathcal{H}^2 approximation of the matrix A with accuracy $\varepsilon = 10^{-3}$ sparsified and factorized with ILUt decomposition. Figure 5.8 shows convergence of the GMRES method with different ILUt threshold parameters. The required residual of the GMRES method $r = 10^{-10}$.



Fig. 5.8: Convergence of GMRES with different drop tolerance parameter of the ILUt preconditioner

The standard trade off: the more time on the preconditioner building we spend, the faster iterations converge. Figure 5.9 illustrates the total time required for the system solution (including the sparsification construction).



Fig. 5.9: Contribution into total time of sparsification, factorization and iterations

We show the total time required for \mathscr{H}^2 matrix sparsification, time required for building the ILUt preconditioner with dropping tolerance $\tau = 2 \times 10^{-2}$ and iterations timing in Figure 5.10a.



Fig. 5.10: Total solution time

Example with ill-conditioned matrix. Consider the matrix from Example 5.4 with $d = 10^{-2}$, condition number cond $(A) = 10^4$. As in the previous paragraph, we solve this system using GMRES iterative solver for the matrix approximated in \mathcal{H}^2 format with accuracy $\varepsilon = 10^{-9}$ and as a preconditioner we used \mathcal{H}^2 approximation of the matrix A with accuracy $\varepsilon = 10^{-3}$ sparsified and factorized with ILUt decomposition. In Figure 5.11 convergence till the tolerance 10^{-10} for different ILUt parameters is shown.



Fig. 5.11: Convergence of GMRES with different preconditioners, $N = 10^5$

In Figure 5.12 the total solution time for different ILUt parameters is shown.



Fig. 5.12: Contribution into total time of sparsification, factorization and iterations, $N = 10^5$

As we can see, the optimal ILUt parameter for this problem is $\tau = 10^{-2}$. Figure 5.13b presents the total time required for solution of the system with optimal ILUt parameter.



factorization and iterations

Fig. 5.13: Total solution time

We compared the results of our solver (GMRES preconditioned by sparsification method) with results of IFMM solver presented in[11] for the same problem, same parameters and similar hardware. For test we used matrix from Example 5.4 with 3D data, matrix size $N = 10^5$, iterative method GMRES till the residual $r = 10^{-10}$, parameters $d = \{10^{-3}, 10^{-2}\}$ for well-and ill-conditioned systems. We present the best total solution time achieved in experiments for both methods in Table 5.2.

	$d = 10^{-3}$	$d = 10^{-2}$
IFMM solution time, sec	118	301
Sparsification solution time, sec	96	218

Table 5.2: Comparison with IFMM method.

Note that sparsification method is implemented in Python programing language and can be significantly improved by applying Cython or switching to another programing language (as C++ or Fortran).

6. Related work. Hierarchical low-rank matrix formats such as $\mathscr{H}[15, 17, 16, 8]$, HODLR[1, 3] (Hierarchical Off-Diagonal Low-Rank), HSS[21, 10, 23] (Hierarchically Semiseparable), \mathscr{H}^2 [16, 6] matrices and etc., that are matrix analogies of the fast multipole method[14, 13], have two significant features: they do store information in data-sparse formats and they provide the fast matrix by vector product. Fast ($\mathscr{O}(N)$, where N is size of the matrix) matrix by vector product allows to apply iterative solvers. Data-sparse representation allows to store matrix in $\mathscr{O}(N)$ cells of memory, but storage scheme is usually complicated.

If the hierarchical matrix is ill-conditioned, then pure iterative solver fails and it is required to apply either approximate direct solver or preconditioner (that is also approximate direct solver, probably with lower accuracy). Due to complex storage schemes of hierarchical matrices, construction of the approximate direct solver is a challenging problem. There exists two general approaches to approximate direct solution of \mathcal{H}^2 matrix: factorization of hierarchical matrix[4, 21, 23], and sparsification of the hierarchical matrix followed by factorization of the sparse matrix[2, 26]. The factorization approach is more popular for hierarchical matrices with strong lowrank structure, also known as hierarchical matrices with weak-admissibility criteria[18] (\mathscr{H} [15, 17], HODLR[1, 3], HSS[21, 10, 23] matrices). For the \mathscr{H} matrix, the algorithm \mathscr{H} -LU[4] with almost linear complexity was proposed. This algorithm has been successfully applied to many problems. The major drawback of the \mathscr{H} -LU algorithm is that factorization time and memory required for *L* and *U* factors can be quite large. Approximate direct solvers based on factorization of HSS and HODLR matrices are also well studied and found many[23, 28, 20, 24, 1, 3, 21] successful applications.

One of the approaches to the solution of the systems with \mathcal{H}^2 matrices is the sparse factorization (sparsification). The sparsification approach is usually applied to hierarchical matrices with weak-admissibility criteria (\mathcal{H}^2 matrices). Sparsification algorithms transform \mathcal{H}^2 matrix into the sparse matrix and then factorize the sparse matrix. Algorithm, proposed in this paper is the sparsification algorithm. The main difference between presented work and the other sparse factorizations[2, 11, 26] is a size of sparse factors. The main benefit of the non-extensive sparsification is that it preserves the size of the factorized \mathcal{H}^2 matrix, while the other sparsification algorithms return extended factors.

7. Conclusions. We have proposed a new approach to the solution of the systems with \mathscr{H}^2 matrices that is based on sufficient non-extensive sparsification of the \mathscr{H}^2 matrix. Proposed sparsification is suitable for any \mathscr{H}^2 matrices (including non-symmetric) and preserves such important properties of the matrix as its size, symmetry (if exists) and positive definite (if exists).

REFERENCES

- S. AMBIKASARAN AND E. DARVE, An O(nlogn) fast direct solver for partial hierarchically semi-separable matrices, J. Sci. Comput., 57 (2013), pp. 477–501.
- [2] S. AMBIKASARAN AND E. DARVE, The inverse fast multipole method, arXiv preprint arXiv:1309.1773, (2014).
- [3] S. AMBIKASARAN, D. FOREMAN-MACKEY, L. GREENGARD, D. W. HOGG, AND M. ONEIL, Fast direct methods for gaussian processes, IEEE T. Pattern Anal., 38 (2016), pp. 252–265.
- [4] M. BEBENDORF, Hierarchical LU decomposition-based preconditioners for BEM, Computing, 74 (2005), pp. 225–247.
- [5] S. BÖRM, *H2lib package*. http://www.h2lib.org/.
- [6] —, Efficient numerical methods for non-local operators: H²-matrix compression, algorithms and analysis, vol. 14, European Mathematical Society, 2010.
- [7] —, *H²-matrix compression*, in New Developments in the Visualization and Processing of Tensor Fields, Springer, 2012, pp. 339–362.
- S. BÖRM, L. GRASEDYCK, AND W. HACKBUSCH, Introduction to hierarchical matrices with applications, Eng. Anal. Bound Elem., 27 (2003), pp. 405–422.
- [9] S. BÖRM AND K. REIMER, Efficient arithmetic operations for rank-structured matrices based on hierarchical low-rank updates, Computing and Visualization in Science, 16 (2013), pp. 247–258.
- [10] S. CHANDRASEKARAN, P. DEWILDE, M. GU, W. LYONS, AND T. PALS, A fast solver for HSS representations via sparse matrices, SIAM J. Matrix Anal. A., 29 (2006), pp. 67–81.
- [11] P. COULIER, H. POURANSARI, AND E. DARVE, The inverse fast multipole method: using a fast approximate direct solver as a preconditioner for dense linear systems, arXiv preprint arXiv:1508.01835, (2015).
- [12] T. A. DAVIS AND W. W. HAGER, Dynamic supernodes in sparse Cholesky update/downdate and triangular solves, ACM T. Math. Software, 35 (2009), p. 27.
- [13] L. GREENGARD AND V. ROKHLIN, A fast algorithm for particle simulations, J. Comput. Phys., 73 (1987), pp. 325–348.
- [14] L. GREENGARD AND V. ROKHLIN, The rapid evaluation of potential fields in three dimensions, Springer, 1988.
- [15] W. HACKBUSCH, A sparse matrix arithmetic based on *H*-matrices. part i: Introduction to *H*-matrices, Computing, 62 (1999), pp. 89–108.
- [16] W. HACKBUSCH, B. KHOROMSKIJ, AND S. SAUTER, On *H²-matrices*, in H.-J. Bungartz, et al. (eds.), Lectures on Applied Mathematics, Springer-Verlag, Berlin Heidelberg, 2000, pp. 9–30.

- [17] W. HACKBUSCH AND B. N. KHOROMSKIJ, A sparse h-matrix arithmetic., Computing, 64 (2000), pp. 21-47.
- [18] W. HACKBUSCH, B. N. KHOROMSKIJ, AND R. KRIEMANN, Hierarchical matrices based on a weak admissibility criterion, Computing, 73 (2004), pp. 207–243.
- [19] M. MA AND D. JIAO, Accuracy directly controlled fast direct solutions of general H²-matrices and its application to electrically large integral-equation-based electromagnetic analysis, arXiv preprint arXiv:1703.06155, (2017).
- [20] P. G. MARTINSSON, A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix, SIAM J. Matrix Anal. A., 32 (2011), pp. 1251–1274.
- [21] P.-G. MARTINSSON AND V. ROKHLIN, A fast direct solver for boundary integral equations in two dimensions, J. Comput. Phys., 205 (2005), pp. 1–23.
- [22] A. Y. MIKHALEV AND I. V. OSELEDETS, *Iterative representing set selection for nested cross approximation*, Numerical Linear Algebra with Applications, 23 (2016), pp. 230–248.
- [23] Z. SHENG, P. DEWILDE, AND S. CHANDRASEKARAN, Algorithms to solve hierarchically semi-separable systems, in System theory, the Schur algorithm and multidimensional analysis, Springer, 2007, pp. 255– 294.
- [24] S. SOLOVYEV, Multifrontal hierarchically solver for 3d discretized elliptic equations, in International Conference on Finite Difference Methods, Springer, 2014, pp. 371–378.
- [25] D. A. SUSHNIKOVA AND I. V. OSELEDETS, "Compress and eliminate" solver for symmetric positive definite sparse matrices, arXiv preprint arXiv:1603.09133, (2016).
- [26] D. A. SUSHNIKOVA AND I. V. OSELEDETS, Preconditioners for hierarchical matrices based on their extended sparse form, Russ. J. Numer. Anal. M., 31 (2016), pp. 29–40.
- [27] E. E. TYRTYSHNIKOV, Mosaic-skeleton approximations, Calcolo, 33 (1996), pp. 47–57.
- [28] J. XIA, S. CHANDRASEKARAN, M. GU, AND X. S. LI, Superfast multifrontal method for large structured linear systems of equations, SIAM J. Matrix Anal. A., 31 (2009), pp. 1382–1411.
- [29] K. YANG, H. POURANSARI, AND E. DARVE, Sparse hierarchical solvers with guaranteed convergence, arXiv preprint arXiv:1611.03189, (2016).