NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2016; **23**:230–248
Published online 5 November 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nla.2021

Iterative representing set selection for nested cross approximation

A. Yu. Mikhalev[1] and I. V. Oseledets[2,*,†]

[1]*Skolkovo Institute of Science and Technology, Novaya St. 100, Skolkovo, Odintsovsky district, 143025 Russia*
[2]*Institute of Numerical Mathematics, Russian Academy of Sciences. Gubkina St. 8, 119333 Moscow, Russia*

SUMMARY

A new fast algebraic method for obtaining an $\mathcal{H}^2$-approximation of a matrix from its entries is presented. The main idea behind the method is based on the nested representation and the maximum volume principle to select submatrices in low-rank matrices. A special iterative approach for the computation of so-called representing sets is established. The main advantage of the method is that it uses only the hierarchical partitioning of the matrix and does not require special 'proxy surfaces' to be selected in advance.

The numerical experiments for the electrostatic problem and for the boundary integral operator confirm the effectiveness and robustness of the approach. The complexity is linear in the matrix size and polynomial in the ranks. The algorithm is implemented as an open-source Python package that is available online.

Received 24 September 2014; Revised 3 September 2015; Accepted 18 September 2015

KEY WORDS: hiearchical matrices; cross approximation; skeleton decomposition; low-rank approximation

1. INTRODUCTION

Cross approximation [1, 2] is a basic reduction technique for the approximation of large low-rank matrices. These matrices often appear as blocks of dense matrices, coming from the discretization of non-local operators (in particular, in FEM/BEM applications). The whole matrix is split into large blocks related to geometrically separated sets of sources and receivers, and those blocks are approximated by low-rank matrices. This partitioning forms the basis for the mosaic-skeleton method [3] or the $\mathcal{H}$-matrix format [4, 5]. The approximation can be computed only from the entries of the matrix and from the additional geometrical information that induces the partitioning of the matrix into blocks. The simplest partitioning scheme corresponding to the so-called *weak admissibility condition* [6] is presented on Figure 1 and is usually used for one-dimensional problems.

In the $\mathcal{H}$-matrix format, the blocks are approximated independently of each other, which leads to the logarithmic factor in storage and complexity. The $\mathcal{H}^2$-matrix format [7] provides a more effective representation of a matrix based on the analogy with the well-known fast multipole method [8]. The approximation is performed by taking into account common information about the column and row bases in the low-rank blocks. This leads to a 'nested' representation: we do not need to store the factors for each block, but only the coefficients that allow us to express the bases on the upper level from the previous level bases (so-called *transfer matrices*) and coefficients of interactions between bases of sources and receivers (so-called *interaction matrices*). This structure may lead to significant reduction in memory and complexity [9].

*Correspondence to: Ivan Oseledets, Skolkovo Institute of Science and Technology, Novaya St. 100, Skolkovo, Odintsovsky district, 143025, Russia.
†E-mail: i.oseledets@skoltech.ru

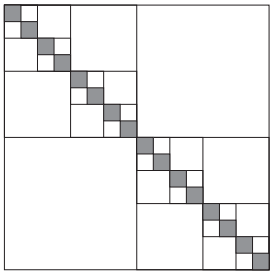

Figure 1. Simplest mosaic partitioning. The blocks denoted in white correspond to low-rank matrices.

If the partitioning of a matrix into blocks is fixed, and the subroutine that allows to compute any prescribed element of a matrix is given, then the main problem is to compute the approximation of such a matrix without computing all of its entries (that would give $\mathcal{O}(N^2)$ complexity because the matrix is dense). For the $\mathcal{H}$-matrix format, the problem reduces to the approximation of a low-rank matrix from its entries. A rank-r matrix can be exactly recovered from r columns and rows using skeleton decomposition [10] and in the approximate case; the existence of a quasi-optimal skeleton decomposition can be proved based on the maximal volume principle [11]. In practice, such approximations are computed using cross approximation techniques [1, 2].

It is natural to consider a question, whether the $\mathcal{H}^2$-matrix representation can be computed directly from the entries of a matrix in a quasi-optimal cost (i.e. the number of sampled elements is close to the number of parameters in the representation). This construction can be performed efficiently using interpolation [12], but it is not a purely algebraic approach. In [13] a *nested cross approximation* was proposed that uses only the entries of the matrix. Two methods were described, ACAGEO and ACAMERGE, which rely on different sampling strategies for the *far zone* (in other words, selection of the *representing sets*). This selection was not adaptive for ACAGEO, and for ACAMERGE, as we will see in numerical experiments; it may lead to significant loss of accuracy. We propose a new purely algebraic method for the adaptive selection of the representing sets to compute the $\mathcal{H}^2$-matrix approximant from the entries. The algorithm was motivated by two completely different approaches. Its first step comes as a generalization of the classical Barnes-Hut algorithm [14], which can be considered as one of the first algorithms for the fast approximate computation of dense matrix-by-vector products. The obtained approximation is satisfactory for small accuracies (say, $\varepsilon = 10^{-3}$) but does not perform well when the approximation threshold goes to zero. To make the method more robust, we establish the analogy between $\mathcal{H}^2$-matrix approximation and multidimensional tensors, and apply the ideas from the cross approximation of tensors [15] to our problem. The top-to-bottom and bottom-to-top tree traversal algorithm, proposed in this paper, is similar to the left-right sweeps in the TT-cross algorithm of [15]. By numerical experiments, we show that the resulting approach is robust in the sense that it is able to achieve better accuracy in comparison with the ACAGEO and ACAMERGE algorithms of [13].

Related work. The problem of computing the $\mathcal{H}^2$-matrix approximation from the entries was first considered in this setting in [13]. Our work follows a similar framework but has several important differences, and the proposed algorithm is more robust for high approximation accuracies. Similar approaches for constructing the approximation were proposed in the *kernel-independent fast multipole methods* [16], but they still use additional information from the problem. Different wavelet-based methods [17, 18], which proved to have superior performance over hierarchical matrices, suffer from a similar problem: they require additional information from the problem and require the computation of Galerkin elements with wavelet basis function. Some other approaches consider 'hierarchically semiseparable matrices' [19]; this format is very close to $\mathcal{H}^2$. For this class of matrices (HSS-matrices), randomized algorithm has been proposed to recover the representation from the entries [20]. However, the HSS representation is essentially a one-dimensional $\mathcal{H}^2$-matrix, and the randomized algorithm of [20] also requires a fast matrix-by-vector procedure to be available.

We organize our paper as follows. In Section 2, we gather all necessary mathematical tools to work with $\mathcal{H}^2$-matrices. In Section 3, we derive the multicharge Barnes–Hut (MBCH) representation

DOI: 10.1002/nla

for the $\mathcal{H}^2$-matrix that is main approximation ansatz and propose a simple algorithm to compute it. In Section 4, the connection with good submatrices is discussed. In Section 5, the main improvement of the algorithm is proposed. In Section 6, we provide numerical experiments. And finally, in Section 7, we compare our iterative approach, proposed in Section 5, with methods from [13].

2. BASIC NOTATIONS AND DEFINITIONS

A detailed review can be found in the books [7, 9]. Here, we give a summary of our notations.

To define the $\mathcal{H}^2$-matrix format, we need several standard definitions. The main idea is to split a matrix into large blocks that are approximately of low rank. The matrix element A_{ij} describes interaction between the i-th receiver x_i and j-th source y_j. Typical examples are function-related matrices $A_{ij} = f(x_i, y_j)$ and integral operators, discretized by collocation, Nyström or Galerkin methods with basis functions with local support. The whole set of receivers will be denoted by indices $\mathbf{I}$, and the whole set of sources will be denoted by indices $\mathbf{J}$. For simplicity, we will consider only binary trees, but the extension to an arbitrary number of sons is trivial.

Definition 2.1 (Row and column cluster trees)
Because sources, denoted by $\mathbf{J}$, are presented by columns and receivers, denoted by $\mathbf{I}$, are presented by rows of matrix A, let call $\mathcal{T}_\mathbf{I}$ as row cluster tree and $\mathcal{T}_\mathbf{J}$ as column cluster tree,

(1) if $\mathbf{I}$ is the root of $\mathcal{T}_\mathbf{I}$ and $\mathbf{J}$ is the root of $\mathcal{T}_\mathbf{J}$; and
(2) if $t \in \mathcal{T}_\mathbf{I}$, then t is a disjoint union of its sons $\mathbf{t_1} \in \mathcal{T}_\mathbf{I}$ and $\mathbf{t_2} \in \mathcal{T}_\mathbf{I}$; if $t \in \mathcal{T}_\mathbf{J}$, then t is a disjoint union of its sons $\mathbf{t_1} \in \mathcal{T}_\mathbf{J}$ and $\mathbf{t_2} \in \mathcal{T}_\mathbf{J}$:

$$\mathbf{sons(t)} = \{\mathbf{t_1}, \mathbf{t_2}\}.$$

The cluster trees are constructed for the rows and the columns of the matrix and are typically based on a certain geometrical construction (i.e., kd-trees, quadtrees, and many others). The construction of the partition $\mathbf{P}$ of a given matrix into low-rank blocks is based on the *admissibility condition*. The standard form of the admissibility condition is also geometrical. A set of receivers $\mathbf{Y}$ and a set of sources $\mathbf{X}$ are said to be admissible, if

$$\max\{\mathrm{diam}\mathbf{Y}, \mathrm{diam}\mathbf{X}\} \leqslant \eta \mathrm{dist}(\mathbf{Y}, \mathbf{X}),$$

where $\eta \geqslant 0$ (it is equal to 0 for Figures 1 and 2). In this case, it can be proven that for a class of asymptotically smooth kernels [2]; the corresponding block can be well-approximated by a low-rank matrix. This condition is simple to check, and the recursive partition is typically generated by a recursive procedure. It starts from the roots of the cluster trees. If for a given block an admissibility condition is satisfied, the block $(t, \mathbf{s})$ is added to the partition $\mathbf{P}$. Otherwise, all sons of t and $\mathbf{s}$ are processed recursively. This process creates a list of admissible blocks $(\mathbf{t}, \mathbf{s})$.

Definition 2.2 ($\chi_\mathbf{t}$, $\chi_\mathbf{s}$)
Let $\mathbf{t} \in \mathcal{T}_\mathbf{I}$, then $\chi_\mathbf{t}$ is a diagonal matrix with the following property: $\mathbf{i}$-th diagonal element is 1 if the row $\mathbf{i} \in \mathbf{t}$ and 0 otherwise. Let $\mathbf{s} \in \mathcal{T}_\mathbf{J}$, then $\chi_\mathbf{s}$ is a diagonal matrix with the following property: $\mathbf{i}$-th diagonal element is 1 if the column $\mathbf{i} \in \mathbf{s}$ and 0 otherwise.

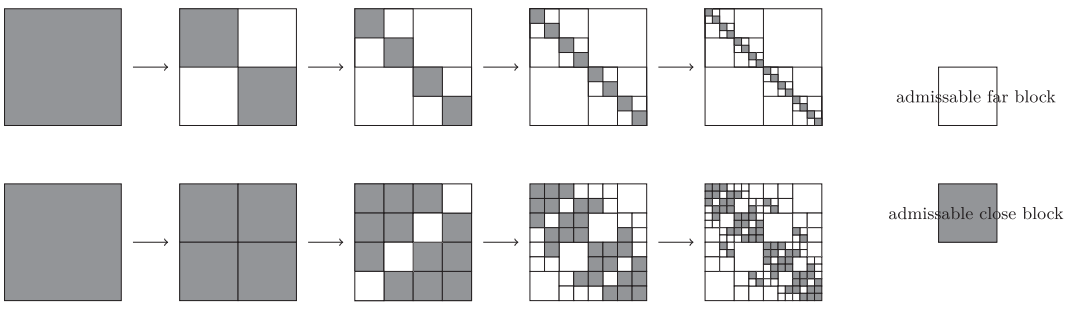

Figure 2. Examples of mosaic partitioning in one and two dimensions.

DOI: 10.1002/nla

For any pair of admissible nodes $\mathbf{t}$ and $\mathbf{s}$, their corresponding submatrix $A(\mathbf{t}, \mathbf{s})$ can be written as follows:

$$A(\mathbf{t}, \mathbf{s}) = \chi_{\mathbf{t}} A \chi_{\mathbf{s}}.$$

Definition 2.3 ($\mathbf{S_t}, \mathbf{S_s}$)
Assume $\mathbf{t}$ is a node of row cluster tree $\mathcal{T}_{\mathbf{I}}$. Then 'set of admissible nodes', $\mathbf{S_t} = \{\mathbf{s_0}, \ldots, \mathbf{s_k}\}$, is a set of nodes of a column cluster tree $\mathcal{T}_{\mathbf{J}}$, so that each pair $(\mathbf{t}, \mathbf{s_i})$ is admissible if and only if $\mathbf{s_i} \in \mathbf{S_t}$. Similarly, $\mathbf{S}_s$ is a set of admissible nodes for node $\mathbf{s} \in \mathcal{T}_{\mathbf{J}}$.

Definition 2.4 ($\mathbf{R_t^*}, \mathbf{C_s^*}$)
Assume $\mathbf{t}$ is a node of row cluster tree $\mathcal{T}_{\mathbf{I}}$ with a corresponding set of admissible nodes $\mathbf{S}_t$. An admissible block row $\mathbf{R_t^*}$ is defined as follows:

$$\mathbf{R_t^*} = \chi_{\mathbf{t}} A \sum_{\mathbf{s} \in \mathbf{S}_t} \chi_{\mathbf{s}}.$$

Similarly, we define admissible block column $\mathbf{C_s^*}$ as follows:

$$\mathbf{C_s^*} = \left(\sum_{\mathbf{t} \in \mathbf{S}_s} \chi_{\mathbf{t}} \right) A \chi_{\mathbf{s}}.$$

*Definition 2.5 (*pred(t))
Assume $\mathbf{t} = \mathbf{t_0}$, and $\{\mathbf{t_1}, \ldots, \mathbf{t_k}\}$ is a set of nodes of cluster tree $\mathcal{T}_{\mathbf{I}}$ or $\mathcal{T}_{\mathbf{J}}$, such that $\mathbf{t_k}$ is the root of tree, and $\mathbf{t_{k-i}}$ is the parent of $\mathbf{t_{k-i-1}}$ for each $\mathbf{i} = 0, \ldots, \mathbf{k-1}$. Then, the set $\{\mathbf{t_1}, \ldots, \mathbf{t_k}\}$ is called 'predecessors' of the node $\mathbf{t}$ and can be written as

$$\mathbf{pred(t)} = \{\mathbf{t_1}, \ldots, \mathbf{t_k}\}.$$

Definition 2.6 ($\mathbf{R_t^+}, \mathbf{C_s^+}, \mathbf{R_t}, \mathbf{C_1}$)
Assume $\mathbf{t} = \mathbf{t_0}$, and $\{\mathbf{t_1}, \ldots, \mathbf{t_k}\} = \mathbf{pred(t)}$ are predecessors of the node $\mathbf{t} \in \mathcal{T}_{\mathbf{I}}$. $\{\mathbf{S}_0, \ldots, \mathbf{S}_k\}$ are sets of admissible nodes corresponding to $\{\mathbf{t_0}, \ldots, \mathbf{t_k}\}$. Then $\mathbf{R_t^+}$ and $\mathbf{R_t}$ are defined as follows:

$$\mathbf{R_t^+} = \chi_{\mathbf{t}} A \sum_{\mathbf{i}=1}^{\mathbf{k}} \sum_{\mathbf{s} \in \mathbf{S}_i} \chi_{\mathbf{s}},$$

$$\mathbf{R_t} = \mathbf{R_t^+} + \mathbf{R_t^*} = \chi_{\mathbf{t}} A \sum_{\mathbf{i}=0}^{\mathbf{k}} \sum_{\mathbf{s} \in \mathbf{S}_i} \chi_{\mathbf{s}},$$

where $\mathbf{R_t}$ is called 'block row'. As for $\mathbf{s_0} = \mathbf{s} \in \mathcal{T}_{\mathbf{J}}$, we define the following

$$\{\mathbf{s_1}, \ldots, \mathbf{s_k}\} = \mathbf{pred(s)},$$

$$\mathbf{S_i} = \mathbf{S_{s_i}},$$

$$\mathbf{C_s^+} = \left(\sum_{\mathbf{i}=1}^{\mathbf{k}} \sum_{\mathbf{t} \in \mathbf{S_i}} \chi_{\mathbf{t}} \right) A \chi_{\mathbf{s}},$$

$$\mathbf{C_1} = \mathbf{C_s^+} + \mathbf{C_s^*} = \left(\sum_{\mathbf{i}=0}^{\mathbf{k}} \sum_{\mathbf{t} \in \mathbf{S_i}} \chi_{\mathbf{t}} \right) A \chi_{\mathbf{s}},$$

where $\mathbf{C_1}$ is called 'block column'.

DOI: 10.1002/nla

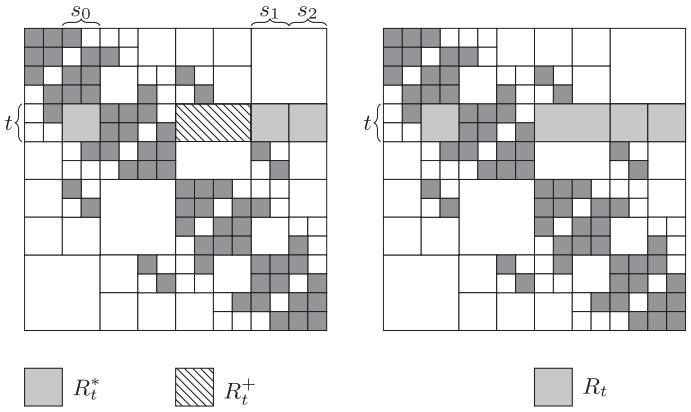

Figure 3. The different block rows for node $\mathbf{t}$ are $\mathbf{R_t}$, $\mathbf{R_t^+}$, and $\mathbf{R_t^*}$.

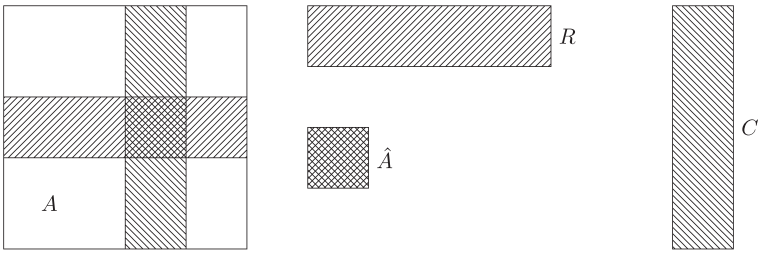

Figure 4. Skeleton decomposition of matrix A by basis rows R and basis columns C.

All kinds of block rows are shown in Figure 3.

If $\mathbf{t} \in \mathcal{T}_\mathbf{I}$ is a nonleaf node with $\mathbf{sons(t)} = \{\mathbf{t_1}, \mathbf{t_2}\}$, then block row $\mathbf{R_t}$ can be computed through block rows of its sons:

$$\mathbf{R_t} = \mathbf{R_{t_1}^+} + \mathbf{R_{t_2}^+}.$$

3. NESTED CROSS APPROXIMATION

In the following, we will always assume exact low-rank property for simplicity; however, all the derivation is true for approximate low-rank case as well. In this case, the equality sign should be replaced by approximate equality sign. The $\mathcal{H}^2$-matrix structure has a simple algebraic characterization: all block rows and columns have bounded rank. From this property, we can derive a low-rank representation based on the skeleton decomposition [21]. Recall that the skeleton decomposition for a low-rank matrix A has the following form (see Figure 4):

$$A = C\,\hat{A}^{-1}R,$$

where C consists of **basis columns** of A, R consists of **basis rows** of A, and $\hat{A}$ is the submatrix of A on the intersection of basis rows and columns. Simple modification of this formula gives us formula for approximating entire matrix with its basis rows or columns:

$$A = \tilde{C}\,R = C\,\tilde{R},$$

with $\tilde{C} = C\,\hat{A}^{-1}$, and $\tilde{R} = \hat{A}^{-1}R$.

From here, we assume that matrix A is an $\mathcal{H}^2$-matrix and every block row and block column is a low-rank matrix, and all following equations are written as equalities (for convenience). Because any matrix from real problem can only be close to $\mathcal{H}^2$-matrix (each block column and block row is ε-low-rank), we refer to error analysis from [13].

DOI: 10.1002/nla

Let us assume that $\mathbf{t}$ is a nonleaf node of row cluster tree $\mathcal{T}_\mathbf{I}$ with sons $\mathbf{t_1}$ and $\mathbf{t_2}$. From the skeleton decomposition for block rows $\mathbf{R_t}, \mathbf{R_{t_1}}$, and $\mathbf{R_{t_2}}$, we acquire the following formulas for $\mathbf{R_t}, \mathbf{R_{t_1}^+}$, and $\mathbf{R_{t_2}^+}$:

$$\mathbf{R_t} = U_\mathbf{t} \hat{R}_\mathbf{t}^{-1} V_\mathbf{t},$$

$$\mathbf{R_{t_1}^+} = U_{\mathbf{t}_1} \left(\hat{R}_{\mathbf{t}_1}^+\right)^{-1} V_{\mathbf{t}_1},$$

$$\mathbf{R_{t_2}^+} = U_{\mathbf{t}_2} \left(\hat{R}_{\mathbf{t}_2}^+\right)^{-1} V_{\mathbf{t}_2},$$

where $V_\mathbf{t}, V_{\mathbf{t}_1}$, and $V_{\mathbf{t}_2}$ are based on basis rows of block rows $\mathbf{R_t}, \mathbf{R_{t_1}}$, and $\mathbf{R_{t_2}}$ correspondingly. Because $\mathbf{R_t} = \mathbf{R_{t_1}^+} + \mathbf{R_{t_2}^+}$, block row $\mathbf{R_t}$ can be rewritten as

$$\mathbf{R_t} = U_{\mathbf{t}_1} \hat{R}_{\mathbf{t}_1}^{-1} V_{\mathbf{t}_1} + U_{\mathbf{t}_2} \hat{R}_{\mathbf{t}_2}^{-1} V_{\mathbf{t}_2} = \begin{bmatrix} U_{\mathbf{t}_1} \left(\hat{R}_{\mathbf{t}_1}^+\right)^{-1} & U_{\mathbf{t}_2} \left(\hat{R}_{\mathbf{t}_2}^+\right)^{-1} \end{bmatrix} \begin{bmatrix} V_{\mathbf{t}_1} \\ V_{\mathbf{t}_2} \end{bmatrix}.$$

Because $V_{\mathbf{t}_1}$ and $V_{\mathbf{t}_2}$ are rows of matrix $\mathbf{R_t}$, they can be obtained via skeleton decomposition of $\mathbf{R_t}$:

$$\begin{bmatrix} V_{\mathbf{t}_1} \\ V_{\mathbf{t}_2} \end{bmatrix} = \hat{U}_\mathbf{t} \hat{R}_\mathbf{t}^{-1} V_\mathbf{t}.$$

Matrix $\hat{U}_\mathbf{t} \hat{R}_\mathbf{t}^{-1}$ is a *local transfer matrix* from basis rows of block row $\mathbf{R_t}$ to basis rows of block rows $\mathbf{R_{t_1}}$ and $\mathbf{R_{t_2}}$. Here and later, we mark it as $M_\mathbf{t}$. Matrices $U_{\mathbf{t}_1} \hat{R}_{\mathbf{t}_1}^{-1}$ and $U_{\mathbf{t}_2} \hat{R}_{\mathbf{t}_2}^{-1}$ are *global transfer matrices* from basises of block rows $\mathbf{R_{t_1}}$ and $\mathbf{R_{t_2}}$ to all corresponding rows. They are marked as $P_{\mathbf{t}_1}$ and $P_{\mathbf{t}_2}$. Within new notations, we can rewrite low-rank decomposition of block row $\mathbf{R_t}$:

$$\mathbf{R_t} = \begin{bmatrix} P_{\mathbf{t}_1} & P_{\mathbf{t}_2} \end{bmatrix} M_\mathbf{t} V_\mathbf{t}.$$

For the following argumentation, we will need additional definition.

Definition 3.1 ($\hat{\chi}_\mathbf{t}$, $\hat{\chi}_\mathbf{s}$)
Let $\hat{\chi}_\mathbf{t}$ for $\mathbf{t} \in \mathcal{T}_\mathbf{I}$ be a diagonal matrix with the following property: $\mathbf{i}$-th diagonal element is 1 if the row $\mathbf{i}$ is in the set of basis rows for the block row $\mathbf{R_t}$ and 0 otherwise. Let $\hat{\chi}_\mathbf{s}$ for $\mathbf{s} \in \mathcal{T}_\mathbf{J}$ be a diagonal matrix with the following property: $\mathbf{i}$-th diagonal element is 1 if the column $\mathbf{i}$ is in the set of basis columns for the block column $\mathbf{C_s}$ and 0 otherwise.

For any admissible pair of nodes $(\mathbf{t}, \mathbf{s})$, corresponding submatrix $A(\mathbf{t}, \mathbf{s})$ can be written as follows:

$$A(\mathbf{t}, \mathbf{s}) = \mathbf{R_t} \chi_\mathbf{s} = P_\mathbf{t} V_\mathbf{t} \chi_\mathbf{s} = P_\mathbf{t} \hat{\chi}_\mathbf{t} A \chi_\mathbf{s} = P_\mathbf{t} \hat{\chi}_\mathbf{t} A(\mathbf{t}, \mathbf{s}),$$

where $P_\mathbf{t}$ is a global transfer matrix and $V_\mathbf{t}$ is a matrix, based on basis rows of $\mathbf{R_t}$. Because $\mathbf{C_1} = U_\mathbf{s} P_\mathbf{s}$, where $U_\mathbf{s}$ is a matrix, based on basis columns of $\mathbf{C_1}$, and $P_\mathbf{s}$ is a global transfer matrix, we also get

$$A(\mathbf{t}, \mathbf{s}) = A(\mathbf{t}, \mathbf{s}) \hat{\chi}_\mathbf{s} P_\mathbf{s}.$$

Transforming two equations into one

$$A(\mathbf{t}, \mathbf{s}) = P_\mathbf{t} \hat{\chi}_\mathbf{t} A(\mathbf{t}, \mathbf{s}) \hat{\chi}_\mathbf{s} P_\mathbf{s}.$$

Finally, we obtain

$$A(\mathbf{t}, \mathbf{s}) = P_\mathbf{t} \hat{A}(\mathbf{t}, \mathbf{s}) P_\mathbf{s},$$

where $\hat{A}(\mathbf{t}, \mathbf{s})$ is a submatrix, based on basis rows of $\mathbf{R_t}$ and basis columns of $\mathbf{C_1}$. Taking into account that $P_\mathbf{t}$ and $P_\mathbf{s}$ can be computed recursively, we obtain an $\mathcal{H}^2$-type factorization.

DOI: 10.1002/nla

First and most simple idea, proposed in the article, is to choose basis rows for each block row $\mathbf{R_t}$ hierarchically: use $\hat{\chi}_{\mathbf{t_1}}$ and $\hat{\chi}_{\mathbf{t_2}}$ to obtain $\hat{\chi}_{\mathbf{t}}$, where $\mathbf{t_1}$ and $\mathbf{t_2}$ are sons of the node $\mathbf{t}$. Technically, we can write it as follows:

$$\hat{\chi}_{\mathbf{t_1}}\mathbf{R_t} + \hat{\chi}_{\mathbf{t_2}}\mathbf{R_t} = M_{\mathbf{t}}\hat{\chi}_{\mathbf{t}}\mathbf{R_t}.$$

Obviously, matrices $M_{\mathbf{t}}$ and $\hat{\chi}_{\mathbf{t}}$ can be computed with help of skeleton decomposition. The same holds true for basis columns of each block column $\mathbf{C_1}$:

$$\mathbf{C_1}\hat{\chi}_{\mathbf{s_1}} + \mathbf{C_1}\hat{\chi}_{\mathbf{s_2}} = \mathbf{C_1}\hat{\chi}_{\mathbf{s}}M_{\mathbf{s}}.$$

Because all basises are chosen hierarchically, this kind of $\mathcal{H}^2$-decomposition can be named as *nested cross approximation*. In assumption, we generalize proposed proto-method for row cluster tree $\mathcal{T}_{\mathbf{I}}$ in Algorithm 1, which can be easily adapted for column cluster tree $\mathcal{T}_{\mathbf{J}}$.

Algorithm 1 Computation of basis rows and transfer matrices for each node of row cluster tree.

Require: Row cluster tree $\mathcal{T}_{\mathbf{I}}$, block row $\mathbf{R_t}$ for each node $\mathbf{t} \in \mathcal{T}_{\mathbf{I}}$.
Ensure: nested basises $\hat{\chi}_{\mathbf{t}}$ and transfer matrices $M_{\mathbf{t}}$ for each node of $\mathcal{T}_{\mathbf{I}}$.

```
for each t ∈ T_I {from bottom of the row cluster tree to top} do
    {define auxiliary matrix R̂ to compute basis rows and transfer matrix}
    if sons(t) is empty then
        R̂ = R_t
    else {sons(t) is not empty, {t_1, t_2} = sons(t)}
        R̂ = (χ̂_{t_1} + χ̂_{t_2}) R_t
    end if
    {compute basis rows χ̂_t and transfer matrix M_t}
    R̂ = M_t χ̂_t R̂ {obviously, χ̂_t R̂ = χ̂_t R_t}
end for
```

4. MAXIMUM VOLUME PRINCIPLE

The main idea of the nested cross approximation is to select basis rows and basis columns for special auxiliary matrices ($\hat{R}$ from the Algorithm 1). If each block row has precise rank, it does not matter what rows we choose for basises. Problem arises when block rows are close to low-rank matrices. Skeleton approximation

$$A \approx C\,\hat{A}^{-1}R$$

is about choosing submatrix $\hat{A}$. If it has very small volume (modulus of determinant), it is very close to singular, and approximation error is close to infinity. Guaranteed approximation error can be obtained with the help of *maximum volume principle* [10]: if $\hat{A}$ is a submatrix of maximum volume (modulus of determinant) among all $r \times r$ submatrices, following estimation holds true:

$$\|A - C\,\hat{A}^{-1}R\|_C \leqslant (r+1)\sigma_{r+1},$$

where σ_{r+1} is a $r+1$ singular value of matrix A and $\|\cdot\|_C$ is an infinite or Chebyshev norm (maximum in modulus element in matrix).

Finding maximum volume submatrix has exponential complexity, but suboptimal submatrices can be found in a fast way by using greedy algorithms. First of all, we can find a good submatrix in a low rank matrix, close to A:

$$A \approx UV^T + E,\ A \in \mathbb{C}^{n\times m}, E \in \mathbb{C}^{n\times m}, U \in \mathbb{C}^{n\times r}, V \in \mathbb{C}^{m\times r}.$$

DOI: 10.1002/nla

Because

$$\det(XY) = \det(X)\det(Y)$$

for any given square matrices X and Y, computation of good submatrix in A requires computation of r good rows in matrix U and r good rows in matrix V. Recall that our initial problem was to define basis rows for block rows and basis columns for block columns, so we need either left factor or right factor. So without prejudice to the generality, we need to find a good $r \times r$ submatrix in an $n \times r$ matrix. Following Algorithm 2, *maximum-volume algorithm* [22] (which we call *maxvol*), solves this problem.

Suppose we need to find a good $r \times r$ submatrix in a matrix $A \in \mathbb{C}^{n\times r}$, with $n > r$. First step of *maxvol* algorithm is to find a nonsingular submatrix of matrix A, so we use the LU decomposition with row pivoting, which requires $O(nr^2)$ operations. In other words, initialization gives us a submatrix $\hat{A} \in \mathbb{C}^{r\times r}$ and coefficients $C \in \mathbb{C}^{n\times r}$ such that $A = C\hat{A}$. Each iteration of the algorithm swaps two rows in order to increase the volume of the submatrix. As shown in [22], each iteration requires only $O(nr)$ operations because it finds absolute maximum element in C and applies rank-1 update to it. In practice, the algorithm converges very fast, so we can estimate the complexity of *maxvol* algorithm as $O(nr^2)$ operations.

Algorithm 2 *maxvol* [22] algorithm

Require: Nonsingular matrix $A \in \mathbb{C}^{n\times r}, n > r$
Ensure: Good submatrix $\hat{A} \in \mathbb{C}^{r\times r}$ and coefficients $C \in \mathbb{C}^{n\times r}$ such, that $A = C\hat{A}$
1: Find nonsingular $r \times r$ submatrix $\hat{A}$ in matrix A
2: {i.e. by pivots from LU factorization}
3: $C \leftarrow A\hat{A}^{-1}, \{i, j\} \leftarrow \text{argmax}(|C|)$
4: {$|C|$ means element-wise modulus, i and j are row and column numbers of maximum in modulus element in C}
5: **while** $|C_{ij}| > 1$ **do**
6: {C_{ij} is element on intersection of i-th row and j-th column of C}
7: $\hat{A}_j \leftarrow A_i$
8: {A_i is i-th row of A, $\hat{A}_j$ is j-th row of $\hat{A}$}
9: $C \leftarrow C - C_j(C_i - e_j)/C_{ij}$
10: {C_i is i-th row and C_j is j-th column of C, e_j is j-th row of identity matrix of size $r \times r$}
11: $\{i, j\} \leftarrow \text{argmax}(|C|)$
12: **end while**
13: **return** $C, \hat{A}$

5. PRACTICAL ALGORITHM FOR NESTED CROSS APPROXIMATION

The main disadvantage of the Algorithm 1 is computational cost. Auxiliary matrix $\hat{R}$ of each leaf node $\mathbf{t} \in \mathcal{T}_\mathbf{I}$ contains $n_\mathbf{t}$ (number of receivers, corresponding to node $\mathbf{t}$) nonzero rows and $N_\mathbf{t}$ (number of sources in far zone of $\mathbf{t}$ or **pred(t)**) nonzero columns. Because $\mathbf{t}$ is a nonleaf node, $N_\mathbf{t}$ is close to the number of sources. Obviously, minimum number of operations to compute basis rows for node $\mathbf{t}$ is a multiplication of $n_\mathbf{t}$ and $N_\mathbf{t}$. Summing it for all leaf nodes, we acquire a $O(NM)$ complexity, where N is the number of sources and M is the number of receivers. So we need to reduce the number of nonzero columns of $\hat{R}$, that is, to select a certain *representing set*.

Let us assume that we already have representing sets for each node of $\mathcal{T}_\mathbf{I}$ and $\mathcal{T}_\mathbf{J}$. It is again convenient to use diagonal matrices to work with representing sets.

Definition 5.1 ($\psi_\mathbf{t}$, $\psi_\mathbf{s}$)
Let $\psi_\mathbf{t}$ be a diagonal matrix with the following property: $\mathbf{i}$-th diagonal element is 1 if the column $\mathbf{i}$ is in the representing set for the block row $\mathbf{R}_\mathbf{t}$ and 0 otherwise. Let $\psi_\mathbf{s}$ be a diagonal matrix with

DOI: 10.1002/nla

the following property: **i**-th diagonal element is 1 if the row **i** is in the representing set for the block column $\mathbf{C_1}$ and 0 otherwise.

If a good representing set for each block row $\mathbf{R_t}$ is already known, then basis rows and transfer matrix for each block row can be computed with the help of a small matrix $\hat{R}\psi_{\mathbf{t}}$ instead of $\hat{R}$. *Maxvol* Algorithm 2 can be used to find basis rows or columns together with transfer matrices in an efficient way. Finally, we acquire a prototype Algorithm 3.

Algorithm 3 Computation of basis rows and transfer matrices for row cluster tree with given representing sets.

Require: Row cluster tree $\mathcal{T}_{\mathbf{I}}$, block row $\mathbf{R_t}$ and representing set $\psi_{\mathbf{t}}$ for each node $\mathbf{t} \in \mathcal{T}_{\mathbf{I}}$, accuracy parameter ε.
Ensure: basis rows $\hat{\chi}_{\mathbf{t}}$ and transfer matrix $M_{\mathbf{t}}$ for each block row $\mathbf{R_t}$.
  for $\mathbf{t} \in \mathcal{T}_{\mathbf{I}}$ {from bottom of the row cluster tree to top} **do**
    {define auxiliary matrix $\hat{R}$ to compute basis rows and transfer matrix}
    if sons(t) is empty **then**
      $\hat{R} = \mathbf{R_t}\psi_{\mathbf{t}}$
    else {**sons(t)** is not empty, $\{\mathbf{t_1}, \mathbf{t_2}\} = \mathbf{sons(t)}$}
      $\hat{R} = (\hat{\chi}_{\mathbf{t_1}} + \hat{\chi}_{\mathbf{t_2}})\,\mathbf{R_t}\psi_{\mathbf{t}}$
    end if
    {truncated svd + maxvol}
    $U, S, V = svd(\hat{R}, tol = \varepsilon)$; $M_{\mathbf{t}}, \hat{\chi}_{\mathbf{t}} = maxvol(U)$
  end for

5.1. *Partially fixed representing sets*

The main problem is how to select these representing sets adaptively. Previous approaches [13, 16, 23] used geometrical constructions. In this paper, we propose a purely algebraical method: each representing set is divided into self and predecessors parts, each of which is calculated separately. First of all, assume that predecessors part of each representing set is predefined, then to find rather good self part of representing set for a node **t**, we can use basis rows or columns of all nodes in $\mathbf{S}_t$ or $\mathbf{sonsS}_t$ if these basises are already defined. This leads us to a *level-by-level* algorithm: start on the bottom of trees $\mathcal{T}_{\mathbf{I}}$ and $\mathcal{T}_{\mathbf{J}}$, calculate basis columns for each node of the $\mathcal{T}_{\mathbf{J}}$ on current level of the tree, calculate basis rows for each node of the $\mathcal{T}_{\mathbf{I}}$ on the same level and then go up one level. Under assumption of fixed predecessors part $\psi_{\mathbf{t}}^p$ of each representing set, we summarize *level-by-level* idea in Algorithm 4.

5.2. *Representing sets and iterations*

How to obtain predecessors part $\psi_{\mathbf{t}}^p$ of each representing set? Let us assume that all basis rows and columns are given. Obviously, union of basis columns of nodes from $\mathbf{S}_t \cup \mathbf{S_{pred(t)}}$ is a good representing set for block row $\mathbf{R_t}$. If nonleaf node **t** has sons $\{\mathbf{t_1}, \mathbf{t_2}\} = \mathbf{sons(t)}$, then predecessors part of representing set for node $\mathbf{t_1}$ can be set equal to just full representing set of **t**. This simple idea works perfectly with our previous approach:

(1) with a given predecessors part of representing sets, compute basis rows and columns;
(2) with given basises, update representing sets;
(3) shift newly calculated representing sets to produce predecessors part of representing sets; and
(4) go to step 1.

Obviously, we obtain an alternating iterative method of computing basises and representings. If we recalculate representings just as union of corresponding basises, size of each representing set will depend on number of nodes in $\mathbf{S}_t \cup \mathbf{S_{pred(t)}}$. This problem can be easily solved by additional resampling in top-to-bottom fashion, proposed in Algorithm 5.

DOI: 10.1002/nla

Algorithm 4 Computation of basis rows/columns and transfer matrices for each node of rows/columns cluster tree with given predecessors part of each representing set.

Require: Cluster trees $\mathcal{T}_\mathbf{I}$ and $\mathcal{T}_\mathbf{J}$, block row/column $\mathbf{R_t}$ ($\mathbf{C}_t$) and predecessors part $\psi_\mathbf{t}^p$ of representing set for each node $\mathbf{t}$, accuracy ε
Ensure: basis rows/columns $\hat{\chi}_\mathbf{t}$ and transfer matrix $M_\mathbf{t}$ for each node $t \in \mathcal{T}_\mathbf{I}$ or $\mathbf{t} \in \mathcal{T}_\mathbf{J}$
{Assume, that both $\mathcal{T}_\mathbf{I}$ and $\mathcal{T}_\mathbf{J}$ have **level_count** levels}
for $current\_level =$ **level_count to** 1 {from bottom of trees to top} **do**
  for $\mathbf{s} \in \mathcal{T}_\mathbf{J}$ {on the $current\_level$} **do**
    $\hat{\psi} = \psi_\mathbf{s}^p$ {initialize $\hat{\psi}$ with predecessors part}
    for $\mathbf{t} \in \mathbf{S}_s$ **do**
      if sons(t) is empty **then**
        $\hat{\psi} += \hat{\chi}_\mathbf{t}$ or $\hat{\psi} += \chi_\mathbf{t}$ (whether $\hat{\chi}_\mathbf{t}$ is defined or not)
      else {**sons(t)** is not empty, $\{\mathbf{t_1}, \mathbf{t_2}\} =$ **sons(t)**}
        $\hat{\psi} += \hat{\chi}_{\mathbf{t}_1} + \hat{\chi}_{\mathbf{t}_2}$
      end if
    end for
    if sons$s$ is empty **then**
      $\hat{R} = \hat{\psi}\mathbf{C_1}$, in other terms $\hat{R} = \hat{\psi} A \chi_\mathbf{s}$
    else {**sons**s is not empty, $\{\mathbf{s_1}, \mathbf{s_2}\} =$ **sons**s}
      $\hat{R} = \hat{\psi}\mathbf{C_1}\,(\hat{\chi}_{\mathbf{s}_1} + \hat{\chi}_{\mathbf{s}_2})$, in other terms $\hat{R} = \hat{\psi} A\,(\hat{\chi}_{\mathbf{s}_1} + \hat{\chi}_{\mathbf{s}_2})$
    end if
    $U, S, V = svd(\hat{R}, tol = \varepsilon)$; $M_\mathbf{s}$, $\hat{\chi}_\mathbf{s} = maxvol(V)$ {truncated svd + maxvol}
  end for
  for $\mathbf{t} \in \mathcal{T}_\mathbf{I}$ {on the $current\_level$} **do**
    $\hat{\psi} = \psi_\mathbf{t}^p$ {initialize $\hat{\psi}$ with predecessors part}
    $\hat{\psi} += \sum_{\mathbf{s} \in \mathbf{S}_t} \hat{\chi}_\mathbf{s}$
    if sons(t) is empty **then**
      $\hat{R} = \mathbf{R_t}\hat{\psi}$, in other terms $\hat{R} = \chi_\mathbf{t} A \hat{\psi}$
    else {**sons(t)** is not empty, $\{\mathbf{t_1}, \mathbf{t_2}\} =$ **sons(t)**}
      $\hat{R} = (\hat{\chi}_{\mathbf{t}_1} + \hat{\chi}_{\mathbf{t}_2})\,\mathbf{R_t}\hat{\psi}$, in other terms $\hat{R} = (\hat{\chi}_{\mathbf{t}_1} + \hat{\chi}_{\mathbf{t}_2})\,A\hat{\psi}$
    end if
    $U, S, V = svd(\hat{R}, tol = \varepsilon)$; $M_\mathbf{t}$, $\hat{\chi}_\mathbf{t} = maxvol(U)$ {truncated svd + maxvol}
  end for
end for

Algorithm 5 Computation of full representing sets with given basises.

Require: Cluster trees $\mathcal{T}_\mathbf{I}$ and $\mathcal{T}_\mathbf{J}$, basis rows/columns $\hat{\chi}_\mathbf{t}$ for each node $\mathbf{t}$
Ensure: Representing set $\psi_\mathbf{t}$ for each node $\mathbf{t}$
{Assume, that both $\mathcal{T}_\mathbf{I}$ and $\mathcal{T}_\mathbf{J}$ have **level_count** levels}
for $current\_level = 1$ **to level_count** {from bottom of trees to top} **do**
  for $\mathbf{t} \in \mathcal{T}_\mathbf{I}$ {on the $current\_level$} **do**
    {Initialize $\hat{\psi}$ as a resampled union of basis rows/columns from $\mathbf{S_{pred(t)}}$}
    if $current\_level$ is 1 {check if node $\mathbf{t}$ is the root node} **then**
      $\hat{\psi} = 0$
    else {$\mathbf{p}$ is the parent node of $\mathbf{t}$}
      $\hat{\psi} = \psi_\mathbf{p}$
    end if
    $\hat{\psi} += \sum_{\mathbf{s} \in \mathbf{S}_t} \hat{\chi}_\mathbf{s}$ {Add basis rows/columns from $\mathbf{S}_t$}
    $\psi_\mathbf{t} = maxvol(\hat{\chi}_\mathbf{t} A \hat{\psi})$ {resample $\hat{\psi}$ into $\psi_\mathbf{t}$}
  end for
end for

DOI: 10.1002/nla

5.3. Iterative multicharge method

Finally, the proposed MCBH algorithm is an iterative method and is summarized in Algorithm 6. Initialization is very easy: basis rows and columns are empty. Each iteration consists of three steps:

(1) Obtaining representing sets with Algorithm 5 for each cluster tree;
(2) Shifting representing sets into predecessors part of representing sets; and
(3) Computing basis rows/columns and transfer matrices with Algorithm 4.

Each iteration uses all the information of the previous iteration and increases accuracy. Numerical examples showed that 1-2 iterations are usually enough to obtain the desired accuracy.

Algorithm 6 Iterative MCBH algorithm.

```
Initialize basis rows/columns as empty
while did not get required accuracy do
  run Algorithm 5 for each cluster tree to get representing sets
  reevaluate representing sets into predecessors part of representing sets
  run Algorithm 4 to obtain new improved basis rows/columns and transfer matrices
end while
```

5.4. Complexity estimates

Because the form of $\mathcal{H}^2$-factorization presented in 3 is the same, as in [13], storage and error analysis holds the same as in [13]. Only thing to be analyzed is complexity. The main approach is formulated as step-by-step Algorithm 6, so we analyze each step separately.

Let us assume we have a matrix $A \in \mathbb{C}^{N\times M}$ that is already partitioned into blocks with row cluster tree $\mathcal{T}_\mathbf{I}$ and column cluster tree $\mathcal{T}_\mathbf{J}$. For simplicity, assume that all the ranks are equal to r. Then we have to do the following steps.

Step 1: Initialize basis rows/columns as empty. Obviously, it requires $O(|\mathcal{T}_\mathbf{I}| + |\mathcal{T}_\mathbf{J}|)$ operations ($|\mathcal{T}|$ is a number of nodes in cluster tree $\mathcal{T}$).

Step 2: Run Algorithm 5 for each cluster tree to get representing sets. Algorithm 5 works in a top-to-bottom manner with each node of clusters $\mathcal{T}_\mathbf{I}$ and $\mathcal{T}_\mathbf{J}$. On the zero iteration (we enumerate iterations from 0), basises are empty, so this step does not require any operations. Assume that for any other iteration basis size for each node of cluster trees $\mathcal{T}_\mathbf{I}$ and $\mathcal{T}_\mathbf{J}$ is equal to constant r. Also, assume any node $t \in \mathcal{T}_\mathbf{I} \cup \mathcal{T}_\mathbf{J}$ has C_F admissibly far nodes, and any leaf node $t \in \mathcal{T}_\mathbf{I} \cup \mathcal{T}_\mathbf{J}$ has C_C admissibly close nodes. Then, computation of representing sets with given basises for all nodes will require $O(maxvol((C_F + 1)r \times r)) * (|\mathcal{T}_\mathbf{I}| + |\mathcal{T}_\mathbf{J}|) = O(C_F r^3(|\mathcal{T}_\mathbf{I}| + |\mathcal{T}_\mathbf{J}|))$ operations.

Step 3: Reevaluate representing sets into the predecessors part of representing sets. This step is a simple move of representing sets from parent to child for each node (except leaves) of cluster trees $\mathcal{T}_\mathbf{I}$ and $\mathcal{T}_\mathbf{J}$. Obviously, it requires $O(|\mathcal{T}_\mathbf{I}| + |\mathcal{T}_\mathbf{J}|)$ operations.

Step 4: Run Algorithm 4 to obtain new improved basis rows/columns and transfer matrices. Assume each nonleaf node has K children, each leaf node corresponds to Kr rows or columns. Then, SVD reduction of Algorithm 4 works with $Kr \times (C_F Kr + r)$ matrices for each node $t \in \mathcal{T}_\mathbf{J}$ and with $Kr \times (C_F r + r)$ for each node $t \in \mathcal{T}_\mathbf{I}$. Because $K \geqslant 2$, we can estimate the complexity of all SVD reductions in Algorithm 4 as $O(|\mathcal{T}_\mathbf{J}|\, C_F K^3 r^3)$ operations. Each *maxvol* procedure works with a $Kr \times r$ matrix, so totally for all nodes, it requires $O((|\mathcal{T}_\mathbf{I}| + |\mathcal{T}_\mathbf{J}|)Kr^3)$ operations.

With further simplifications ($N = M$, complexity of SVD is of the same order as the complexity of *maxvol*), total complexity is about $O(N_{iters} C_F K^3 r^3\, |\mathcal{T}_\mathbf{J}|)$ operations. As we already assumed each leaf node has exactly Kr elements in it, so we have $|\mathcal{T}_\mathbf{J}| = O(MK^{-1}r^{-1}) = O(NK^{-1}r^{-1})$. So the complexity of the approximation of far interactions in the matrix A requires $O(C_F NK^2 r^2)$ operations. Because number of leaf nodes is $O(NK^{-1}r^{-1})$ and each leaf node requires $O(C_C K^2 r^2)$ to save close interactions, we have the following complexity on close interactions: $O(C_C NKr)$ operations. Totally,

DOI: 10.1002/nla

complexity of proposed MCBH Algorithm 6 is $O(C_F NK^2r^2 + C_C NKr)$ operations, where C_F is a mean number of admissibly far nodes for all nodes, C_C is a mean number of admissibly close nodes for all leaf nodes, N is a number of rows and columns, K is a number of children nodes for each node of cluster trees, and r is a mean basis size (rank) of each node.

Another interesting question is the number of matrix entries, required to compute approximation with Algorithm 6. Often, each matrix element is computed via complex function, which require many operations just for a single matrix element. Matrix elements are only used in close interactions and steps 2 and 4 of Algorithm 6.

Close interactions: Storage cost is the same, as number of operations ($O(C_C NKr)$).

Step 2: Run Algorithm 5 for each cluster tree to obtain representing sets. This operation requires computation of *maxvol* for $(C_F + 1)r \times r$ submatrices of A for each node. Step 4: Run Algorithm 4 to obtain new improved basis rows/columns and transfer matrices. Each iteration uses $Kr \times (C_F Kr + r)$ submatrix of A for each node $t \in \mathcal{T}_\mathbf{J}$ and $Kr \times (C_F r + r)$ submatrix of A for each node $t \in TI$.

So each iteration uses $O(C_F K^2 r^2 |\mathcal{T}_\mathbf{J}|)$ matrix entries. Assuming $|\mathcal{T}_\mathbf{J}| = O(NK^{-1}r^{-1})$, we summarize total number of matrix elements used: N_{iter} iterations of MCBH require $O(N_{iter} C_F NKr + C_C NKr)$ matrix values.

6. NUMERICAL EXPERIMENTS

This section contains two numerical examples. The first is to compute electric field potential, created by n given particles $X_1, \ldots, X_n$ with charges $q_1, \ldots, q_n$, in coordinates of particles $X_1, \ldots, X_n$. Particles are distributed randomly (uniform distribution) over a cube $[0, 1]^3$. This problem can be reformulated as computation of a matrix-vector product:

$$Aq = f, \tag{1}$$

where q is a vector of charges of particles, f is desired electric field potential, and matrix A is defined as follows:

$$A_{ij} = \begin{cases} \frac{1}{|X_i - X_j|}, & i \neq j \\ 0, & i = j \end{cases} \tag{2}$$

The second example is a solvation problem in the framework of polarized continuum model [24–26]. It arises in computer modeling of drugs: find the surface charge density σ on a given solvent excluded surface Ω, such that

$$\sigma(\mathbf{r}) = \frac{1-\varepsilon}{2\pi(1+\varepsilon)} \left(\sum_i \frac{Q_i((\mathbf{r} - \mathbf{R_i}) \cdot \mathbf{n})}{|\mathbf{r} - \mathbf{R_i}|^3} + \int_\Omega \frac{\sigma(\mathbf{r}')((\mathbf{r} - \mathbf{r}') \cdot \mathbf{n})}{|\mathbf{r} - \mathbf{r}'|^3} dS' \right), \tag{3}$$

where Q_i is a charge of i-th atom in molecule, $\mathbf{R_i}$ is a position vector of i-th atom in molecule, $\mathbf{r}$ is a radius vector to surface, $\mathbf{n}$ is a perpendicular from surface to solvent and ε is a relative permittivity. The surface is approximated by discrete elements with the Nyström method for the off-diagonal elements and the diagonal elements are computed from the identities:

$$\begin{aligned} &\int_\Omega \frac{((\mathbf{r} - \mathbf{r}') \cdot \mathbf{n}')}{|\mathbf{r} - \mathbf{r}'|^3} dS' = 2\pi, \\ &\int_{\Omega_\varepsilon} \frac{\sigma(\mathbf{r}')((\mathbf{r} - \mathbf{r}') \cdot \mathbf{n})}{|\mathbf{r} - \mathbf{r}'|^3} dS' \approx \sigma(\mathbf{r}) \left(2\pi - \int_{\Omega \setminus \Omega_\varepsilon} \frac{((\mathbf{r} - \mathbf{r}') \cdot \mathbf{n}')}{|\mathbf{r} - \mathbf{r}'|^3} dS' \right). \end{aligned} \tag{4}$$

So after discretizing and integrating over each discrete element, we obtain the following system:

$$Aq = BQ,$$

DOI: 10.1002/nla

where q is a vector of charges of discrete elements, and Q is a vector of charges of atoms in molecule. Matrices A and B are defined as follows:

$$A_{ij} = \begin{cases} \frac{(\varepsilon-1)}{4\pi(1+\varepsilon)} \frac{((\mathbf{r_i}-\mathbf{r_j})\cdot\mathbf{n_i})S_i}{|\mathbf{r_i}-\mathbf{r_j}|^3}, & i \neq j \\ \frac{\varepsilon}{1+\varepsilon} - \sum_{k\neq j} A_{kj}, & i = j \end{cases}, \tag{5}$$

$$B_{ij} = \frac{1-\varepsilon}{4\pi(1+\varepsilon)} \frac{((\mathbf{r_i}-\mathbf{R_j})\cdot\mathbf{n_i})S_i}{|\mathbf{r_i}-\mathbf{R_j}|^3},$$

where $\mathbf{r_i}$ is a radius vector to center of i-th discrete element, $\mathbf{n_i}$ is a perpendecular to i-th discrete element, and S_i is an area of i-th discrete element. In the following numerical examples, we study only the approximation of the matrix (5); we are not considering the problem of solution of linear systems with such matrices.

6.1. *Implementation remarks*

Multicharge Barnes–Hut Algorithm 6 is implemented in Python with the most computationally intensive parts reimplemented in Cython [http://cython.org]. For the basic linear algebra tasks, the

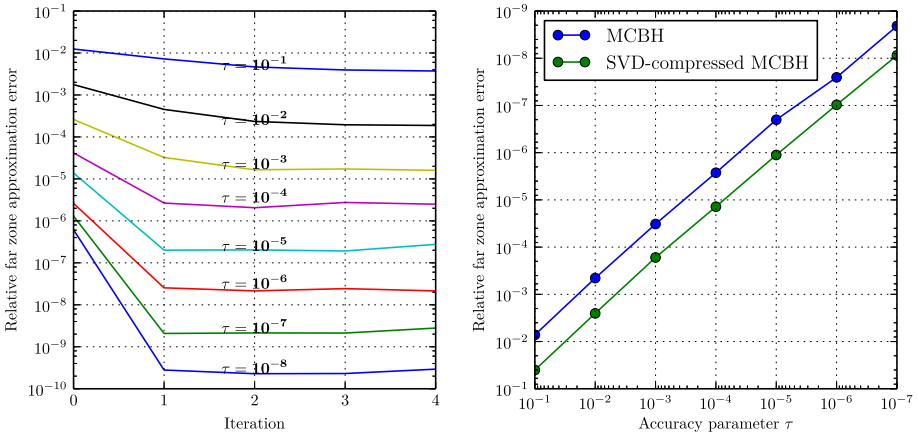

Figure 5. Dependence of approximation error on number of iterations (left) and accuracy parameter τ (right) for the electrostatics problem (2), $N = 100000$.

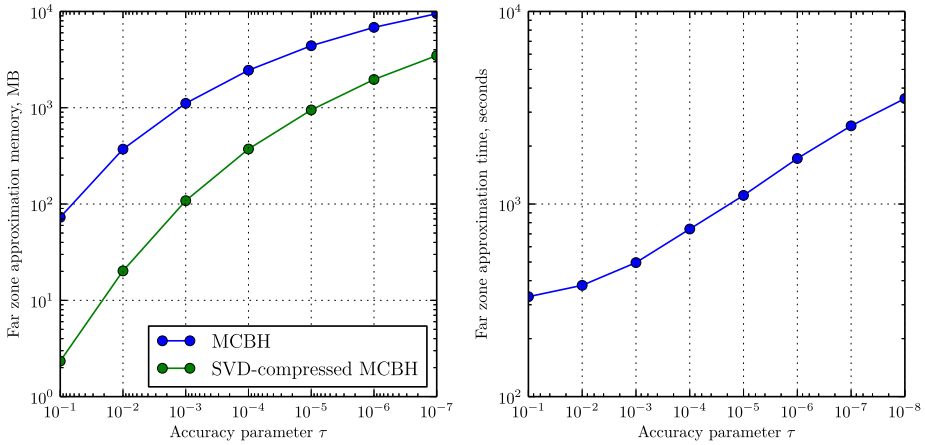

Figure 6. Dependence of approximation time (right) and memory (left) on accuracy parameter τ for the electrostatics problem, $N = 100000$.

DOI: 10.1002/nla

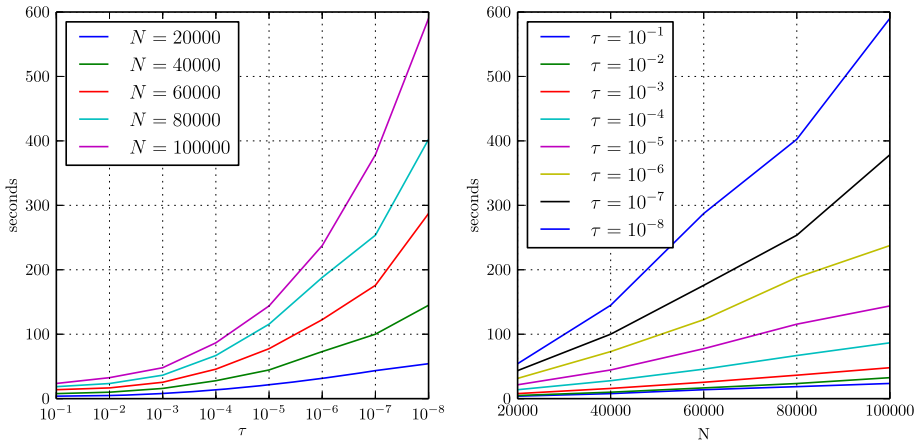

Figure 7. Dependence of maximum (100 tests) approximation time on accuracy parameter (left) and number of particles (right) for the electrostatics problem (2).

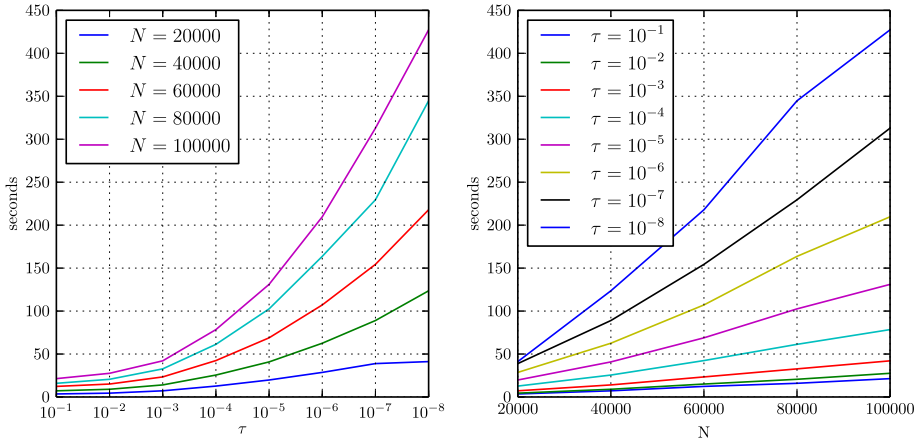

Figure 8. Dependence of mean (100 tests) approximation time on accuracy parameter (left) and number of particles (right) for the electrostatics problem (2).

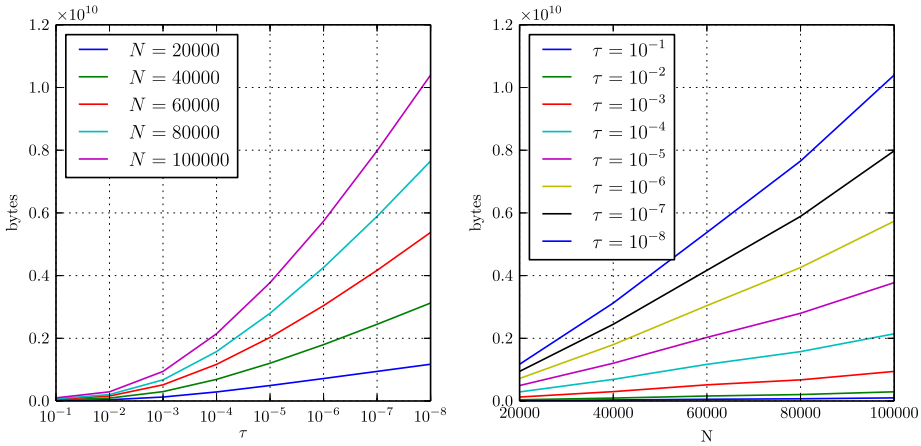

Figure 9. Dependence of maximum (100 tests) approximation memory on accuracy parameter (left) and number of particles (right) for the electrostatics problem (2).

DOI: 10.1002/nla

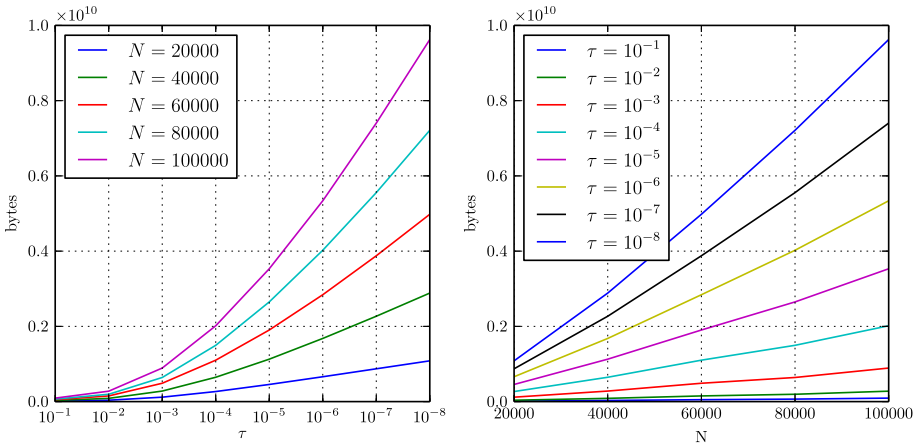

Figure 10. Dependence of mean (100 tests) approximation memory on accuracy parameter (left) and number of particles (right) for the electrostatics problem (2).

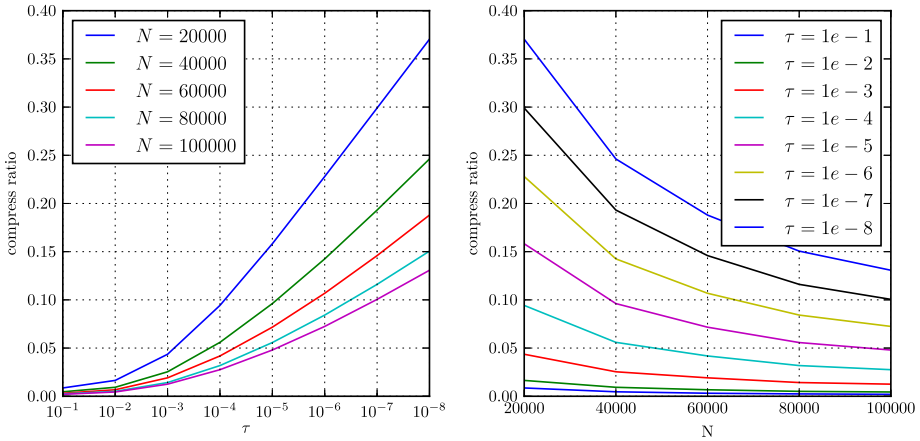

Figure 11. Dependence of worst (100 tests) compress ratio on accuracy parameter (left) and number of particles (right) for the electrostatics problem (2).

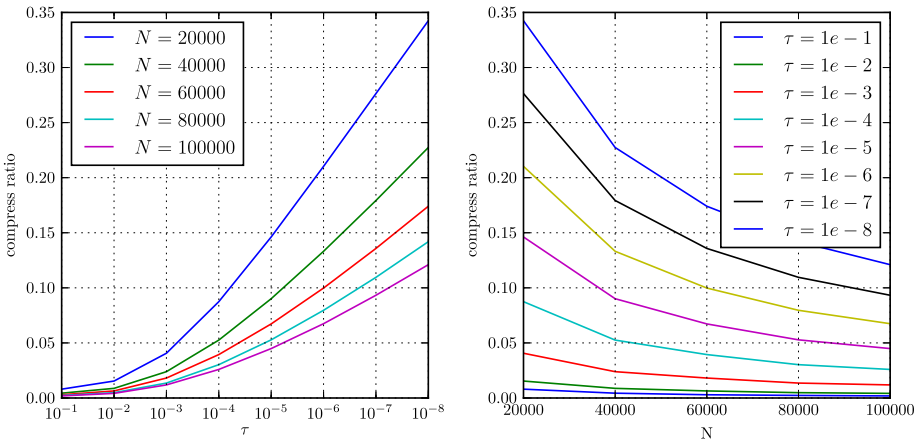

Figure 12. Dependence of mean (100 tests) compress ratio on accuracy parameter (left) and number of particles (right) for the electrostatics problem (2).

DOI: 10.1002/nla

MKL library is used. The code can be obtained at http://bitbucket.org/muxas/h2tools. To build the hierarchical tree, we used recursive inertial bisection with admissibility parameter $\eta = 0$ and *block_size* = 50, where *block_size* is a constant, such that we divide cluster into subclusters only if it has more than *block_size* elements. For Figure 5, Figure 6, Figure 13 and Figure 14 each SVD reduction of Algorithm 6 is computed with the relative tolerance $\frac{\tau}{level\_count}$. We used it instead of τ to align relative error, reached in solvation problem on 1 iteration of our algorithm

(see Figure 13). All the tests, except tests on number of iterations, were performed with 1 iteration of MCBH. In addition, we tested the optimization of the ranks by the SVD recompression of the $\mathcal{H}^2$-matrices [9] with different tolerances. As a starting point for the SVD recompression we used the MCBH approximation with the accuracy parameter $\tau = \frac{10^{-8}}{level\_count}$ and 1 additional iteration. Memory requirement in the figures below is the memory to store both *transfer matrices* and *interaction matrices*. The error in figures is relative error of only far field approximation in the spectral norm. It is computed with the help of Propack package [http://sun.stanford.edu/~rmunk/PROPACK/].

Tests were performed on a server with 2 Intel(R) Xeon(R) CPU E5504 @ 2.00GHz processors with 72GB of random-access memory. However, only 2 threads were used (this is default number of threads for MKL). Python, Cython, and MKL are from the Enthought Python distribution

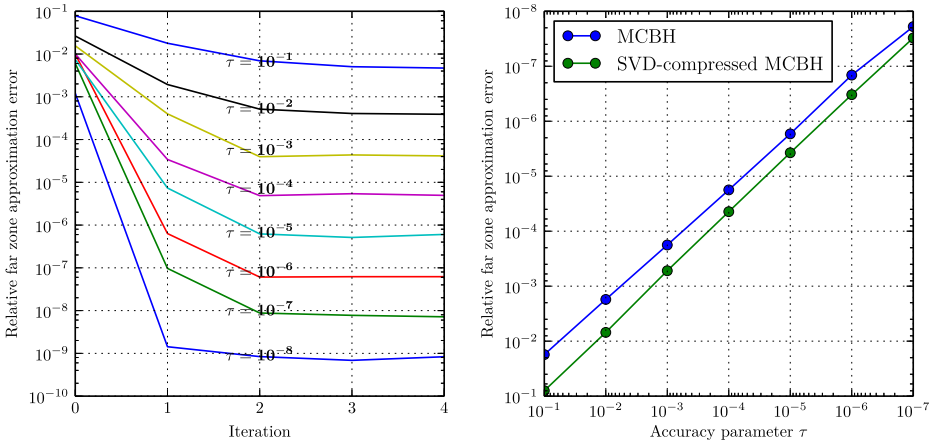

Figure 13. Dependence of approximation error on number of iterations (left) and accuracy parameter τ (right) for the boundary integral Eq. 3, surface consists of 222,762 discrete elements.

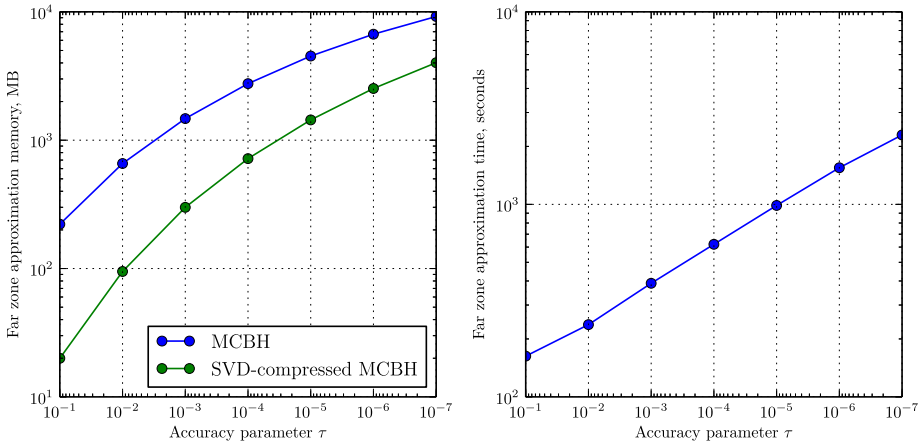

Figure 14. Dependence of approximation time (right) and memory (left) on accuracy parameter τ for the boundary integral operator (3), surface consists of 222,762 discrete elements.

DOI: 10.1002/nla

(7.3-1 ,64-bit) [https://www.enthought.com/]. Python version is '2.7.3'; Cython version is '0.16'; MKL version is '10.3-1'.

6.2. Experiment with particles and interaction matrix (2)

Figure 5 shows dependence of relative error on number of iterations and accuracy parameter. Figure 6 shows dependence of approximation time and memory on accuracy parameter. To show dependence of approximation time and memory on number of particles and accuracy parameter, we measured maximum and mean values for 100 different random particle configurations. Figures 7, 8, 9, 10, 11, and 12 tests used τ accuracy parameter instead of mentioned $\frac{\tau}{level\_count}$. Figures 7 and Figure 8 show dependence of maximum and mean approximation time on accuracy parameter and number of particles. Figures 9 and 10 show dependence of maximum and mean approximation memory on accuracy parameter and number of particles. Figures 11 and 12 show compress ratio (approximation size vs matrix size).

6.3. Boundary integral equation experiment

In this experiment, the test surface consists of 222,762 triangles. Figure 13 shows dependence of relative error on number of iterations and accuracy parameter. Figure 14 shows dependence of approximation time and memory on accuracy parameter.

Table I. Comparison of time and error of MCBH and ACAGEO for electrostatics problem (2).

		MCBH			ACAGEO		
τ	block_size	iters	time	error	points	time	error
10^{-2}	25	0	9.7	$2.2 * 10^{-2}$	8	10.7	$1.8 * 10^{-2}$
					27	29.5	$2.2 * 10^{-2}$
10^{-3}	25	0	15.5	$3.2 * 10^{-3}$	27	30.5	$3.8 * 10^{-3}$
10^{-4}	25	0	26.8	$4.3 * 10^{-4}$	27	32.3	$6.3 * 10^{-4}$
10^{-5}	25	0	46.3	$1.2 * 10^{-4}$	27	35	$2.4 * 10^{-4}$
		1	148	$3 * 10^{-5}$	64	217.4	$1.8 * 10^{-4}$
10^{-6}	25	1	262	$3.3 * 10^{-6}$	27	33.4	$1.8 * 10^{-4}$
					64	221	$4 * 10^{-5}$

MCBH, multicharge Barnes-Hut.

Table II. Comparison of time and error of MCBH and ACAGEO for solvation problem.

		MCBH			ACAGEO		
τ	block_size	iters	time	error	points	time	error
10^{-2}	25	0	11.5	$8.4 * 10^{-2}$	8	13.2	$2.6 * 10^{-1}$
	25	1	29	$6.4 * 10^{-2}$	27	35.7	$9.6 * 10^{-2}$
10^{-3}	25	0	18.6	$2.4 * 10^{-2}$	8	13.3	$2.6 * 10^{-1}$
	25	1	50	$1.1 * 10^{-2}$	27	37	$3.3 * 10^{-2}$
10^{-4}	25	0	30.8	$1.2 * 10^{-2}$	8	13.2	$2.6 * 10^{-1}$
	25	1	101.9	$1.6 * 10^{-3}$	27	38.2	$1.7 * 10^{-2}$
	25	2	174.9	$1.6 * 10^{-3}$	64	163	$1.5 * 10^{-2}$
10^{-5}	25	0	45.9	$9 * 10^{-3}$	27	38.6	$1.4 * 10^{-2}$
	25	1	181.6	$2.1 * 10^{-4}$	64	166.3	$7.6 * 10^{-3}$
	25	2	321.8	$2 * 10^{-4}$	125	609	$8.6 * 10^{-3}$

MCBH, multicharge Barnes-Hut.

DOI: 10.1002/nla

7. COMPARISON WITH OTHER METHODS

For the comparison, we have chosen the methods describe in the paper [13], namely, ACAMERGE and ACAGEO.

We have found that ACAMERGE method is equivalent to the zero iteration of proposed MCBH. As it can be seen from the numerical experiments, this iteration can have limited accuracy. Impact of the number of iterations on relative accuracy for different problems is shown in figures 5 and 13. So ACAMERGE cannot get relative error of far field approximation lower than 10^{-3} at least for the double-layer-type problem considered earlier.

ACAGEO method is based on the Chebyshev grid introduced in the bounding box of a given cluster node. Because the original code from [13] is not available, for a fair comparison, we have reimplemented the ACAGEO method. Tables I and II correspond to application of MCBH and ACAGEO to approximation of matrices for two problems from Section 6. Notation used is the following: τ – accuracy parameter;

block_size – maximum size of leaf node (number of elements in it);
iters – number of iterations for MCBH method;
time – factorization time in seconds;
error – relative spectral error of approximation, measured only by far field; and
points – number of Chebyshev points for ACAGEO method (number of potential basis elements for each node of cluster trees).

8. CONCLUSIONS AND FUTURE WORK

The proposed algorithm is robust in the sense that high approximation accuracies are achievable using at most two additional iterations. It also does not require the construction of a priori representing sets. An important question that needs to be solved is the existence of a good MCBH-type approximation, provided that a good $\mathcal{H}^2$-approximation exists. A preliminary study shows that it is possible to derive the existence of a quasi-optimal approximation in the spirit of the recent paper [27].

Computational speed and memory requirements of the algorithm can be improved in several ways. The algorithm can be readily parallelized because it has the usual fast multipole method structure. Also, in the case when the elements of a matrix can be computed in a cheap way, the interaction matrices can be computed online, and the memory consumption becomes much smaller. For the case when the computation of a single element of a matrix is expensive (for example, in the Galerkin method for the solution of a boundary integral equation), the application of the $\mathcal{H}^2$-recompression algorithm may give a noticeable reduction in the storage.

The code is available as a part of the h2tools package http://bitbucket.org/muxas/h2tools, and we plan to extended it with different compression algorithms and solvers. It will be interesting to apply the new method to different problems, for example, to the problem of vortex ring dynamics [28].

ACKNOWLEDGEMENT

This work was supported by Russian Science Foundation Grant 14-11-00659.

REFERENCES

1. Bebendorf M. Approximation of boundary element matrices. *Numerical Mathematics* 2000; **86**(4):565–589. DOI: 10.1007/PL00005410.
2. Tyrtyshnikov EE. Incomplete cross approximation in the mosaic–skeleton method. *Computing* 2000; **64**(4):367–380. DOI: 10.1007/s006070070031.
3. Tyrtyshnikov EE. Mosaic-skeleton approximations. *Calcolo* 1996; **33**(1):47–57. DOI: 10.1007/BF02575706.
4. Hackbusch W. A sparse matrix arithmetic based on H-matrices. Part I: introduction to H-matrices. *Computing* 1999; **62**(2):89–108. DOI: 10.1007/s006070050015.
5. Hackbusch W, Khoromskij BN. A sparse H-matrix arithmetic. Part II: application to multi-dimensional problems. *Computing* 2000; **64**(1):21–47. DOI: 10.1007/PL00021408.

6. Hackbusch W, Khoromskij B N, Kriemann R. Hierarchical matrices based on a weak admissibility criterion. *Computing* 2004; **73**(3):207–243.
7. Hackbusch W, Khoromskij B, Sauter SA. *On $\mathcal{H}^2$-matrices*. Springer, 2000. DOI: 10.1007/978-3-642-59709-1_2.
8. Greengard L, Rokhlin V. A fast algorithm for particle simulations. *Journal of Computational Physics* 1987; **73**(2):325–348. DOI: 10.1016/0021-9991(87)90140-9.
9. Börm S. Efficient numerical methods for non-local operators: $\mathcal{H}^2$-matrix compression, algorithms and analysis. *European Mathematical Society*, posted on 2010. DOI: 10.4171/091.
10. Goreinov S A, Tyrtyshnikov E E, Zamarashkin N L. A theory of pseudo–skeleton approximations. *Linear Algebra Appl* 1997; **261**:1–21. DOI: 10.1016/S0024-3795(96)00301-1.
11. Goreinov S A, Tyrtyshnikov E E. The maximal-volume concept in approximation by low-rank matrices. *Contemporary Mathematics* 2001; **208**:47–51.
12. Hackbusch W, Börm S. $\mathcal{H}^2$-matrix approximation of integral operators by interpolation. *Applied Numerical Mathematics* 2002; **43**(1):129–143.
13. Bebendorf M, Venn R. Constructing nested bases approximations from the entries of non-local operators. *Numerical Mathematics* 2012; **121**(4):609–635. DOI: 10.1007/s00211-012-0449-9.
14. Barnes J, Hut P. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature* 1986; **324**:446–449. DOI: 10.1038/324446a0.
15. Oseledets IV, Tyrtyshnikov EE. TT-cross approximation for multidimensional arrays. *Linear Algebra and its Applications* 2010; **432**(1):70–88. DOI: 10.1016/j.laa.2009.07.024.
16. Ying L, Biros G, Zorin D. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics* 2004; **196**(2):591–626. DOI: 10.1016/j.jcp.2003.11.021.
17. Lage C, Schwab C. Wavelet Galerkin algorithms for boundary integral equations. *SIAM Journal on Scientific Computing* 1999; **20**(6):2195–2222.
18. Amaratunga K, Williams J R, Qian S, Weiss J. Wavelet–Galerkin solutions for one-dimensional partial differential equations. *International Journal for Numerical Methods in Engineering* 1994; **37**(16):2703–2716.
19. Sheng Z, Dewilde P, Chandrasekaran S. Algorithms to Solve Hierarchically Semi-separable Systems. *System Theory, the Schur Algorithm and Multidimensional Analysis*, Springer, 2007; 255–294.
20. Martinsson PG. A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix. *SIAM Journal on Matrix Analysis and Applications* 2011; **32**(4):1251–1274. DOI: 10.1137/100786617.
21. Goreinov SA, Zamarashkin NL, Tyrtyshnikov EE. Pseudo–skeleton approximations by matrices of maximum volume. *Mathematical Notes* 1997; **62**(4):515–519. DOI: 10.1007/BF02358985.
22. Goreinov SA, Oseledets IV, Savostyanov DV, Tyrtyshnikov EE, Zamarashkin NL. How to find a good submatrix,. In *Matrix Methods: Theory, Algorithms, Applications*, Olshevsky V, Tyrtyshnikov E (eds). World Scientific: Hackensack, NY, 2010.
23. Ho K L, Greengard L. A fast direct solver for structured linear systems by recursive skeletonization. *SIAM Journal on Scientific Computing* 2012; **34**(5):A2507–A2532. DOI: 10.1137/120866683.
24. Tomasi J, Persico M. Molecular interactions in solution: an overview of method based on continuous distributions of the solvent. *Chem. Rev.* 1994; **94**(7):2027–2094. DOI: 10.1021/cr00031a013.
25. Cramer C, Truhlar D. Implicit solvation models: equilibria, structure, spectra, and dynamics. *Chemical Reviews* 1999; **99**(8):2161–2200. DOI: 10.1021/cr960149m.
26. Totrov M, Abagyan R. Rapid boundary element solvation electrostatics calculations in folding simulations: successful folding of a 23-residue peptide. *Journal of Peptide Science* 2001; **60**(2):124–133. DOI: 10.1002/1097-0282(2001)60:2<124::AID-BIP1008>3.0.CO;2-S.
27. Savostyanov DV. Quasioptimality of maximum–volume cross interpolation of tensors. *Linear Algebra and its Applications* 2014; **458**:217–244.
28. Stavtsev SL. Application of the method of incomplete cross approximation to a nonstationary problem of vortex rings dynamics. *Russian Journal of Numerical Analysis and Mathematical Modelling* 2012; **27**(3):303–320. DOI: 10.1515/rnam-2012-0017.