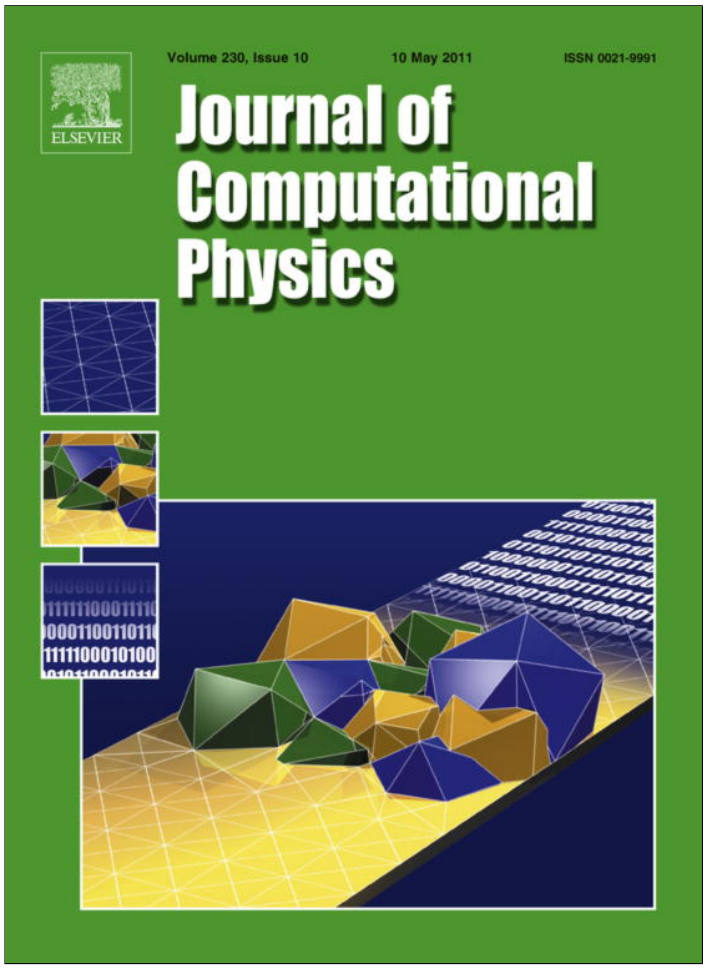

Journal of Computational Physics 230 (2011) 4071–4087

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Fast construction of hierarchical matrix representation from matrix–vector multiplication

Lin Lin [a,*], Jianfeng Lu [b], Lexing Ying [c]

[a] Program in Applied and Computational Mathematics, Princeton University, Princeton, NJ 08544, United States
[b] Department of Mathematics, Courant Institute of Mathematical Sciences, New York University, 251 Mercer St., New York, NY 10012, United States
[c] Department of Mathematics and ICES, University of Texas at Austin, 1 University Station, C1200, Austin, TX 78712, United States

ARTICLE INFO

Article history:
Received 31 December 2009
Received in revised form 28 January 2011
Accepted 22 February 2011

Keywords:
Fast algorithm
Hierarchical matrix construction
Randomized singular value decomposition
Matrix–vector multiplication
Elliptic operator
Green's function

ABSTRACT

We develop a hierarchical matrix construction algorithm using matrix–vector multiplications, based on the randomized singular value decomposition of low-rank matrices. The algorithm uses $\mathcal{O}(\log n)$ applications of the matrix on structured random test vectors and $\mathcal{O}(n \log n)$ extra computational cost, where n is the dimension of the unknown matrix. Numerical examples on constructing Green's functions for elliptic operators in two dimensions show efficiency and accuracy of the proposed algorithm.

1. Introduction

In this work, we consider the following problem: Assume that an unknown symmetric matrix G has the structure of a hierarchical matrix ($\mathcal{H}$-matrix) [1–3], that is, certain off-diagonal blocks of G are low-rank or approximately low-rank (see the definitions in Sections 1.3 and 2.2). The task is to construct G efficiently only from a "black box" matrix–vector multiplication subroutine (which shall be referred to as matvec in the following). In a slightly more general setting when G is not symmetric, the task is to construct G from "black box" matrix–vector multiplication subroutines of both G and G^T. In this paper, we focus on the case of a symmetric matrix G. The proposed algorithm can be extended to the non-symmetric case in a straightforward way.

1.1. Motivation and applications

Our motivation is mainly the situation that G is given as the Green's function of an elliptic equation. In this case, it is proved that G is an $\mathcal{H}$-matrix under mild regularity assumptions [4]. For elliptic equations, methods like preconditioned conjugate gradient, geometric and algebraic multigrid methods, sparse direct methods provide application of the matrix G on vectors. The algorithm proposed in this work then provides an efficient way to construct the matrix G explicitly in the $\mathcal{H}$-matrix form.

Once we obtain the matrix G as an $\mathcal{H}$-matrix, it is possible to apply G on vectors efficiently, since the application of an $\mathcal{H}$-matrix on a vector is linear scaling. Of course, for elliptic equations, it might be more efficient to use available fast solvers

* Corresponding author.
E-mail addresses: linlin@math.princeton.edu (L. Lin), jianfeng@cims.nyu.edu (J. Lu), lexing@math.utexas.edu (L. Ying).

0021-9991/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2011.02.033

directly to solve the equation, especially if only a few right hand sides are to be solved. However, sometimes, it would be advantageous to obtain G since it is then possible to further compress G according to the structure of the data (the vectors that G will be acting on), for example as in numerical homogenization [5]. Another scenario is that the data has special structure like sparsity in the choice of basis, the application of the resulting compressed matrix will be more efficient than the "black box" elliptic solver.

Let us remark that, in the case of elliptic equations, it is also possible to use the $\mathcal{H}$-matrix algebra to invert the direct matrix (which is an $\mathcal{H}$-matrix in *e.g.* finite element discretization). Our method, on the other hand, provides an efficient alternative algorithm when a fast matrix–vector multiplication is readily available, and is able to compute the inverse of an $\mathcal{H}$-matrix of dimension $n \times n$ with $\mathcal{O}(\log n)$ matrix–vector multiplications. We also remark that the pre-constant in front of the $\mathcal{O}(\log n)$ scaling can be large, and this may hinder the application of the current version of the algorithm in many scenarios. However, from a computational point of view, what is probably more attractive is that our algorithm facilitates a parallelized construction of the $\mathcal{H}$-matrix, while the direct inversion has a sequential nature [2].

As another motivation, the purpose of the algorithm is to recover the matrix via a "black box" matrix–vector multiplication subroutine. A general question of this kind will be that under which assumptions of the matrix, one can recover the matrix efficiently by matrix–vector multiplications. If the unknown matrix is low-rank, the recently developed randomized singular value decomposition algorithms [6–8] provide an efficient way to obtain the low-rank approximation through application of the matrix on random vectors. Low-rank matrices play an important role in many applications. However, the assumption is too strong in many cases that the whole matrix is low-rank. Since the class of $\mathcal{H}$-matrices is a natural generalization of the one of low-rank matrices, the proposed algorithm can be viewed as a further step in this direction.

1.2. Randomized singular value decomposition algorithm

A repeatedly leveraged tool in the proposed algorithm is the randomized singular value decomposition algorithm for computing a low rank approximation of a given numerically low-rank matrix. This has been an active research topic in the past several years with vast literature. For the purpose of this work, we have adopted the algorithm developed in [7], although other variants of this algorithm with similar ideas can also be used here. For a given matrix A that is numerically low-rank, this algorithm goes as following to compute a rank-r factorization.

Algorithm 1. Construct a low-rank approximation $A \approx U_1 M U_2^{\mathrm{T}}$ for rank r

1: Choose a Gaussian random matrix $R_1 \in \mathbb{R}^{n\times(r+c)}$ where c is a small constant;
2: Form $A\,R_1$ and apply SVD to $A\,R_1$. The first r left singular vectors give U_1;
3: Choose a Gaussian random matrix $R_2 \in \mathbb{R}^{n\times(r+c)}$;
4: Form $R_2^{\mathrm{T}}A$ and apply SVD to $A^{\mathrm{T}}R_2$. The first r left singular vectors give U_2;
5: $M = (R_2^{\mathrm{T}}U_1)^{\dagger}[R_2^{\mathrm{T}}(AR_1)](U_2^{\mathrm{T}}R_1)^{\dagger}$, where $B^{\dagger}$ denotes the Moore–Penrose pseudoinverse of matrix B [9, pp. 257–258].

The accuracy of this algorithm and its variants has been studied thoroughly by several groups. If the matrix 2-norm is used to measure the error, it is well-known that the best rank-r approximation is provided by the singular value decomposition (SVD). When the singular values of A decay rapidly, it has been shown that Algorithm 1 results in almost optimal factorizations with an overwhelming probability [6]. As Algorithm 1 is to be used frequently in our algorithm, we analyze briefly its complexity step by step. The generation of random numbers is quite efficient, therefore in practice one may ignore the cost of steps 1 and 3. Step 2 takes $(r+c)$ matvec of matrix A and $\mathcal{O}(n(r+c)^2)$ steps for applying the SVD algorithms on an $n \times (r+c)$ matrix. The cost of step 4 is the same as the one of step 2. Step 5 involves the computation of $R_2^{\mathrm{T}}(AR_1)$, which takes $\mathcal{O}(n(r+c)^2)$ steps as we have already computed AR_1 in step 2. Once $R_2^{\mathrm{T}}(AR_1)$ is ready, the computation of M takes additional $\mathcal{O}((r+c)^3)$ steps. Therefore, the total complexity of Algorithm 1 is $\mathcal{O}(r+c)$ matvecs plus $\mathcal{O}(n(r+c)^2)$ extra steps.

1.3. Top-down construction of $\mathcal{H}$-matrix

We illustrate the core idea of our algorithm using a simple one-dimensional example. The algorithm of constructing a hierarchical matrix G is a top-down pass. We assume throughout the article that G is symmetric.

For clarity, we will first consider a one dimension example. The details of the algorithm in two dimensions will be given in Section 2. We assume that a symmetric matrix G has a hierarchical low-rank structure corresponding to a hierarchical dyadic decomposition of the domain. The matrix G is of dimension $n \times n$ with $n = 2^{L_M}$ for an integer L_M. Denote the set for all indices as $\mathcal{I}_{0;1}$, where the former subscript indicates the level and the latter is the index for blocks in each level. At the first level, the set is partitioned into $\mathcal{I}_{1;1}$ and $\mathcal{I}_{1;2}$, with the assumption that $G(\mathcal{I}_{1;1},\mathcal{I}_{1;2})$ and $G(\mathcal{I}_{1;2},\mathcal{I}_{1;1})$ are numerically low-rank, say of rank r for a prescribed error tolerance ε. At level l, each block $\mathcal{I}_{l-1;i}$ on the above level is dyadically decomposed into two blocks $\mathcal{I}_{l;2i-1}$ and $\mathcal{I}_{l;2i}$ with the assumption that $G(\mathcal{I}_{l;2i-1},\mathcal{I}_{l;2i})$ and $G(\mathcal{I}_{l;2i},\mathcal{I}_{l;2i-1})$ are also numerically low-rank (with the same rank r for the tolerance ε). Clearly, at level l, we have in total 2^l off-diagonal low-rank blocks. We stop at level L_M, for which the block $\mathcal{I}_{L_M,i}$ only has one index $\{i\}$. For simplicity of notation, we will abbreviate $G(\mathcal{I}_{l;i},\mathcal{I}_{l;j})$ by $G_{l;ij}$. We remark that the

assumption that off-diagonal blocks are low-rank matrices may not hold for general elliptic operators in higher dimensions. However, this assumption simplifies the introduction of the concept of our algorithm. More realistic case will be discussed in detail in Sections 2.3 and 2.4.

The overarching strategy of our approach is to peel off the off-diagonal blocks level by level and simultaneously construct their low-rank approximations. On the first level, $G_{1;12}$ is numerically low-rank. In order to use the randomized SVD algorithm for $G_{1;12}$, we need to know the product of $G_{1;12}$ and also $G_{1;12}^{\mathrm{T}} = G_{1;21}$ with a collection of random vectors. This can be done by observing that

$$\begin{pmatrix} G_{1;11} & G_{1;12} \\ G_{1;21} & G_{1;22} \end{pmatrix} \begin{pmatrix} R_{1;1} \\ 0 \end{pmatrix} = \begin{pmatrix} G_{1;11}R_{1;1} \\ G_{1;21}R_{1;1} \end{pmatrix}, \tag{1}$$

$$\begin{pmatrix} G_{1;11} & G_{1;12} \\ G_{1;21} & G_{1;22} \end{pmatrix} \begin{pmatrix} 0 \\ R_{1;2} \end{pmatrix} = \begin{pmatrix} G_{1;12}R_{1;2} \\ G_{1;22}R_{1;2} \end{pmatrix}, \tag{2}$$

where $R_{1;1}$ and $R_{1;2}$ are random matrices of dimension $n/2 \times (r+c)$. We obtain $(G_{1;21}R_{1;1})^{\mathrm{T}} = R_{1;1}^{\mathrm{T}}G_{1;12}$ by restricting the right hand side of Eq. (1) to $\mathcal{I}_{1;2}$ and obtain $G_{1;12}R_{1;2}$ by restricting the right hand side of Eq. (2) to $\mathcal{I}_{1;1}$, respectively. The low-rank approximation using Algorithm 1 results in

$$G_{1;12} \approx \widehat{G}_{1;12} = U_{1;12}M_{1;12}U_{1;21}^{\mathrm{T}}. \tag{3}$$

$U_{1;12}$ and $U_{1;21}$ are $n/2 \times r$ matrices and $M_{1;12}$ is an $r \times r$ matrix. Due to the fact that G is symmetric, a low-rank approximation of $G_{1;21}$ is obtained as the transpose of $G_{1;12}$.

Now on the second level, the matrix G has the form

$$\begin{pmatrix} G_{2;11} & G_{2;12} & & \\ G_{2;21} & G_{2;22} & G_{1;12} & \\ & G_{1;21} & G_{2;33} & G_{2;34} \\ & & G_{2;43} & G_{2;44} \end{pmatrix}.$$

The submatrices $G_{2;12}$, $G_{2;21}$, $G_{2;34}$, and $G_{2;43}$ are numerically low-rank, to obtain their low-rank approximations by the randomized SVD algorithm. Similar to the first level, we could apply G on random matrices of the form like $(R_{2;1},0,0,0)^{\mathrm{T}}$. This will require $4(r+c)$ number of matrix–vector multiplications. However, this is not optimal: Since we already know the interaction between $\mathcal{I}_{1;1}$ and $\mathcal{I}_{1;2}$, we could combine the calculations together to reduce the number of matrix–vector multiplications needed. Observe that

$$\begin{pmatrix} G_{2;11} & G_{2;12} & & \\ G_{2;21} & G_{2;22} & G_{1;12} & \\ & G_{1;21} & G_{2;33} & G_{2;34} \\ & & G_{2;43} & G_{2;44} \end{pmatrix} \begin{pmatrix} R_{2;1} \\ 0 \\ R_{2;3} \\ 0 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} G_{2;11}R_{2;1} \\ G_{2;21}R_{2;1} \end{pmatrix} + G_{1;12}\begin{pmatrix} R_{2;3} \\ 0 \end{pmatrix} \\ \begin{pmatrix} G_{2;33}R_{2;3} \\ G_{2;43}R_{2;3} \end{pmatrix} + G_{1;21}\begin{pmatrix} R_{2;1} \\ 0 \end{pmatrix} \end{pmatrix}. \tag{4}$$

Denote

$$\widehat{G}^{(1)} = \begin{pmatrix} 0 & \widehat{G}_{1;12} \\ \widehat{G}_{1;21} & 0 \end{pmatrix} \tag{5}$$

with $\widehat{G}_{1;12}$ and $\widehat{G}_{1;21}$ the low-rank approximations we constructed on the first level, then

$$\widehat{G}^{(1)} \begin{pmatrix} R_{2;1} \\ 0 \\ R_{2;3} \\ 0 \end{pmatrix} = \begin{pmatrix} \widehat{G}_{1;12}\begin{pmatrix} R_{2;3} \\ 0 \end{pmatrix} \\ \widehat{G}_{1;21}\begin{pmatrix} R_{2;1} \\ 0 \end{pmatrix} \end{pmatrix}. \tag{6}$$

Therefore,

$$\left(G - \widehat{G}^{(1)}\right) \begin{pmatrix} R_{2;1} \\ 0 \\ R_{2;3} \\ 0 \end{pmatrix} \approx \begin{pmatrix} G_{2;11}R_{2;1} \\ G_{2;21}R_{2;1} \\ G_{2;33}R_{2;3} \\ G_{2;43}R_{2;3} \end{pmatrix}, \tag{7}$$

so that we simultaneously obtain $(G_{2;21}R_{2;1})^{\mathrm{T}} = R_{2;1}^{\mathrm{T}}G_{2;12}$ and $(G_{2;43}R_{2;3})^{\mathrm{T}} = R_{2;3}^{\mathrm{T}}G_{2;34}$. Similarly, applying G on $(0,R_{2;2},0,R_{2;4})^{\mathrm{T}}$ provides $G_{2;12}R_{2;2}$ and $G_{2;34}R_{2;4}$. We can then obtain the following low-rank approximations by invoking Algorithm 1.

$$\begin{aligned} G_{2;12} &\approx \widehat{G}_{2;12} = U_{2;12}M_{2;12}U_{2;21}^{\mathrm{T}}, \\ G_{2;34} &\approx \widehat{G}_{2;34} = U_{2;34}M_{2;34}U_{2;43}^{\mathrm{T}}. \end{aligned} \tag{8}$$

The low-rank approximations of $G_{2;21}$ and $G_{2;43}$ are again given by the transposes of the above formulas.

Similarly, on the third level, the matrix G has the form

$$\begin{pmatrix} G_{3;11} & G_{3;12} & & & & & & \\ G_{3;21} & G_{3;22} & G_{2;12} & & & & & \\ & & G_{3;33} & G_{3;34} & & & & \\ & G_{2;21} & G_{3;43} & G_{3;44} & & G_{1;12} & & \\ & & & & G_{3;55} & G_{3;56} & & \\ & & & & G_{3;65} & G_{3;66} & G_{2;34} & \\ & & G_{1;21} & & & & G_{3;77} & G_{3;78} \\ & & & & & G_{2;43} & G_{3;87} & G_{3;88} \end{pmatrix} \tag{9}$$

and define

$$\widehat{G}^{(2)} = \begin{pmatrix} 0 & \widehat{G}_{2;12} & & \\ \widehat{G}_{2;21} & 0 & & \\ & & 0 & \widehat{G}_{2;34} \\ & & \widehat{G}_{2;43} & 0 \end{pmatrix}. \tag{10}$$

We could simultaneously obtain the product of $G_{3;12}$, $G_{3;34}$, $G_{3;56}$ and $G_{3;78}$ with random vectors by applying the matrix G with random vectors of the form

$$\left(R_{3;1}^{\mathrm{T}}, 0, R_{3;3}^{\mathrm{T}}, 0, R_{3;5}^{\mathrm{T}}, 0, R_{3;7}^{\mathrm{T}}, 0\right)^{\mathrm{T}},$$

then subtract the product of $\widehat{G}^{(1)} + \widehat{G}^{(2)}$ with the same vectors. Again invoking Algorithm 1 provides us the low-rank approximations of these off-diagonal blocks.

The algorithm continues in the same fashion for higher levels. The combined random tests lead to a constant number of matvec at each level. As there are $\log(n)$ levels in total, the total number of matrix–vector multiplications scales logarithmically.

When the block size on a level becomes smaller than the given criteria (for example, the numerical rank r used in the construction), one could switch to a deterministic way to get the off-diagonal blocks. In particular, we stop at a level L ($L < L_M$) such that each $\mathcal{I}_{L;i}$ contains about r entries. Now only the elements in the diagonal blocks $G_{L,ii}$ need to be determined. This can be completed by applying G to the matrix

$$(I, I, \ldots, I)^{\mathrm{T}},$$

where I is the identity matrix whose dimension is equal to the number of indices in $\mathcal{I}_{L;i}$.

Let us summarize the structure of our algorithm. From the top level to the bottom level, we peel off the numerically low-rank off-diagonal blocks using the randomized SVD algorithm. The matrix–vector multiplications required by the randomized SVD algorithms are computed effectively by *combining* several random tests into one using the zero pattern of the *remaining* matrix. In this way, we get an efficient algorithm for constructing the hierarchical representation for the matrix G.

1.4. Related works

Our algorithm is built on top of the framework of the $\mathcal{H}$-matrices proposed by Hackbusch and his collaborators [1,2,4]. The definitions of the $\mathcal{H}$-matrices will be summarized in Section 2. In a nutshell, the $\mathcal{H}$-matrix framework is an operational matrix algebra for efficiently representing, applying, and manipulating discretizations of operators from elliptic partial differential equations. Though we have known how to represent and apply these matrices for quite some time [10], it is the contribution of the $\mathcal{H}$-matrix framework that enables one to manipulate them in a general and coherent way. A closely related matrix algebra is also developed in a more numerical-linear-algebraic viewpoint under the name *hierarchical semiseparable matrices* by Chandrasekaran et al. [11,12]. Here, we will follow the notations of the $\mathcal{H}$-matrices as our main motivations are from numerical solutions of elliptic PDEs.

A basic assumption of our algorithm is the existence of a fast matrix–vector multiplication subroutine. The most common case is when G is the inverse of the stiffness matrix H of a general elliptic operator. Since H is often sparse, much effort has been devoted to computing $u = Gf$ by solving the linear system $Hu = f$. Many ingenious algorithms have been developed for this purpose in the past forty years. Commonly-seen examples include multifrontal algorithms [13,14], geometric multigrids [15,16,2], algebraic multigrids (AMG) [17], domain decompositions methods [18,19], wavelet-based fast algorithms [20] and preconditioned conjugate gradient algorithms (PCG) [21], to name a few. Very recently, both Chandrasekaran et al. [22] and Martinsson [23] have combined the idea of the multifrontal algorithms with the $\mathcal{H}$-matrices to obtain highly efficiently direct solvers for $Hu = f$. Another common case for which a fast matrix–vector multiplication subroutine is available comes from the boundary integral equations where G is often a discretization of a Green's function restricted to a domain boundary.

Fast algorithms developed for this case include the famous fast multipole method [10], the panel clustering method [24], and others. All these fast algorithms mentioned above can be used as the "black box" algorithm for our method.

As shown in the previous section, our algorithm relies heavily on the randomized singular value decomposition algorithm for constructing the factorizations of the off-diagonal blocks. This topic has been a highly active research area in the past several years and many different algorithms have been proposed in the literature. Here, for our purpose, we have adopted the algorithm described in [7,8]. In a related but slightly different problem, the goal is to find low-rank approximations $A = CUR$ where C contains a subset of columns of A and R contains a subset of rows. Papers devoted to this task include [25–28]. In our setting, since we assume no direct access of entries of the matrix A but only its impact through matrix–vector multiplications, the algorithm proposed by Liberty et al. [7] is the most relevant choice. An excellent recent review of this fast growing field can be found in [6].

In a recent paper [29], Martinsson considered also the problem of constructing the $\mathcal{H}$-matrix representation of a matrix, but he assumed that one can access arbitrary entries of the matrix besides the fast matrix–vector multiplication subroutine. Under this extra assumption, he showed that one can construct the $\mathcal{H}^2$ representation of the matrix with $\mathcal{O}(1)$ matrix–vector multiplications and accesses of $\mathcal{O}(n)$ matrix entries. However, in many situations including the case of G being the inverse of the stiffness matrix of an elliptic differential operator, accessing entries of G is by no means a trivial task. Comparing with Martinsson's work, our algorithm only assumes the existence of a fast matrix–vector multiplication subroutine, and hence is more general.

As we mentioned earlier, one motivation for computing G explicitly is to further compress the matrix G. The most common example in the literature of numerical analysis is the process of numerical homogenization or upscaling [5]. Here the matrix G is often again the inverse of the stiffness matrix H of an elliptic partial differential operator. When H contains information from all scales, the standard homogenization techniques fail. Recently, Owhadi and Zhang [30] proposed an elegant method that, under the assumption that the Cordes condition is satisfied, upscales a general H in divergence form using metric transformation. Computationally, their approach involves d solves of form $Hu = f$ with d being the dimension of the problem. On the other hand, if G is computed using our algorithm, one can obtain the upscaled operator by inverting a low-passed and down-sampled version of G. Complexity-wise, our algorithm is more costly since it requires $\mathcal{O}(\log n)$ solves of $Hu = f$. However, since our approach makes no analytic assumptions about H, it is expected to be more general.

2. Algorithm

We now present the details of our algorithm in two dimensions. In addition to a top-down construction using the peeling idea presented in the introduction, the complexity will be further reduced using the $\mathcal{H}^2$ property of the matrix [1,3]. The extension to three dimensions is straightforward.

In two dimensions, a more conservative partition of the domain is required to guarantee the low-rankness of the matrix blocks. We will start with discussion of this new geometric setup. Then we will recall the notion of hierarchical matrices and related algorithms in Section 2.2. The algorithm to construct an $\mathcal{H}^2$ representation for a matrix using matrix–vector multiplications will be presented in Sections 2.3 and 2.4. Finally, variants of the algorithm for constructing the $\mathcal{H}^1$ and uniform $\mathcal{H}^1$ representations will be described in Section 2.5.

2.1. Geometric setup and notations

Let us consider an operator G defined on a 2D domain $[0,1)^2$ with periodic boundary condition. We discretize the problem using an $n = N \times N$ uniform grid with N being a power of 2: $N = 2^{L_M}$. Denote the set of all grid points as

$$\mathcal{I}_0 = \{(k_1/N, k_2/N) | k_1, k_2 \in \mathbb{N}, 0 \leqslant k_1, k_2 < N\} \tag{11}$$

and partition the domain hierarchically into $L + 1$ levels ($L < L_M$). On each level l ($0 \leqslant l \leqslant L$), we have $2^l \times 2^l$ boxes denoted by $\mathcal{I}_{l;ij} = [(i-1)/2^l, i/2^l) \times [(j-1)/2^l, j/2^l)$ for $1 \leqslant i, j \leqslant 2^l$. The symbol $\mathcal{I}_{l;ij}$ will also be used to denote the grid points that lies in the box $\mathcal{I}_{l;ij}$. The meaning should be clear from the context. We will also use $\mathcal{I}_l$ (or $\mathcal{J}_l$) to denote a general box on certain level l. The subscript l will be omitted, when the level is clear from the context. For a given box $\mathcal{I}_l$ for $l \geqslant 1$, we call a box $\mathcal{J}_{l-1}$ on level $l-1$ its parent if $\mathcal{I}_l \subset \mathcal{J}_{l-1}$. Naturally, $\mathcal{I}_l$ is called a child of $\mathcal{J}_{l-1}$. It is clear that each box except those on level L will have four children boxes.

For any box $\mathcal{I}$ on level l, it covers $N/2^l \times N/2^l$ grid points. The last level L can be chosen so that the leaf box has a constant number of points in it (*i.e.* the difference $L_M - L$ is kept to be a constant when N increases).

For simplicity of presentation, we will start the method from level 3. It is also possible to start from level 2. Level 2 needs to be treated specially, as for level 3. We define the following notations for a box $\mathcal{I}$ on level l ($l \geqslant 3$):

$\mathrm{NL}(\mathcal{I})$ Neighbor list of box $\mathcal{I}$. This list contains the boxes on level l that are adjacent to $\mathcal{I}$ and also $\mathcal{I}$ itself. There are 9 boxes in the list for each $\mathcal{I}$.

$\mathrm{IL}(\mathcal{I})$ Interaction list of box $\mathcal{I}$. When $l = 3$, this list contains all the boxes on level 3 minus the set of boxes in $\mathrm{NL}(\mathcal{I})$. There are 55 boxes in total. When $l > 3$, this list contains all the boxes on level l that are children of boxes in $\mathrm{NL}(\mathcal{P})$ with $\mathcal{P}$ being $\mathcal{I}$'s parent minus the set of boxes in $\mathrm{NL}(\mathcal{I})$. There are 27 such boxes.

Notice that these two lists determine two symmetric relationship: $\mathcal{J} \in \mathrm{NL}(\mathcal{I})$ if and only if $\mathcal{I} \in \mathrm{NL}(\mathcal{J})$ and $\mathcal{J} \in \mathrm{IL}(\mathcal{I})$ if and only if $\mathcal{I} \in \mathrm{IL}(\mathcal{J})$. Figs. 1 and 2 illustrate the computational domain and the lists for $l = 3$ and $l = 4$, respectively.

For a vector f defined on the $N \times N$ grid $\mathcal{I}_0$, we define $f(\mathcal{I})$ to be the restriction of f to grid points $\mathcal{I}$. For a matrix $G \in \mathbb{R}^{N^2 \times N^2}$ that represents a linear map from $\mathcal{I}_0$ to itself, we define $G(\mathcal{I}, \mathcal{J})$ to be the restriction of G on $\mathcal{I} \times \mathcal{J}$.

A matrix $G \in \mathbb{R}^{N^2 \times N^2}$ has the following decomposition

$$G = G^{(3)} + G^{(4)} + \cdots + G^{(L)} + D^{(L)}. \tag{12}$$

Here, for each l, $G^{(l)}$ incorporates the interaction on level l between a box with its interaction list. More precisely, $G^{(l)}$ has a $2^{2l} \times 2^{2l}$ block structure:

$$G^{(l)}(\mathcal{I}, \mathcal{J}) = \begin{cases} G(\mathcal{I}, \mathcal{J}), & \mathcal{I} \in \mathrm{IL}(\mathcal{J}) \ (\text{eq. } \mathcal{J} \in \mathrm{IL}(\mathcal{I})); \\ 0, & \text{otherwise} \end{cases}$$

with $\mathcal{I}$ and $\mathcal{J}$ both on level l. The matrix $D^{(L)}$ includes the interactions between adjacent boxes at level L:

$$D^{(L)}(\mathcal{I}, \mathcal{J}) = \begin{cases} G(\mathcal{I}, \mathcal{J}), & \mathcal{I} \in \mathrm{NL}(\mathcal{J}) \ (\text{eq. } \mathcal{J} \in \mathrm{NL}(\mathcal{I})); \\ 0, & \text{otherwise} \end{cases}$$

with $\mathcal{I}$ and $\mathcal{J}$ both on level L. To show that (12) is true, it suffices to prove that for any two boxes $\mathcal{I}$ and $\mathcal{J}$ on level L, the right hand side gives $G(\mathcal{I}, \mathcal{J})$. In the case that $\mathcal{I} \in \mathrm{NL}(\mathcal{J})$, this is obvious. Otherwise, it is clear that we can find a level l, and boxes $\mathcal{I}'$ and $\mathcal{J}'$ on level l, such that $\mathcal{I}' \in \mathrm{IL}(\mathcal{J}')$, $\mathcal{I} \subset \mathcal{I}'$ and $\mathcal{J} \subset \mathcal{J}'$, and hence $G(\mathcal{I}, \mathcal{J})$ is given through $G(\mathcal{I}', \mathcal{J}')$. Throughout the text, we will use $\|A\|_2$ to denote the matrix 2-norm of matrix A.

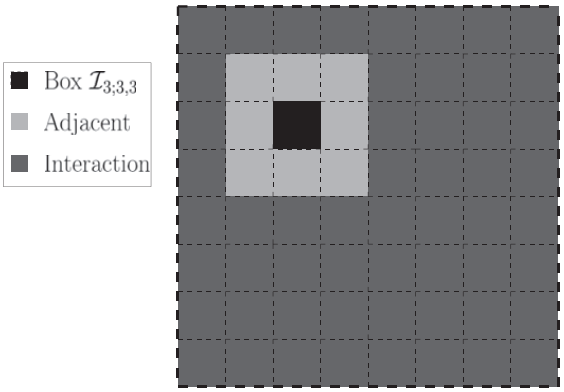

Fig. 1. Illustration of the computational domain at level 3. $\mathcal{I}_{3;3,3}$ is the black box. The neighbor list $\mathrm{NL}(\mathcal{I}_{3;3,3})$ consists of 8 adjacent light gray boxes and the black box itself, and the interaction list $\mathrm{IL}(\mathcal{I}_{3;3,3})$ consists of the 55 dark gray boxes.

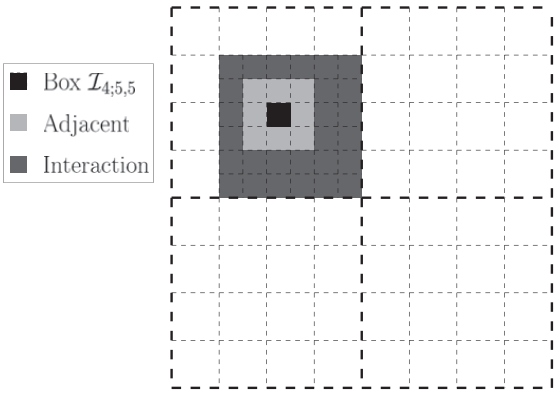

Fig. 2. Illustration of the computational domain at level 4. $\mathcal{I}_{4;5,5}$ is the black box. The neighbor list $\mathrm{NL}(\mathcal{I}_{4;5,5})$ consists of 8 adjacent light gray boxes and the black box itself, and the interaction list $\mathrm{IL}(\mathcal{I}_{4;5,5})$ consists of the 27 dark gray boxes.

2.2. Hierarchical matrix

Our algorithm works with the so-called hierarchical matrices. We recall in this subsection some basic properties of this type of matrices and also some related algorithms. For simplicity of notations and representation, we will only work with symmetric matrices. For a more detailed introduction of the hierarchical matrices and their applications in fast algorithms, we refer the readers to [2,3].

2.2.1. $\mathcal{H}^1$ matrices

Definition 1. G is a (symmetric) $\mathcal{H}^1$-matrix if for any $\varepsilon > 0$, there exists $r(\varepsilon) \lesssim \log(\varepsilon^{-1})$ such that for any pair $(\mathcal{I}, \mathcal{J})$ with $\mathcal{I} \in \mathsf{IL}(\mathcal{J})$, there exist orthogonal matrices $U_{\mathcal{I}\mathcal{J}}$ and $U_{\mathcal{J}\mathcal{I}}$ with $r(\varepsilon)$ columns and matrix $M_{\mathcal{I}\mathcal{J}} \in \mathbb{R}^{r(\varepsilon)\times r(\varepsilon)}$ such that

$$\|G(\mathcal{I},\mathcal{J}) - U_{\mathcal{I}\mathcal{J}} M_{\mathcal{I}\mathcal{J}} U_{\mathcal{J}\mathcal{I}}^{\mathrm{T}}\|_2 \leqslant \varepsilon \|G(\mathcal{I},\mathcal{J})\|_2. \tag{13}$$

The main advantage of the $\mathcal{H}^1$ matrix is that the application of such matrix on a vector can be efficiently evaluated: Within error $\mathcal{O}(\varepsilon)$, one can use $\widehat{G}(\mathcal{I},\mathcal{J}) = U_{\mathcal{I}\mathcal{J}} M_{\mathcal{I}\mathcal{J}} U_{\mathcal{J}\mathcal{I}}^{\mathrm{T}}$, which is low-rank, instead of the original block $G(\mathcal{I},\mathcal{J})$. The algorithm is described in Algorithm 2. It is standard that the complexity of the matrix–vector multiplication for an $\mathcal{H}^1$ matrix is $\mathcal{O}(N^2 \log N)$ [2].

Algorithm 2. Application of a $\mathcal{H}^1$-matrix G on a vector f

```
u = 0;
for l = 3 to L do
  for I on level l do
    for J ∈ IL(I) do
      u(I) = u(I) + U_IJ ( M_IJ ( U_JI^T f(J) ) );
    end for
  end for
end for
for I on level L do
  for J ∈ NL(I) do
    u(I) = u(I) + G(I,J) f(J);
  end for
end for
```

2.2.2. Uniform $\mathcal{H}^1$ matrix

Definition 2. G is a (symmetric) uniform $\mathcal{H}^1$-matrix if for any $\varepsilon > 0$, there exists $r_U(\varepsilon) \lesssim \log(\varepsilon^{-1})$ such that for each box $\mathcal{I}$, there exists an orthogonal matrix $U_{\mathcal{I}}$ with $r_U(\varepsilon)$ columns such that for any pair $(\mathcal{I}, \mathcal{J})$ with $\mathcal{I} \in \mathsf{IL}(\mathcal{J})$

$$\left\| G(\mathcal{I},\mathcal{J}) - U_{\mathcal{I}} N_{\mathcal{I}\mathcal{J}} U_{\mathcal{J}}^{\mathrm{T}} \right\|_2 \leqslant \varepsilon \|G(\mathcal{I},\mathcal{J})\|_2 \tag{14}$$

with $N_{\mathcal{I}\mathcal{J}} \in \mathbb{R}^{r_U(\varepsilon)\times r_U(\varepsilon)}$.

The application of a uniform $\mathcal{H}^1$ matrix to a vector is described in Algorithm 3. The complexity of the algorithm is still $\mathcal{O}(N^2 \log N)$. However, the prefactor is much better as each $U_{\mathcal{I}}$ is applied only once. The speedup over Algorithm 2 is roughly $27 r(\varepsilon)/r_U(\varepsilon)$ [2].

Algorithm 3. Application of a uniform $\mathcal{H}^1$-matrix G on a vector f

```
u = 0;
for l = 3 to L do
  for J on level l do
    f̃_J = U_J^T f(J);
  end for
end for
for l = 3 to L do
  for I on level l do
    u(I) = u(I) + U_I ũ_I;
  end for
end for
for I on level L do
  for J ∈ NL(I) do
```

(continued on next page)

Algorithm 3 (*continued*)

Algorithm 3. Application of a uniform $\mathcal{H}^1$-matrix G on a vector f

```
8:   for 𝓘 on level l do
9:     ũ_𝓘 = 0;
10:    for 𝓙 ∈ IL(𝓘) do
11:      ũ_𝓘 = ũ_𝓘 + N_{𝓘𝓙} f̃_𝓙;
12:    end for
13:  end for
14: end for
22:    u(𝓘) = u(𝓘) + G(𝓘, 𝓙) f(𝓙);
23:  end for
24: end for
```

2.2.3. $\mathcal{H}^2$ matrices

Definition 3. G is an $\mathcal{H}^2$ matrix if

- it is a uniform $\mathcal{H}^1$ matrix;
- Suppose that $\mathcal{C}$ is any child of a box $\mathcal{I}$, then

$$\|U_{\mathcal{I}}(\mathcal{C},:) - U_{\mathcal{C}} T_{\mathcal{C}\mathcal{I}}\|_2 \lesssim \varepsilon \tag{15}$$

for some matrix $T_{\mathcal{C}\mathcal{I}} \in \mathbb{R}^{r_U(\varepsilon)\times r_U(\varepsilon)}$.

The application of an $\mathcal{H}^2$ matrix to a vector is described in Algorithm 4 and it has a complexity of $\mathcal{O}(N^2)$, Notice that, compared with $\mathcal{H}^1$ matrix, the logarithmic factor is reduced [3].

Algorithm 4. Application of a $\mathcal{H}^2$-matrix G on a vector f

```
1: u = 0;
2: for 𝓙 on level L do
3:   f̃_𝓙 = U_𝓙^T f(𝓙);
4: end for
5: for l = L − 1 down to 3 do
6:   for 𝓙 on level l do
7:     f̃_𝓙 = 0;
8:     for each child 𝒞 of 𝓙 do
9:       f̃_𝓙 = f̃_𝓙 + T_{𝒞𝓙}^T f̃_𝒞;
10:    end for
11:  end for
12: end for
13: for l = 3 to L do
14:   for 𝓘 on level l do
15:     ũ_𝓘 = 0;
16:     for 𝓙 ∈ IL(𝓘) do
17:       ũ_𝓘 = ũ_𝓘 + N_{𝓘𝓙} f̃_𝓙;
18:     end for
19:   end for
20: end for
18: for l = 3 to L − 1 do
19:   for 𝓘 on level l do
20:     for each child 𝒞 of 𝓘 do
21:       ũ_𝒞 = ũ_𝒞 + T_{𝒞𝓘} ũ_𝓘;
22:     end for
23:   end for
24: end for
25: for 𝓘 on level L do
26:   u(𝓘) = U_𝓘 ũ_𝓘;
27: end for
28: for 𝓘 on level L do
29:   for 𝓙 ∈ NL(𝓘) do
30:     u(𝓘) = u(𝓘) + G(𝓘, 𝓙) f(𝓙);
31:   end for
32: end for
```

Remark. Applying an $\mathcal{H}^2$ matrix to a vector can indeed be viewed as the matrix form of the fast multipole method (FMM) [10]. One recognizes in Algorithm 4 that the second top-level **for** loop corresponds to the M2M (multipole expansion to multipole expansion) translations of the FMM; the third top-level **for** loop is the M2L (multipole expansion to local expansion) translations; and the fourth top-level **for** loop is the L2L (local expansion to local expansion) translations.

In the algorithm to be introduced, we will also need to apply a partial matrix $G^{(3)} + G^{(4)} + \cdots + G^{(L')}$ for some $L' \leqslant L$ to a vector f. This amounts to a variant of Algorithm 4, described in Algorithm 5.

Algorithm 5. Application of a partial $\mathcal{H}^2$-matrix $G^{(3)} + \cdots + G^{(L')}$ on a vector f

```
u = 0;
for 𝒥 on level L' do
  f̃_𝒥 = U_𝒥^T f(𝒥);
end for
for l = L' − 1 down to 3 do
  for 𝒥 on level l do
    f̃_𝒥 = 0;
    for each child 𝒞 of 𝒥 do
      f̃_𝒥 = f̃_𝒥 + T_{𝒞𝒥}^T f̃_𝒞;
    end for
  end for
end for
for l = 3 to L' do
  for ℐ on level l do
    ũ_ℐ = 0;
    for 𝒥 ∈ IL(ℐ) do
      ũ_ℐ = ũ_ℐ + N_{ℐ𝒥} f̃_𝒥;
    end for
  end for
end for
for l = 3 to L' − 1 do
  for ℐ on level l do
    for each child 𝒞 of ℐ do
      ũ_𝒞 = ũ_𝒞 + T_{𝒞ℐ} ũ_ℐ;
    end for
  end for
end for
for ℐ on level L' do
  u(ℐ) = U_ℐ ũ_ℐ;
end for
```

2.3. Peeling algorithm: outline and preparation

We assume that G is a symmetric $\mathcal{H}^2$ matrix and that there exists a fast matrix–vector subroutine for applying G to any vector f as a "black box". The goal is to construct an $\mathcal{H}^2$ representation of the matrix G using only a small number of test vectors.

The basic strategy is a top-down construction: For each level $l = 3,\ldots,L$, assume that an $\mathcal{H}^2$ representation for $G^{(3)} + \cdots + G^{(l-1)}$ is given, we construct $G^{(l)}$ by the following three steps:

1. *Peeling*. Construct an $\mathcal{H}^1$ representation for $G^{(l)}$ using the peeling idea and the $\mathcal{H}^2$ representation for $G^{(3)} + \cdots + G^{(l-1)}$.
2. *Uniformization.* Construct a uniform $\mathcal{H}^1$ representation for $G^{(l)}$ from its $\mathcal{H}^1$ representation.
3. *Projection.* Construct an $\mathcal{H}^2$ representation for $G^{(3)} + \cdots + G^{(l)}$.

The names of these steps will be made clear in the following discussion. Variants of the algorithm that only construct an $\mathcal{H}^1$ representation (a uniform $\mathcal{H}^1$ representation, respectively) of the matrix G can be obtained by only doing the peeling step (the peeling and uniformization steps, respectively). These variants will be discussed in Section 2.5.

After we have the $\mathcal{H}^2$ representation for $G^{(3)} + \cdots + G^{(L)}$, we use the peeling idea again to extract the diagonal part $D^{(L)}$. We call this whole process the *peeling* algorithm.

Before detailing the peeling algorithm, we mention two procedures that serve as essential components of our algorithm. The first procedure concerns with the uniformization step, in which one needs to get a uniform $\mathcal{H}^1$ representation for $G^{(l)}$ from its $\mathcal{H}^1$ representation, *i.e.* from $\widehat{G}(\mathcal{I},\mathcal{J}) = U_{\mathcal{I}\mathcal{J}} M_{\mathcal{I}\mathcal{J}} U_{\mathcal{J}\mathcal{I}}^{\mathrm{T}}$ to $\widehat{G}(\mathcal{I},\mathcal{J}) = U_{\mathcal{I}} N_{\mathcal{I}\mathcal{J}} U_{\mathcal{J}}^{\mathrm{T}}$, for all pairs of boxes $(\mathcal{I},\mathcal{J})$ with $\mathcal{I} \in \mathrm{IL}(\mathcal{J})$. To this end, what we need to do is to find the column space of

$$\left[U_{\mathcal{I}\mathcal{J}_1} M_{\mathcal{I}\mathcal{J}_1} | U_{\mathcal{I}\mathcal{J}_2} M_{\mathcal{I}\mathcal{J}_2} | \cdots | U_{\mathcal{I}\mathcal{J}_t} M_{\mathcal{I}\mathcal{J}_t}\right], \tag{16}$$

where $\mathcal{J}_j$ are the boxes in $\mathrm{IL}(\mathcal{I})$ and $t = |\mathrm{IL}(\mathcal{I})|$. Notice that we weight the singular vectors U by M, so that the singular vectors corresponding to larger singular values will be more significant. This column space can be found by the usual SVD algorithm or a more effective randomized version presented in Algorithm 6. The important left singular vectors are denoted by $U_{\mathcal{I}}$, and the diagonal matrix formed by the singular values associated with $U_{\mathcal{I}}$ is denoted by $S_{\mathcal{I}}$.

Algorithm 6. Construct a uniform $\mathcal{H}^1$ representation of G from the $\mathcal{H}^1$ representation at a level l

1: **for** each box $\mathcal{I}$ on level l **do**
2: Generate a Gaussian random matrix $R \in \mathbb{R}^{(r(\varepsilon)\times t)\times(r_U(\varepsilon)+c)}$;
3: Form product $[U_{\mathcal{IJ}_1}M_{\mathcal{IJ}_1}|\cdots|U_{\mathcal{IJ}_t}M_{\mathcal{IJ}_t}]R$ and apply SVD to it. The first $r_U(\varepsilon)$ left singular vectors give $U_{\mathcal{I}}$, and the corresponding singular values give a diagonal matrix $S_{\mathcal{I}}$;
4: **for** $\mathcal{J}_j \in \mathrm{IL}(\mathcal{I})$ **do**
5: $I_{\mathcal{IJ}_j} = U_{\mathcal{I}}^{\mathrm{T}}U_{\mathcal{IJ}_j}$;
6: **end for**
7: **end for**
8: **for** each pair $(\mathcal{I},\mathcal{J})$ on level l with $\mathcal{I} \in \mathrm{IL}(\mathcal{J})$ **do**
9: $N_{\mathcal{IJ}} = I_{\mathcal{IJ}}M_{\mathcal{IJ}}I_{\mathcal{JI}}^{\mathrm{T}}$;
10: **end for**

Complexity analysis: For a box $\mathcal{I}$ on level l, the number of grid points in $\mathcal{I}$ is $(N/2^l)^2$. Therefore, $U_{\mathcal{IJ}_j}$ are all of size $(N/2^l)^2 \times r(\varepsilon)$ and $M_{\mathcal{IJ}}$ are of size $r(\varepsilon)\times r(\varepsilon)$. Forming the product $[U_{\mathcal{IJ}_1}M_{\mathcal{IJ}_1}|\cdots|U_{\mathcal{IJ}_t}M_{\mathcal{IJ}_t}]R$ takes $\mathcal{O}((N/2^l)^2 r(\varepsilon)(r_U(\varepsilon)+c))$ steps and SVD takes $\mathcal{O}((N/2^l)^2(r_U(\varepsilon)+c)^2)$ steps. As there are 2^{2l} boxes on level l, the overall cost of Algorithm 6 is $\mathcal{O}(N^2(r_U(\varepsilon)+c)^2) = \mathcal{O}(N^2)$. One may also apply to $[U_{\mathcal{IJ}_1}M_{\mathcal{IJ}_1}|\cdots|U_{\mathcal{IJ}_t}M_{\mathcal{IJ}_t}]$ the deterministic SVD algorithm, which has the same order of complexity but with a prefactor about $27r(\varepsilon)/(r_U(\varepsilon)+c)$ times larger.

The second procedure is concerned with the projection step of the above list, in which one constructs an $\mathcal{H}^2$ representation for $G^{(3)}+\cdots+G^{(l)}$. Here, we are given the $\mathcal{H}^2$ representation for $G^{(3)}+\cdots+G^{(l-1)}$ along with the uniform $\mathcal{H}^1$ representation for $G^{(l)}$ and the goal is to compute the transfer matrix $T_{\mathcal{CI}}$ for a box $\mathcal{I}$ on level $l-1$ and its child $\mathcal{C}$ on level l such that

$$\|U_{\mathcal{I}}(\mathcal{C},:) - U_{\mathcal{C}}T_{\mathcal{CI}}\|_2 \lesssim \varepsilon.$$

In fact, the existing $U_{\mathcal{C}}$ matrix of the uniform $\mathcal{H}^1$ representation may not be rich enough to contain the columns of $U_{\mathcal{I}}(\mathcal{C},:)$ in its span. Therefore, one needs to update the content of $U_{\mathcal{C}}$ as well. To do that, we perform a singular value decomposition for the combined matrix

$$[U_{\mathcal{I}}(\mathcal{C},:)S_{\mathcal{I}}|U_{\mathcal{C}}S_{\mathcal{C}}]$$

and define a matrix $V_{\mathcal{C}}$ to contain $r_U(\varepsilon)$ left singular vectors. Again $U_{\mathcal{I}}, U_{\mathcal{C}}$ should be weighted by the corresponding singular values. The transfer matrix $T_{\mathcal{CI}}$ is then given by

$$T_{\mathcal{CI}} = V_{\mathcal{C}}^{\mathrm{T}}U_{\mathcal{I}}(\mathcal{C},:)$$

and the new $U_{\mathcal{C}}$ is set to be equal to $V_{\mathcal{C}}$. Since $U_{\mathcal{C}}$ has been changed, the matrices $N_{\mathcal{CD}}$ for $\mathcal{D} \in \mathrm{IL}(\mathcal{C})$ and also the corresponding singular values $S_{\mathcal{C}}$ need to be updated as well. The details are listed in Algorithm 7.

Algorithm 7. Construct an $\mathcal{H}^2$ representation of G from the uniform $\mathcal{H}^1$ representation at level l

1: **for** each box $\mathcal{I}$ on level $l-1$ **do**
2: **for** each child $\mathcal{C}$ of $\mathcal{I}$ **do**
3: Form matrix $[U_{\mathcal{I}}(\mathcal{C},:)S_{\mathcal{I}}|U_{\mathcal{C}}S_{\mathcal{C}}]$ and apply SVD to it. The first $r_U(\varepsilon)$ left singular vectors give $V_{\mathcal{C}}$, and the corresponding singular values give a diagonal matrix $W_{\mathcal{C}}$;
4: $K_{\mathcal{C}} = V_{\mathcal{C}}^{\mathrm{T}}U_{\mathcal{C}}$;
5: $T_{\mathcal{CI}} = V_{\mathcal{C}}^{\mathrm{T}}U_{\mathcal{I}}(\mathcal{C},:)$;
6: $U_{\mathcal{C}} = V_{\mathcal{C}}$;
7: $S_{\mathcal{C}} = W_{\mathcal{C}}$;
8: **end for**
9: **end for**
10: **for** each pair $(\mathcal{C},\mathcal{D})$ on level l with $\mathcal{C} \in \mathrm{IL}(\mathcal{D})$ **do**
11: $N_{\mathcal{CD}} = K_{\mathcal{C}}N_{\mathcal{CD}}K_{\mathcal{D}}^{\mathrm{T}}$;
12: **end for**

Complexity analysis: The main computational task of Algorithm 7 is again the SVD part. For a box $\mathcal{C}$ on level l, the number of grid points in $\mathcal{I}$ is $(N/2^l)^2$. Therefore, the combined matrix $[U_{\mathcal{I}}(\mathcal{C},:)S_{\mathcal{I}}|U_{\mathcal{C}}S_{\mathcal{C}}]$ is of size $(N/2^l)^2 \times 2r_U(\varepsilon)$. The SVD computation clearly takes $\mathcal{O}((N/2^l)^2 r_U(\varepsilon)^2) = \mathcal{O}((N/2^l)^2)$ steps. Taking into the consideration that there are 2^{2l} boxes on level l gives rise to an $\mathcal{O}(N^2)$ estimate for the cost of Algorithm 7.

2.4. Peeling algorithm: details

With the above preparation, we are now ready to describe the peeling algorithm in detail at different levels, starting from level 3. At each level, we follow exactly the three steps listed at the beginning of Section 2.3.

2.4.1. Level 3

First in the peeling step, we construct the $\mathcal{H}^1$ representation for $G^{(3)}$. For each pair $(\mathcal{I},\mathcal{J})$ on level 3 such that $\mathcal{I}\in \mathsf{IL}(\mathcal{J})$, we will invoke randomized SVD Algorithm 1 to construct the low rank approximation of $G_{\mathcal{I},\mathcal{J}}$. However, in order to apply the algorithm we need to compute $G(\mathcal{I},\mathcal{J})R_{\mathcal{J}}$ and $R_{\mathcal{I}}^{\mathrm{T}}G(\mathcal{I},\mathcal{J})$, where $R_{\mathcal{I}}$ and $R_{\mathcal{J}}$ are random matrices with $r(\varepsilon)+c$ columns. For each box $\mathcal{J}$ on level 3, we construct a matrix R of size $N^2\times(r(\varepsilon)+c)$ such that

$$R(\mathcal{J},:)=R_{\mathcal{J}}\quad\text{and}\quad R(\mathcal{I}_0\setminus\mathcal{J},:)=0.$$

Computing GR using $r(\varepsilon)+c$ matvecs and restricting the result to grid points $\mathcal{I}\in\mathsf{IL}(\mathcal{J})$ gives $G(\mathcal{I},\mathcal{J})R_{\mathcal{J}}$ for each $\mathcal{I}\in\mathsf{IL}(\mathcal{J})$.

After repeating these steps for all boxes on level 3, we hold for any pair $(\mathcal{I},\mathcal{J})$ with $\mathcal{I}\in\mathsf{IL}(\mathcal{J})$ the following data:

$$G(\mathcal{I},\mathcal{J})R_{\mathcal{J}}\quad\text{and}\quad R_{\mathcal{I}}^{\mathrm{T}}G(\mathcal{I},\mathcal{J})=(G(\mathcal{J},\mathcal{I})R_{\mathcal{I}})^{\mathrm{T}}.$$

Now, applying Algorithm 1 to them gives the low-rank approximation

$$\widehat{G}(\mathcal{I},\mathcal{J})=U_{\mathcal{I}\mathcal{J}}M_{\mathcal{I}\mathcal{J}}U_{\mathcal{J}\mathcal{I}}^{\mathrm{T}}. \tag{17}$$

In the uniformization step, in order to get the uniform $\mathcal{H}^1$ representation for $G^{(3)}$, we simply apply Algorithm 6 to the boxes on level 3 to get the approximations

$$\widehat{G}(\mathcal{I},\mathcal{J})=U_{\mathcal{I}}N_{\mathcal{I}\mathcal{J}}U_{\mathcal{J}}^{\mathrm{T}}. \tag{18}$$

Finally in the projection step, since we only have 1 level now (level 3), we have already the $\mathcal{H}^2$ representation for $G^{(3)}$.

Complexity analysis: The dominant computation is the construction of the $\mathcal{H}^1$ representation for $G^{(3)}$. This requires $r(\varepsilon)+c$ matvecs for each box $\mathcal{I}$ on level 3. Since there are in total 64 boxes at this level, the total cost is $64(r(\varepsilon)+c)$ matvecs. From the complexity analysis in Section 2.3, the computation for the second and third steps cost an extra $\mathcal{O}(N^2)$ steps.

2.4.2. Level 4

First in the peeling step, in order to construct the $\mathcal{H}^1$ representation for $G^{(4)}$, we need to compute the matrices $G(\mathcal{I},\mathcal{J})R_{\mathcal{J}}$ and $R_{\mathcal{I}}^{\mathrm{T}}G(\mathcal{I},\mathcal{J})$ for each pair $(\mathcal{I},\mathcal{J})$ on level 4 with $\mathcal{I}\in\mathsf{IL}(\mathcal{J})$. Here $R_{\mathcal{I}}$ and $R_{\mathcal{J}}$ are again random matrices with $r(\varepsilon)+c$ columns.

One approach is to follow exactly what we did for level 3: Fix a box $\mathcal{J}$ at this level, construct R of size $N^2\times(r(\varepsilon)+c)$ such that

$$R(\mathcal{J},:)=R_{\mathcal{J}}\quad\text{and}\quad R(\mathcal{I}_0\setminus\mathcal{J},:)=0.$$

Next apply $G-G^{(3)}$ to R, by subtracting GR and $G^{(3)}R$. The former is computed using $r(\varepsilon)+c$ matvecs and the latter is done by Algorithm 5. Finally, restrict the result to grid points $\mathcal{I}\in\mathsf{IL}(\mathcal{J})$ gives $G(\mathcal{I},\mathcal{J})R_{\mathcal{J}}$ for each $\mathcal{I}\in\mathsf{IL}(\mathcal{J})$.

However, we have observed in the simple one-dimensional example in Section 1.3 that random tests can be combined together as in Eqs. (6) and (7). We shall detail this observation in the more general situation here as following. Observe that $G-G^{(3)}=G^{(4)}+D^{(4)}$, and $G^{(4)}(\mathcal{J},\mathcal{I})$ and $D^{(4)}(\mathcal{J},\mathcal{I})$ for boxes $\mathcal{I}$ and $\mathcal{J}$ on level 4 is only nonzero if $\mathcal{I}\in\mathsf{NL}(\mathcal{J})\cup\mathsf{IL}(\mathcal{J})$. Therefore, $(G-G^{(3)})R$ for R coming from $\mathcal{J}$ can only be nonzero in $\mathsf{NL}(\mathcal{P})$ with $\mathcal{P}$ being $\mathcal{J}$'s parent. The rest is automatically zero (up to error ε as $G^{(3)}$ is approximated by its $\mathcal{H}^2$ representation). Therefore, we can combine the calculation of different boxes as long as their non-zero regions do not overlap.

More precisely, we introduce the following sets $\mathcal{S}_{pq}$ for $1\leqslant p,q\leqslant 8$ with

$$\mathcal{S}_{pq}=\{\mathcal{J}_{4;ij}|i\equiv p\,(\mathrm{mod}8),j\equiv q\,(\mathrm{mod}8)\}. \tag{19}$$

There are 64 sets in total, each consisting of four boxes. Fig. 3 illustrates one such set at level 4. For each set $\mathcal{S}_{pq}$, first construct R with

$$R(\mathcal{J},:)=\begin{cases}R_{\mathcal{J}}, & \mathcal{J}\in\mathcal{S}_{pq};\\ 0, & \text{otherwise}.\end{cases}$$

Then, we apply $G-G^{(3)}$ to R, by subtracting GR and $G^{(3)}R$. The former is computed using $r(\varepsilon)+c$ matvecs and the latter is done by Algorithm 5. For each $\mathcal{J}\in\mathcal{S}_{pq}$, restricting the result to $\mathcal{I}\in\mathsf{IL}(\mathcal{J})$ gives $G(\mathcal{I},\mathcal{J})R_{\mathcal{J}}$. Repeating this computation for all sets $\mathcal{S}_{pq}$ then provides us with the following data:

$$G(\mathcal{I},\mathcal{J})R_{\mathcal{J}}\quad\text{and}\quad R_{\mathcal{I}}^{\mathrm{T}}G(\mathcal{I},\mathcal{J})=(G(\mathcal{J},\mathcal{I})R_{\mathcal{I}})^{\mathrm{T}},$$

for each pair $(\mathcal{I},\mathcal{J})$ with $\mathcal{I}\in\mathsf{IL}(\mathcal{J})$. Applying Algorithm 1 to them gives the required low-rank approximations

$$\widehat{G}(\mathcal{I},\mathcal{J})=U_{\mathcal{I}\mathcal{J}}M_{\mathcal{I}\mathcal{J}}U_{\mathcal{J}\mathcal{I}}^{\mathrm{T}} \tag{20}$$

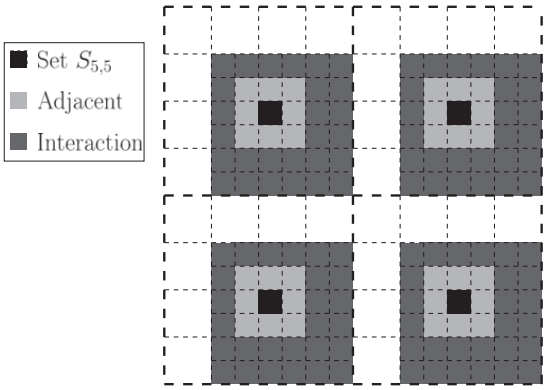

Fig. 3. Illustration of the set S_{55} at level 4. This set consists of four black boxes $\{\mathcal{I}_{4;5,5}, \mathcal{I}_{4;13,5}, \mathcal{I}_{4;5,13}, \mathcal{I}_{4;13,13}\}$. The light gray boxes around each black box are in the neighbor list and the dark gray boxes in the interaction list.

with $U_{\mathcal{I}\mathcal{J}}$ orthogonal.

Next in the uniformization step, the task is to construct the uniform $\mathcal{H}^1$ representation of $G^{(4)}$. Similar to the computation at level 3, we simply apply Algorithm 6 to the boxes on level 4 to get

$$\widehat{G}(\mathcal{I},\mathcal{J}) = U_{\mathcal{I}} N_{\mathcal{I}\mathcal{J}} U_{\mathcal{J}}^{\mathrm{T}}. \tag{21}$$

Finally in the projection step, to get $\mathcal{H}^2$ representation for $G^{(3)} + G^{(4)}$, we invoke Algorithm 7 at level 4. Once it is done, we hold the transfer matrices $T_{\mathcal{C}\mathcal{I}}$ between any $\mathcal{I}$ on level 3 and each of its children $\mathcal{C}$, along with the updated uniform $\mathcal{H}^1$-matrix representation of $G^{(4)}$.

Complexity analysis: The dominant computation is again the construction of $\mathcal{H}^1$ representation for $G^{(4)}$. For each group $\mathcal{S}_{pq}$, we apply G to $r(\varepsilon)+c$ vectors and apply $G^{(3)}$ to $r(\varepsilon)+c$ vectors. The latter takes $\mathcal{O}(N^2)$ steps for each application. Since there are 64 sets in total, this computation takes $64(r(\varepsilon)+c)$ matvecs and $\mathcal{O}(N^2)$ extra steps.

2.4.3. Level l

First in the peeling step, to construct the $\mathcal{H}^1$ representation for $G^{(l)}$, we follow the discussion of level 4. Define 64 sets $\mathcal{S}_{pq}$ for $1 \leqslant p, q \leqslant 8$ with

$$\mathcal{S}_{pq} = \{\mathcal{J}_{l;ij} | i \equiv p \,(\mathrm{mod}8), j \equiv q \,(\mathrm{mod}8)\}. \tag{22}$$

Each set contains exactly $2^l/8 \times 2^l/8$ boxes. For each set $\mathcal{S}_{pq}$, construct R with

$$R(\mathcal{J},:) = \begin{cases} R_{\mathcal{J}}, & \mathcal{J} \in \mathcal{S}_{pq}; \\ 0, & \text{otherwise}. \end{cases}$$

Next, apply $G - [G^{(3)} + \cdots + G^{(l-1)}]$ to R, by subtracting GR and $[G^{(3)} + \cdots + G^{(l-1)}]R$. The former is again computed using $r(\varepsilon)+c$ matvecs and the latter is done by Algorithm 5 using the $\mathcal{H}^2$ representation of $G^{(3)} + \cdots + G^{(l-1)}$. For each $\mathcal{J} \in \mathcal{S}_{pq}$, restricting the result to $\mathcal{I} \in \mathsf{IL}(\mathcal{J})$ gives $G(\mathcal{I},\mathcal{J})R_{\mathcal{J}}$. Repeating this computation for all sets $\mathcal{S}_{pq}$ gives the following data for any pair $(\mathcal{I},\mathcal{J})$ with $\mathcal{I} \in \mathsf{IL}(\mathcal{J})$

$$G(\mathcal{I},\mathcal{J})R_{\mathcal{J}} \quad \text{and} \quad R_{\mathcal{I}}^{\mathrm{T}} G(\mathcal{I},\mathcal{J}) = (G(\mathcal{J},\mathcal{I})R_{\mathcal{I}})^{\mathrm{T}}.$$

Now applying Algorithm 1 to them gives the low-rank approximation

$$\widehat{G}(\mathcal{I},\mathcal{J}) = U_{\mathcal{I}\mathcal{J}} M_{\mathcal{I}\mathcal{J}} U_{\mathcal{J}\mathcal{I}}^{\mathrm{T}} \tag{23}$$

with $U_{\mathcal{I}\mathcal{J}}$ orthogonal.

Similar to the computation at level 4, the uniformization step that constructs the uniform $\mathcal{H}^1$ representation of $G^{(l)}$ simply by Algorithm 6 to the boxes on level l. The result gives the approximation

$$\widehat{G}(\mathcal{I},\mathcal{J}) = U_{\mathcal{I}} N_{\mathcal{I}\mathcal{J}} U_{\mathcal{J}}^{\mathrm{T}}. \tag{24}$$

Finally in the projection step, one needs to compute an $\mathcal{H}^2$ representation for $G^{(3)} + \cdots + G^{(l)}$. To this end, we apply Algorithm 7 to level l.

The complexity analysis at level l follows exactly the one of level 4. Since we still have exactly 64 sets $\mathcal{S}_{pq}$, the computation again takes $64(r(\varepsilon)+c)$ matvecs along with $\mathcal{O}(N^2)$ extra steps.

These three steps (peeling, uniformization, and projection) are repeated for each level until we reach level L. At this point, we hold the $\mathcal{H}^2$ representation for $G^{(3)} + \cdots + G^{(L)}$.

2.4.4. Computation of $D^{(L)}$

Finally we construct of the diagonal part

$$D^{(L)} = G - (G^{(3)} + \cdots + G^{(L)}). \tag{25}$$

More specifically, for each box $\mathcal{J}$ on level L, we need to compute $G(\mathcal{I},\mathcal{J})$ for $\mathcal{I} \in \mathsf{NL}(\mathcal{J})$.

Define a matrix E of size $N^2 \times (N/2^L)^2$ (recall that the box $\mathcal{J}$ on level L covers $(N/2^L)^2$ grid points) by

$$E(\mathcal{J},:) = I \quad \text{and} \quad E(\mathcal{I}_0 \setminus \mathcal{J},:) = 0,$$

where I is the $(N/2^L)^2 \times (N/2^L)^2$ identity matrix. Applying $G - (G^{(3)} + \cdots + G^{(L)})$ to E and restricting the results to $\mathcal{I} \in \mathsf{NL}(\mathcal{J})$ gives $G(\mathcal{I},\mathcal{J})$ for $\mathcal{I} \in \mathsf{NL}(\mathcal{J})$. However, we can do better as $(G - (G^{(3)} + \cdots + G^{(L)}))E$is only non-zero in $\mathsf{NL}(\mathcal{J})$. Hence, one can combine computation of different boxes $\mathcal{J}$ as long as $\mathsf{NL}(\mathcal{J})$ do not overlap.

To do this, define the following $4 \times 4 = 16$ sets $\mathcal{S}_{pq}$, $1 \leqslant p,q \leqslant 4$

$$\mathcal{S}_{pq} = \{\mathcal{J}_{L,ij} | i \equiv p \,(\mathrm{mod}4),\ j \equiv q \,(\mathrm{mod}4)\}.$$

For each set $\mathcal{S}_{pq}$, construct matrix E by

$$E(\mathcal{J},:) = \begin{cases} I, & \mathcal{J} \in \mathcal{S}_{pq}; \\ 0, & \text{otherwise}. \end{cases}$$

Next, apply $G - (G^{(3)} + \cdots + G^{(L)})$ to E. For each $\mathcal{J} \in \mathcal{S}_{pq}$, restricting the result to $\mathcal{I} \in \mathsf{NL}(\mathcal{J})$ gives $G(\mathcal{I},\mathcal{J})I = G(\mathcal{I},\mathcal{J})$. Repeating this computation for all 16 sets $\mathcal{S}_{pq}$ gives the diagonal part $D^{(L)}$.

Complexity analysis: The dominant computation is for each group $\mathcal{S}_{pq}$ apply G and $G^{(3)} + \cdots + G^{(L)}$ to E, the former takes $(N/2^L)^2$ matvecs and the latter takes $\mathcal{O}((N/2^L)^2 N^2)$ extra steps. Recall by the choice of L, $N/2^L$ is a constant. Therefore, the total cost for 16 sets is $16(N/2^L)^2 = \mathcal{O}(1)$ matvecs and $\mathcal{O}(N^2)$ extra steps.

Let us now summarize the complexity of the whole peeling algorithm. From the above discussion, it is clear that at each level the algorithm spends $64(r(\varepsilon)+c) = \mathcal{O}(1)$ matvecs and $\mathcal{O}(N^2)$ extra steps. As there are $\mathcal{O}(\log N)$ levels, the overall cost of the peeling algorithm is equal to $\mathcal{O}(\log N)$ matvecs plus $\mathcal{O}(N^2 \log N)$ steps.

It is a natural concern that whether the error from low-rank decompositions on top levels accumulates in the peeling steps. As observed from numerical examples in Section 3, it does not seem to be a problem at least for the examples considered. We do not have a proof for this though.

2.5. Peeling algorithm: variants

In this section, we discuss two variants of the peeling algorithm. Let us recall that the above algorithm performs the following three steps at each level l.

1. *Peeling.* Construct an $\mathcal{H}^1$ representation for $G^{(l)}$ using the peeling idea and the $\mathcal{H}^2$ representation for $G^{(3)} + \cdots + G^{(l-1)}$.
2. *Uniformization.* Construct a uniform $\mathcal{H}^1$ representation for $G^{(l)}$ from its $\mathcal{H}^1$ representation.
3. *Projection.* Construct an $\mathcal{H}^2$ representation for $G^{(3)} + \cdots + G^{(l)}$.

As this algorithm constructs the $\mathcal{H}^2$ representation of the matrix G, we also refer to it more specifically as the $\mathcal{H}^2$ version of the peeling algorithm. In what follows, we list two simpler versions that are useful in practice.

- the $\mathcal{H}^1$ version, and
- the uniform $\mathcal{H}^1$ version.

In the $\mathcal{H}^1$ version, we only perform the peeling step at each level. Since this version constructs only the $\mathcal{H}^1$ representation, it will use the $\mathcal{H}^1$ representation of $G^{(3)} + \cdots + G^{(l)}$ in the computation of $(G^{(3)} + \cdots + G^{(l)})R$ within the peeling step at level $l+1$.

In the uniform $\mathcal{H}^1$ version, we perform the peeling step and the uniformization step at each level. This will give us instead the uniform $\mathcal{H}^1$ version of the matrix. Accordingly, one needs to use the uniform $\mathcal{H}^1$ representation of $G^{(3)} + \cdots + G^{(l)}$ in the computation of $(G^{(3)} + \cdots + G^{(l)})R$ within the peeling step at level $l+1$.

These two simplified versions are of practical value since there are matrices that are in the $\mathcal{H}^1$ or the uniform $\mathcal{H}^1$ class but not the $\mathcal{H}^2$ class. A simple calculation shows that these two simplified versions still take $\mathcal{O}(\log N)$ matvecs but requires $\mathcal{O}(N^2 \log^2 N)$ extra steps. Clearly, the number of extra steps is log$N$ times more expensive than the one of the $\mathcal{H}^2$ version. However, if the fast matrix–vector multiplication subroutine itself takes $\mathcal{O}(N^2 \log N)$ steps per application, using the $\mathcal{H}^1$ or the uniform $\mathcal{H}^1$ version does not change the overall asymptotic complexity.

Between these two simplified versions, the uniform $\mathcal{H}^1$ version requires the uniformization step, while the $\mathcal{H}^1$ version does not. This seems to suggest that the uniform $\mathcal{H}^1$ version is more expensive. However, because (1) our algorithm also utilizes the partially constructed representations for the calculation at future levels and (2) the uniform $\mathcal{H}^1$ representation is much faster to apply, the construction of the uniform $\mathcal{H}^1$ version turns out to be much faster. Moreover, since the uniform

$\mathcal{H}^1$ representation stores one $U_{\mathcal{I}}$ matrix for each box $\mathcal{I}$ while the $\mathcal{H}^1$ version stores about 27 of them, the uniform $\mathcal{H}^1$ is much more memory-efficient, which is very important for problems in higher dimensions.

3. Numerical results

We study the performance of the hierarchical matrix construction algorithm for the inverse of a discretized elliptic operator. The computational domain is a two dimensional square $[0,1)^2$ with periodic boundary condition, discretized as an $N \times N$ equispaced grid. We first consider the operator $H = -\Delta + V$ with Δ being the discretized Laplacian operator and the potential being $V(i,j) = 1 + W(i,j)$, $i, j = 1,\ldots,N$. For all (i,j), $W(i,j)$ are independent random numbers uniformly distributed in $[0,1]$. The potential function V is chosen to have this strong randomness in order to show that the existence of $\mathcal{H}$-matrix representation of the Green's function depends weakly on the smoothness of the potential. The inverse matrix of H is denoted by G. The algorithms are implemented using MATLAB. All numerical tests are carried out on a single-CPU machine.

We analyze the performance statistics by examining both the cost and the accuracy of our algorithm. The cost factors include the time cost and the memory cost. While the memory cost is mainly determined by how the matrix G is compressed and does not depend much on the particular implementation, the time cost depends heavily on the performance of matvec subroutine. Therefore, we report both the wall clock time consumption of the algorithm and the number of calls to the matvec subroutine. The matvec subroutine used here is a nested dissection reordered block Gauss elimination method [14]. For an $N \times N$ discretization of the computational domain, this matvec subroutine has a computational cost of $\mathcal{O}(N^2 \log N)$ steps.

Table 1 summarizes the matvec number, and the time cost per degree of freedom (DOF) for the $\mathcal{H}^1$, the uniform $\mathcal{H}^1$ and the $\mathcal{H}^2$ representations of the peeling algorithm. The time cost per DOF is defined by the total time cost divided by the number of grid points N^2. For the $\mathcal{H}^1$ and the uniform $\mathcal{H}^1$ versions, the error criterion ε in Eqs. (13)–(15) are all set to be 10^{-6}.

The number of calls to the matvec subroutine is the same in all three cases (as the peeling step is the same for all cases) and is reported in the third column of Table 1. It is confirmed that the number of calls to matvec increases logarithmically with respect to N if the domain size at level L, *i.e.* $2^{L_M - L}$, is fixed as a constant. For a fixed N, the time cost is not monotonic with respect to L. When L is too small the computational cost of $D^{(L)}$ becomes dominant. When L is too large, the application of the partial representation $G^{(3)} + \cdots + G^{(L)}$ to a vector becomes expensive. From the perspective of time cost, there is an optimal L_{opt} for a fixed N. We find that this optimal level number is the same for $\mathcal{H}^1$, uniform $\mathcal{H}^1$ and $\mathcal{H}^2$ algorithms. Table 1 shows that $L_{\text{opt}} = 4$ for $N = 32,64,128$, $L_{\text{opt}} = 5$ for $N = 256$, and $L_{\text{opt}} = 6$ for $N = 512$. This suggests that for large N, the optimal performance is achieved when the size of boxes on the final level L is 8×8. In other words, $L = L_M - 3$.

The memory cost per DOF for the $\mathcal{H}^1$, the uniform $\mathcal{H}^1$ and the $\mathcal{H}^2$ algorithms is reported in Table 2. The memory cost is estimated by summing the sizes of low-rank approximations as well as the size of $D^{(L)}$. For a fixed N, the memory cost generally decreases as L increases. This is because as L increases, an increasing part of the original dense matrix is represented using low-rank approximations.

Both Tables 1 and 2 indicate that uniform $\mathcal{H}^1$ algorithm is significantly more advantageous than $\mathcal{H}^1$ algorithm, while the $\mathcal{H}^2$ algorithm leads to a further improvement over the uniform $\mathcal{H}^1$ algorithm especially for large N. This fact can be better seen from Fig. 4 where the time and memory cost per DOF for $N = 32,64,128,256,512$ with optimal level number L_{opt} are shown. We remark that since the number of calls to the matvec subroutine are the same in all cases, the time cost difference comes solely from the efficiency difference of the low rank matrix–vector multiplication subroutines.

We measure the accuracy for the $\mathcal{H}^1$, the uniform $\mathcal{H}^1$ and the $\mathcal{H}^2$ representations of G with its actual value using the operator norm (2-norm) of the error matrix. Here, the 2-norm of a matrix is numerically estimated by power method [9] using several random initial guesses. We report both absolute and relative errors. According to Table 3, the errors are well controlled with respect to both increasing N and L, in spite of the more aggressive matrix compression strategy in the uniform

Table 1
matvec numbers and time cost per degree of freedom (DOF) for the $\mathcal{H}^1$, the uniform $\mathcal{H}^1$ and the $\mathcal{H}^2$ representations with different grid point per dimension N and low rank compression level L. The matvec numbers are by definition the same in the three algorithms.

N	L	matvec number	$\mathcal{H}^1$ time per DOF (s)	Uniform $\mathcal{H}^1$ time per DOF (s)	$\mathcal{H}^2$ time per DOF (s)
32	4	3161	0.0106	0.0080	0.0084
64	4	3376	0.0051	0.0033	0.0033
64	5	4471	0.0150	0.0102	0.0106
128	4	4116	0.0050	0.0025	0.0024
128	5	4639	0.0080	0.0045	0.0045
128	6	5730	0.0189	0.0122	0.0125
256	4	7169	0.015	0.0054	0.0050
256	5	5407	0.010	0.0035	0.0033
256	6	5952	0.013	0.0058	0.0057
256	7	7021	0.025	0.0152	0.0154
512	5	8439	0.025	0.0070	0.0063
512	6	6708	0.018	0.0050	0.0044
512	7	7201	0.022	0.0079	0.0072

Table 2
Memory cost per degree of freedom (DOF) for the $\mathcal{H}^1$, the uniform $\mathcal{H}^1$ and the $\mathcal{H}^2$ versions with different grid point per dimension N and low rank compression level L.

N	L	$\mathcal{H}^1$ memory per DOF (MB)	Uniform $\mathcal{H}^1$ memory per DOF (MB)	$\mathcal{H}^2$ memory per DOF (MB)
32	4	0.0038	0.0024	0.0024
64	4	0.0043	0.0027	0.0026
64	5	0.0051	0.0027	0.0026
128	4	0.0075	0.0051	0.0049
128	5	0.0056	0.0029	0.0027
128	6	0.0063	0.0029	0.0027
256	4	0.0206	0.0180	0.0177
256	5	0.0087	0.0052	0.0049
256	6	0.0067	0.0030	0.0027
256	7	0.0074	0.0030	0.0027
512	5	0.0218	0.0181	0.0177
512	6	0.0099	0.0053	0.0049
512	7	0.0079	0.0031	0.0027

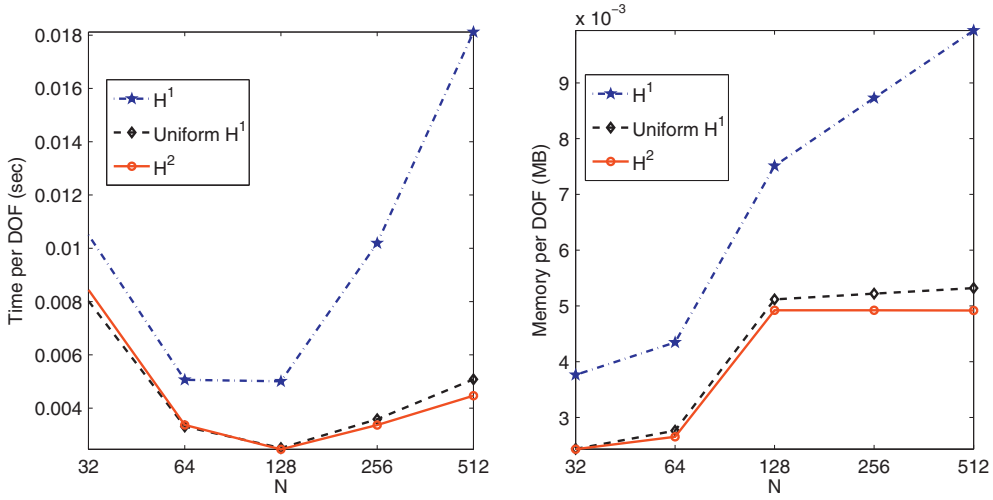

Fig. 4. Comparison of the time and memory costs for the $\mathcal{H}^1$, the uniform $\mathcal{H}^1$ and the $\mathcal{H}^2$ versions with optimal level L_{opt} for $N = 32, 64, 128, 256, 512$. The x-axis (N) is set to be in logarithmic scale.

Table 3
Absolute and relative 2-norm errors for the $\mathcal{H}^1$, the uniform $\mathcal{H}^1$ and the $\mathcal{H}^2$ algorithms with different grid point per dimension N and low rank compression level L. The 2-norm is estimated using power method.

N	L	$\mathcal{H}^1$		Uniform $\mathcal{H}^1$		$\mathcal{H}^2$	
		Absolute error	Relative error	Absolute error	Relative error	Absolute error	Relative error
32 halfline	4	2.16e−07	3.22e−07	2.22e−07	3.31e−07	2.20e−07	3.28e−07
64	4	2.10e−07	3.15e−07	2.31e−07	3.47e−07	2.31e−07	3.46e−07
64	5	1.96e−07	2.95e−07	2.07e−07	3.12e−07	2.07e−07	3.11e−07
128	4	2.16e−07	3.25e−07	2.26e−07	3.39e−07	2.24e−07	3.37e−07
128	5	2.60e−07	3.90e−07	2.68e−07	4.03e−07	2.67e−07	4.02e−07
128	6	2.01e−07	3.01e−07	2.09e−07	3.13e−07	2.08e−07	3.11e−07
256	4	1.78e−07	2.66e−07	1.95e−07	2.92e−07	2.31e−07	3.46e−07
256	5	2.11e−07	3.16e−07	2.26e−07	3.39e−07	2.27e−07	3.40e−07
256	6	2.75e−07	4.12e−07	2.78e−07	4.18e−07	2.30e−07	3.45e−07
256	7	1.93e−07	2.89e−07	2.05e−07	3.08e−07	2.24e−07	3.36e−07
512	5	2.23e−07	3.35e−07	2.33e−07	3.50e−07	1.42e−07	2.13e−07
512	6	2.06e−07	3.09e−07	2.17e−07	3.26e−07	2.03e−07	3.05e−07
512	7	2.67e−07	4.01e−07	2.74e−07	4.11e−07	2.43e−07	3.65e−07

$\mathcal{H}^1$ and the $\mathcal{H}^2$ representations. Moreover, for each box $\mathcal{I}$, the rank $r_U(\varepsilon)$ of the uniform $\mathcal{H}^1$ representation is only slightly larger than the rank $r(\varepsilon)$ of the $\mathcal{H}^1$ representation. This can be seen from Table 4. Here the average rank for a level l is esti-

Table 4
Comparison of the average rank at different levels between the $\mathcal{H}^1$, the uniform $\mathcal{H}^1$, and the $\mathcal{H}^2$ algorithms, for $N = 256$.

l	$\mathcal{H}^1$ average rank	Uniform $\mathcal{H}^1$ average rank	$\mathcal{H}^2$ average rank
4	6	13	13
5	6	13	11
6	6	12	9

Table 5
The number of matvec, and the absolute and relative 2-norm errors for the $\mathcal{H}^2$ representation of the matrix $(-\nabla\cdot(a\nabla)+V)^{-1}$ with $N = 64$, $L = 4$ and two choice of potential function V. The 2-norm is estimated using power method.

Potential	matvec	Absolute error	Relative error
$V(i,j) = 10^{-3}W(i,j)$	4420	5.91e−04	2.97e−07
$V(i,j) = 10^{-6}W(i,j)$	4420	3.60e−03	1.81e−09

mated by averaging the values of $r_U(\varepsilon)$ (or $r(\varepsilon)$) for all low-rank approximations at level l. Note that the rank of the $\mathcal{H}^2$ representation is comparable to or even lower than the rank in the uniform $\mathcal{H}^1$ representation. This is partially due to different weighting choices in the uniformization step and $\mathcal{H}^2$ construction step.

The peeling algorithm for the construction of hierarchical matrix can be applied as well to general elliptic operators in divergence form $H = -\nabla\cdot(a(\mathbf{r})\nabla) + V(\mathbf{r})$. The computational domain, the grids are the same as the example above, and five-point discretization is used for the differential operator. The media is assumed to be high contrast: $a(i,j) = 1 + U(i,j)$, with $U(i,j)$ being independent random numbers uniformly distributed in [0,1]. The potential functions under consideration are (1) $V(i,j) = 10^{-3}W(i,j)$; (2) $V(i,j) = 10^{-6}W(i,j)$. $W(i,j)$ are independent random numbers uniformly distributed in [0,1] and are independent of $U(i,j)$. We test the $\mathcal{H}^2$ version for $N = 64$, $L = 4$, with the compression criterion $\varepsilon = 10^{-6}$. The number of matvec is comparable to that reported in Table 1. The resulting L^2 absolute and relative error of the Green's function are reported in Table 5. The results indicate that the algorithms work well in these cases, despite the fact that the off-diagonal elements of the Green's function have a slower decay than the first example. We also remark that the small relative error for case (2) is due to the large 2-norm of H^{-1} when V is small.

4. Conclusions and future work

In this work, we present a novel algorithm for constructing a hierarchical matrix from its matrix–vector multiplication. One of the main motivations is the construction of the inverse matrix of the stiffness matrix of an elliptic differential operator. The proposed algorithm utilizes randomized singular value decomposition of low-rank matrices. The off-diagonal blocks of the hierarchical matrix are computed through a top-down peeling process. This algorithm is efficient. For an $n \times n$ matrix, it uses only $\mathcal{O}(\log n)$ matrix–vector multiplications plus $\mathcal{O}(n\log n)$ additional steps. The algorithm is also friendly to parallelization. The resulting hierarchical matrix representation can be used as a faster algorithm for matrix–vector multiplications, as well as for numerical homogenization or upscaling.

The performance of our algorithm is tested using two 2D elliptic operators. The $\mathcal{H}^1$, the uniform $\mathcal{H}^1$ and the $\mathcal{H}^2$ versions of the proposed algorithms are implemented. Numerical results show that our implementations are efficient and accurate and that the uniform $\mathcal{H}^1$ representation is significantly more advantageous over $\mathcal{H}^1$ representation in terms of both the time cost and the memory cost, and $\mathcal{H}^2$ representation leads to further improvement for large N.

Although the algorithms presented require only $\mathcal{O}(\log n)$ matvecs, the actual number of matvecs can be quite large (for example, several thousands for the example in Section 3). Therefore, the algorithms presented here might not be the right choice for many applications. However, for computational problems in which one needs to invert the same system with a huge of unknowns or for homogenization problems where analytic approaches do not apply, our algorithm does provide an effective alternative.

The current implementation depends explicitly on the geometric partition of the rectangular domain. However, the idea of our algorithm can be applied to general settings. For problems with unstructured grid, the only modification is to partition the unstructured grid with a quadtree structure and the algorithms essentially require no change. For discretizations of the boundary integral operators, the size of an interaction list is typically much smaller as many boxes contain no boundary points. Therefore, it is possible to design a more effective combination strategy with small number of matvecs. These algorithms can also be extended to the 3D cases in a straightforward way, however, we expect the constant to grow significantly. All these cases will be considered in the future.

Acknowledgement

L.L. is partially supported by DOE under Contract No. DE-FG02-03ER25587 and by ONR under Contract No. N00014-01-1-0674. L.Y. is partially supported by an Alfred P. Sloan Research Fellowship and an NSF CAREER award DMS-0846501. The authors thank Laurent Demanet for providing computing facility and Ming Gu and Gunnar Martinsson for helpful discussions. L.L. and J.L. also thank Weinan E for support and encouragement.

References

[1] S. Börm, L. Grasedyck, W. Hackbusch, Hierarchical matrices, max-Planck-Institute Lecture Notes, 2006.
[2] W. Hackbusch, A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices, Computing 62 (1999) 89–108.
[3] W. Hackbusch, B. Khoromskij, S.A. Sauter, On $\mathcal{H}^2$-matrices, in: Lectures on Applied Mathematics (Munich, 1999), Springer, Berlin, 2000, pp. 9–29.
[4] M. Bebendorf, W. Hackbusch, Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients, Numer. Math. 95 (2003) 1–28.
[5] B. Engquist, O. Runborg, Wavelet-based numerical homogenization with applications, in: Multiscale and Multiresolution Methods, Lect. Notes Comput. Sci. Eng., vol. 20, Springer, Berlin, 2002, pp. 97–148.
[6] N. Halko, P. Martinsson, J. Tropp, Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions, preprint, 2009. Available from: arXiv:0909.4061.
[7] E. Liberty, F. Woolfe, P. Martinsson, V. Rokhlin, M. Tygert, Randomized algorithms for the low-rank approximation of matrices, Proc. Natl. Acad. Sci. USA 104 (2007) 20167–20172.
[8] F. Woolfe, E. Liberty, V. Rokhlin, M. Tygert, A fast randomized algorithm for the approximation of matrices, Appl. Comput. Harmon. Anal. 25 (2008) 335–366.
[9] G. Golub, C. Van Loan, Matrix Computations, third ed., Johns Hopkins Univ. Press, Baltimore, 1996.
[10] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, J. Comput. Phys. 73 (1987) 325–348.
[11] S. Chandrasekaran, M. Gu, W. Lyons, A fast adaptive solver for hierarchically semiseparable representations, Calcolo 42 (3–4) (2005) 171–185.
[12] S. Chandrasekaran, M. Gu, T. Pals, A fast ULV decomposition solver for hierarchically semiseparable representations, SIAM J. Matrix Anal. Appl. 28 (3) (2006) 603–622.
[13] J. Duff, J. Reid, The multifrontal solution of indefinite sparse symmetric linear equations, ACM Trans. Math. Softw. 9 (1983) 302–325.
[14] J. George, Nested dissection of a regular finite element mesh, SIAM J. Numer. Anal. 10 (1973) 345–363.
[15] A. Brandt, Multi-level adaptive solutions to boundary-value problems, Math. Comput. 31 (1977) 333–390.
[16] W. Briggs, V.E. Henson, S.F. McCormick, A Multigrid Tutorial, second ed., Society for Industrial and Applied Mathematics, SIAM, Philadelphia, 2000.
[17] A. Brandt, S. McCormick, J. Ruge, Algebraic multigrid (AMG) for sparse matrix equations, in: Sparsity and Its Applications, Cambridge Univ. Press, Cambridge, 1985, pp. 257–284.
[18] B.F. Smith, P.E. Bjørstad, W.D. Gropp, Domain Decomposition, Cambridge Univ. Press, Cambridge, 1996.
[19] A. Toselli, O. Widlund, Domain Decomposition Methods – Algorithms and Theory, Springer Series in Computational Mathematics, vol. 34, Springer Verlag, Berlin, 2005.
[20] G. Beylkin, R. Coifman, V. Rokhlin, Fast wavelet transforms and numerical algorithms I, Commun. Pure Appl. Math 44 (2) (1991) 141–183.
[21] M. Benzi, C. Meyer, M. Tuma, A sparse approximate inverse preconditioner for the conjugate gradient method, SIAM J. Sci. Comput. 17 (1996) 1135–1149.
[22] S. Chandrasekaran, M. Gu, X.S. Li, J. Xia, Superfast multifrontal method for structured linear systems of equations, Technical Report, LBNL-62897, 2006.
[23] P. Martinsson, A fast direct solver for a class of elliptic partial differential equations, J. Sci. Comput. 38 (2009) 316–330.
[24] W. Hackbusch, Z.P. Nowak, On the fast matrix multiplication in the boundary element method by panel clustering, Numer. Math. 54 (1989) 463–491.
[25] P. Drineas, R. Kannan, M.W. Mahoney, Fast Monte Carlo algorithms for matrices. II: Computing a low-rank approximation to a matrix, SIAM J. Comput. 36 (1) (2006) 158–183.
[26] P. Drineas, R. Kannan, M.W. Mahoney, Fast Monte Carlo algorithms for matrices. III: Computing a compressed approximate matrix decomposition, SIAM J. Comput. 36 (1) (2006) 184–206.
[27] S. Goreinov, E. Tyrtyshnikov, N. Zamarashkin, A theory of pseudoskeleton approximations, Linear Algebra Appl. 261 (1997) 1–21.
[28] M. Mahoney, P. Drineas, CUR matrix decompositions for improved data analysis, Proc. Natl. Acad. Sci. USA 106 (2009) 697–702.
[29] P. Martinsson, Rapid factorization of structured matrices via randomized sampling, preprint, 2008. Available from: arXiv:0806.2339.
[30] H. Owhadi, L. Zhang, Metric-based upscaling, Commun. Pure Appl. Math. 60 (2007) 675–723.