SIAM J. MATRIX ANAL. APPL.
Vol. 42, No. 2, pp. 683–707

EFFICIENT CONSTRUCTION OF AN HSS PRECONDITIONER FOR SYMMETRIC POSITIVE DEFINITE $\mathcal{H}^2$ MATRICES*

XIN XING†, HUA HUANG‡, AND EDMOND CHOW‡

Abstract. In an iterative approach for solving linear systems with dense, ill-conditioned, symmetric positive definite (SPD) kernel matrices, both fast matrix-vector products and fast preconditioning operations are required. Fast (linear-scaling) matrix-vector products are available by expressing the kernel matrix in an $\mathcal{H}^2$ representation or an equivalent fast multipole method representation. This paper is concerned with preconditioning such matrices using the hierarchically semiseparable (HSS) matrix representation. Previously, an algorithm was presented to construct an HSS approximation to an SPD kernel matrix that is guaranteed to be SPD. However, this algorithm has quadratic cost and was only designed for recursive binary partitionings of the points defining the kernel matrix. This paper presents a general algorithm for constructing an SPD HSS approximation. Importantly, the algorithm uses the $\mathcal{H}^2$ representation of the SPD matrix to reduce its computational complexity from quadratic to quasilinear. Numerical experiments illustrate how this SPD HSS approximation performs as a preconditioner for solving linear systems arising from a range of kernel functions.

Key words. symmetric positive definite preconditioner, HSS matrix representation, $\mathcal{H}^2$ matrix representation, kernel matrix

AMS subject classifications. 65F08, 65F55

DOI. 10.1137/20M1365776

1. Introduction.

Fast direct linear solvers exploit the hierarchical low-rank structure of matrix blocks. This structure can be exploited in different ways (e.g., hierarchical off-diagonal low-rank [1], hierarchical semiseparable (HSS) [6, 30], recursive skeletonization [20], hierarchical interpolative factorization [21], inverse fast multipole method [2, 8]), but, invariably, constructing these hierarchical low-rank representations is expensive, its cost being dominated by computing accurate low-rank approximations of matrix blocks and the associated factorizations based on these approximations. Usually, this construction step scales superlinearly and is far more expensive than the subsequent solve step (which may include factorization, e.g., ULV decomposition [30] for the HSS representation).

An alternative to fast direct solvers is to use iterative solvers and fast matrix-vector multiplication provided by more general representations of the hierarchical low-rank structure (e.g., $\mathcal{H}$ [14, 17], $\mathcal{H}^2$ [15, 16], the fast multipole method [12, 13], butterfly factorization [24]). These methods only require relatively cheap or even trivial precomputation to construct the hierarchical low-rank representation and can scale linearly or quasilinearly overall. The main challenge here is slow convergence of the iterative solve for ill-conditioned matrices.

The two approaches above can be combined by using fast direct solvers as preconditioners for the iterative solvers and using fast matrix-vector multiplication. Referred

*Received by the editors September 9, 2020; accepted for publication (in revised form) by S. Le Borne January 11, 2021; published electronically April 27, 2021.
https://doi.org/10.1137/20M1365776
Funding: The work of the authors was supported by National Science Foundation grants ACI-1609842 and ACI-2003683.
†Department of Mathematics, University of California, Berkeley, Berkeley, CA 94720 USA (xxing@berkeley.edu).
‡School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, GA 30332-4017 USA (huangh223@gatech.edu, echow@cc.gatech.edu).

to as *rank-structured preconditioners*, the construction cost of the solvers is greatly reduced due to the lower accuracy required of the low-rank approximations. For symmetric positive definite (SPD) matrices, which are addressed in this paper, it is important that the preconditioner is also SPD. Unfortunately, most rank-structured preconditioners, if only focusing on matrix block approximation, are not able to guarantee that positive definiteness is preserved.

Recently, a *scaling-and-compression* technique has been developed for both dense and sparse SPD matrices to compress matrix blocks into low-rank form as part of the construction of certain rank-structured preconditioners [5, 9, 29, 31, 32, 34]. The resulting preconditioners can be much more effective than if this technique is not used. It has also been found experimentally that preconditioners computed using this technique are more likely to be positive definite. In some cases above, positive definiteness can further be guaranteed when the scaling-and-compression technique is used with the appropriate construction algorithm, but the cost of constructing these SPD preconditioners is at least quadratic for dense SPD matrices.

In this paper, we propose a quasilinear algorithm to efficiently construct an SPD preconditioner in HSS form by accelerating the scaling-and-compression technique, given an $\mathcal{H}^2$ representation of the dense SPD matrix.

The scaling-and-compression technique is illustrated in Figure 1.1 for compressing off-diagonal blocks at one level in the construction process of an HSS representation. The matrix A is partitioned into blocks and the compressed matrix $\tilde{A}$ is produced. In the scaling-and-compression technique, instead of directly compressing each off-diagonal block, the block is scaled before compression. Each off-diagonal block A_{ij} is scaled as $S_i^{-1}A_{ij}S_j^{-T}$, where S_i and S_j are from an easily invertible symmetric factorization (e.g., Cholesky factorization) of the diagonal blocks, $A_{ii} = S_iS_i^T$ and $A_{jj} = S_jS_j^T$. The scaled off-diagonal blocks are then compressed into low-rank form, $S_i^{-1}A_{ij}S_j^{-T} \approx U_{ij}V_{ij}^T$. The final low-rank approximation is $A_{ij} \approx S_iU_{ij}V_{ij}^TS_j^T$.

The scaling step, $A_{ij} \to S_i^{-1}A_{ij}S_j^{-T}$, requires accessing all the matrix entries and leads to quadratic computation cost. If using a fixed approximation rank, the compression of all the scaled off-diagonal blocks, i.e., the step $S_i^{-1}A_{ij}S_j^{-T} \to U_{ij}V_{ij}^T$, using general algebraic methods such as QR decomposition and SVD also takes quadratic computation cost. Thus, both the scaling and compression operations in the scaling-and-compression technique could lead to unfavorable, quadratic HSS construction cost.

A key observation that we utilize in this paper is that if a block is already in low-rank form, its scaling and compression can be efficiently computed (this is utilized in subsection 5.2.2). For example, for $A_{ij} = UV^T$ with tall factors U and V, it is sufficient to compute and compress the two products $S_i^{-1}U$ and $S_j^{-1}V$. If a matrix is

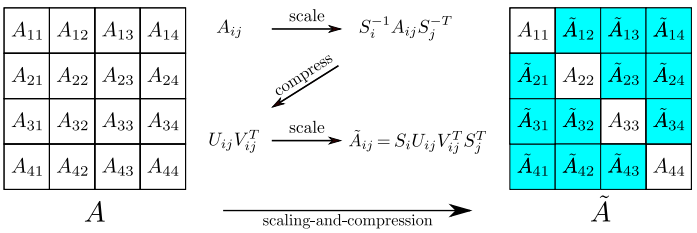


FIG. 1.1. *Illustration of the scaling-and-compression technique for compressing off-diagonal blocks at one level in HSS construction.*

expressed in the $\mathcal{H}^2$ representation, then the vast majority of its off-diagonal blocks is already expressed in low-rank form. This reduces the cost of constructing an HSS representation that uses the scaling-and-compression technique in the construction process. The HSS representation generally requires more of its off-diagonal blocks to be compressed into low-rank form than the $\mathcal{H}^2$ representation. However, the number of additional blocks that need to be compressed in the HSS representation is relatively very small. If a fixed approximation rank is used, these considerations lead to the quasilinear cost of the SPD HSS construction algorithm proposed in this paper. More precisely, if a fixed rank r is used for all HSS block approximations, the new algorithm has computation dominated by $O(r \log N)$ matrix-vector multiplications (using the $\mathcal{H}^2$ representation) and thus scales as $O(rN \log N)$, where N is the number of matrix rows. The scaling and compression of all blocks at one level of the new construction algorithm can be performed in parallel.

There exist related ideas in the current literature. In particular, the construction of the HSS representation and of the butterfly factorization for a matrix can be accelerated if a fast matrix-vector product operation is available for the matrix [11, 25, 26, 28]. One application of these methods is to construct such representations for products of matrices, where each matrix is expressed in an $\mathcal{H}$ or $\mathcal{H}^2$ representation, for example. In a similar spirit, simple rank-structured representations can be post-processed to construct more complicated ones, e.g., converting an $\mathcal{H}$ representation into an $\mathcal{H}^2$ representation [4], by exploiting the efficiencies already afforded by the existing $\mathcal{H}$ representation.

Outline. Previously, a quadratic-scaling algorithm for constructing an SPD HSS approximation was presented [34]. The main concepts behind this algorithm are reviewed in section 3, as the new algorithm of this paper uses the same ideas. The earlier algorithm, however, can only construct HSS representations by recursively partitioning the set of matrix rows (or columns) in binary fashion, leading to a binary partition tree (see Background, section 2). In section 4, we generalize the earlier algorithm to handle nonbinary partition trees. This is a necessary step for our new algorithm because the SPD HSS representation will be derived from an $\mathcal{H}^2$ representation using the same partition tree, and the latter representation can use a nonbinary partition tree. We note that this "generalized" SPD HSS construction algorithm still scales quadratically. In section 5, we propose the new algorithm that uses an $\mathcal{H}^2$ representation of an SPD matrix to accelerate the construction of its SPD HSS approximation, resulting in a quasilinear algorithm. This is the main contribution of this paper. To demonstrate the computational cost of the new algorithm and the utility of the SPD HSS approximation as a preconditioner, the results of numerical experiments are shown in section 6.

2. Background. For an $N \times N$ symmetric matrix A, we denote its row (or column) index set as $I = \{1, 2, \ldots, N\}$. In an applied problem, each index is associated with some element of interest, e.g., a quadrature point, a feature vector, etc. With a recursive partitioning of these elements of interest, the index set I is partitioned into hierarchically enclosed subsets $\{I_i\}_{i \in \mathcal{T}}$, where $\mathcal{T}$ is a *partition tree* that characterizes the recursive partitioning. For each node $i \in \mathcal{T}$, I_i is a subset of I. If i has children $i_1, i_2, \ldots, i_m$, then $I_i = I_{i_1} \cup \cdots \cup I_{i_m}$ and $I_{i_a} \cap I_{i_b} = \emptyset$ for $a \neq b$. Often, $\mathcal{T}$ is chosen to be a binary tree, a quadtree, or an octree associated with the spatial partitioning of the elements of interest in 1-, 2-, or 3-dimensional space, respectively. For simplicity, we assume $\mathcal{T}$ to be a perfect (fully populated in each level) m-ary tree. This assumption can be lifted with minor modifications.

The following notation is used in this paper:

- For $i, j \in \mathcal{T}$, A_{ij} denotes the subblock of A with rows indexed by I_i and columns indexed by I_j.
- The root level of $\mathcal{T}$ is called level L, and the leaf level is called level 1. The levels of the partition tree will be associated with levels in the hierarchical structure of a matrix.
- $\mathrm{lvl}(k)$ denotes the set of nodes in level k of $\mathcal{T}$.
- For node i in level k, we define $i^c = \mathrm{lvl}(k) \setminus \{i\}$, and thus A_{ii^c} denotes the off-diagonal block row of A consisting of all A_{ij} with $j \in i^c$.
- For each nonleaf node i, its children are denoted by $i_1, i_2, \ldots, i_m$.

Low-rank approximation by projection. Given a matrix or matrix block $H \in \mathbb{R}^{n\times s}$, a general approach for compressing H into rank-r form is to compute a tall matrix $V \in \mathbb{R}^{n\times r}$ with orthonormal columns whose column space, $\mathrm{col}(V)$, is close to the *principal column space* of H, i.e., the space spanned by the first r left singular vectors of H. A rank-r approximation can then be written as $H \approx VV^TH$ where VV^T projects each column of H onto $\mathrm{col}(V)$. Such a basis matrix V can be computed by SVD, QR decomposition, randomized methods, etc.

HSS representation. At each level k, an HSS construction algorithm for a matrix A compresses all the off-diagonal blocks A_{ij} with $i \neq j \in \mathrm{lvl}(k)$ into the low-rank form

$$A_{ij} \approx U_i B_{ij} U_j^T, \tag{2.1}$$

where *basis matrix* U_i is shared by all the off-diagonal blocks with rows indexed by I_i, i.e., all blocks in A_{ii^c}, and where U_j^T is similarly shared from the symmetry of A. Assuming U_i has orthonormal columns, *coefficient matrix* B_{ij} can be computed as $U_i^T A_{ij} U_j$. Then the approximation (2.1) projects the columns and rows of A_{ij} onto the column spaces $\mathrm{col}(U_i)$ and $\mathrm{col}(U_j)$, respectively. Matrix U_i captures the principal column space of A_{ii^c} (to compress A_{ii^c}) in a recursive way. If i has children $i_1, \ldots, i_m$, then U_i has the nested form

$$U_i = \begin{bmatrix} U_{i_1} & & \\ & \ddots & \\ & & U_{i_m} \end{bmatrix} R_i \tag{2.2}$$

with *transfer matrix* R_i. An HSS representation consists of (1) dense diagonal blocks A_{ii} associated with leaf nodes and (2) low-rank representations (2.1) of off-diagonal blocks A_{ij} at various levels that are not contained in larger off-diagonal blocks. Such a block A_{ij} is associated with a pair of *sibling* nodes i and j, i.e., nodes i and j have the same parent. Figure 2.1 shows an HSS representation for a binary partition tree.

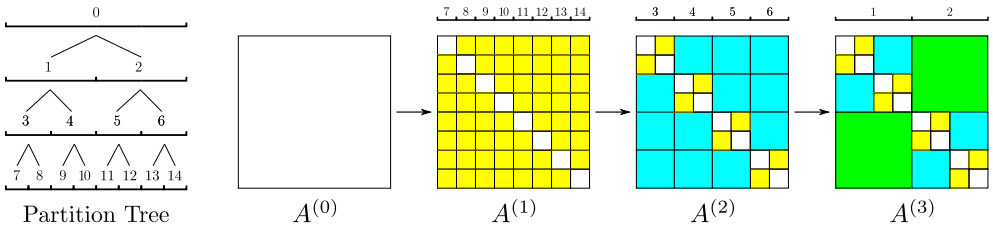


FIG. 2.1. *Recursive construction of an HSS approximation with a binary partition tree with $L = 4$ levels. The colored blocks at different levels are compressed into low-rank form, and $A^{(3)}$ gives an HSS approximation of the original matrix $A^{(0)}$.*

Recursive HSS construction. Constructing an HSS representation starts from the leaf level (level 1) to the level below the root (level $L-1$) of $\mathcal{T}$. At level 1, the original matrix $A^{(0)} = A$ has all its off-diagonal blocks $A_{ij}^{(0)}$ with $i \neq j \in \text{lvl}(1)$ compressed into the low-rank form (2.1) and all its diagonal blocks $A_{ii}^{(0)}$ untouched. This overall approximation to $A^{(0)}$ is denoted as $A^{(1)}$. Recursively, at each level k, $A^{(k-1)}$ from level $(k-1)$ has its off-diagonal blocks $A_{ij}^{(k-1)}$ with $i \neq j \in \text{lvl}(k)$ compressed and is overall approximated by $A^{(k)}$. Lastly, $A^{(L-1)}$ is the HSS representation of A.

Each $A_{ij}^{(k)}$ with $i \neq j \in \text{lvl}(k)$ gives a low-rank approximation of A_{ij} but is constructed *indirectly* by approximating $A_{ij}^{(k-1)}$ and not the original A_{ij}, i.e.,

$$A_{ij} \approx A_{ij}^{(k-1)} \approx A_{ij}^{(k)} = U_i U_i^T A_{ij}^{(k-1)} U_j U_j^T, \quad i \neq j \in \text{lvl}(k).$$

Similarly, basis matrix U_i with $i \in \text{lvl}(k)$ is constructed indirectly by compressing $A_{ii^c}^{(k-1)}$ instead of A_{ii^c}. This helps enforce the nested form (2.2) of U_i.

The recursive HSS construction can be summarized as follows. For levels k from 1 to $L-1$,

(2.3)
$$\begin{aligned} A^{(k)} &= \text{diag}(\{A_{ii}^{(k-1)}\}_{i\in\text{lvl}(k)}) \\ &\quad + \text{diag}(\{U_i U_i^T\}_{i\in\text{lvl}(k)}) \left[A^{(k-1)} - \text{diag}(\{A_{ii}^{(k-1)}\}_{i\in\text{lvl}(k)})\right] \text{diag}(\{U_i U_i^T\}_{i\in\text{lvl}(k)}), \end{aligned}$$

where the notation $\text{diag}(\{H_i\}_{i\in\text{lvl}(k)})$ denotes a block diagonal matrix consisting of all blocks in $\{H_i\}_{i\in\text{lvl}(k)}$. This notation will be simplified as $\text{diag}(H_i)$ with $i \in \text{lvl}(k)$ implied by the context. This recursive construction process is illustrated in Figure 2.1.

3. Review of SPD HSS construction concepts.

In this section, we review the results from [34] that provide the cornerstone for this paper. Specifically, we first show how the scaling-and-compression technique is used with the recursive HSS construction procedure to compress the off-diagonal blocks of $A^{(k-1)}$ to obtain $A^{(k)}$ for each level k. We then explain how this algorithm guarantees that the constructed HSS approximation $A^{(L-1)}$ of A is SPD.

3.1. Scaling-and-compression technique.

Consider the HSS construction at level k that approximates $A^{(k-1)}$ by $A^{(k)}$. Using the scaling-and-compression technique, first compute a symmetric factorization (e.g., Cholesky decomposition) of each diagonal block $A_{ii}^{(k-1)}$ with $i \in \text{lvl}(k)$ as $A_{ii}^{(k-1)} = S_i S_i^T$. Each off-diagonal block $A_{ij}^{(k-1)}$ with $i \neq j \in \text{lvl}(k)$ is then scaled by S_i^{-1} and S_j^{-T} from its left and right, respectively, as

$$A_{ij}^{(k-1)} \xrightarrow{\text{scale}} C_{ij}^{(k-1)} = S_i^{-1} A_{ij}^{(k-1)} S_j^{-T}. \tag{3.1}$$

This is equivalent to multiplying $A^{(k-1)}$ by $\text{diag}(S_i^{-1})$ and $\text{diag}(S_i^{-T})$ from left and right, respectively, making the diagonal blocks of $A^{(k-1)}$ be identity. Next, compress all these scaled off-diagonal blocks $C_{ij}^{(k-1)}$. In this paper, we use the projection approach for compression; see (2.3). Other approaches are possible but may not be able to guarantee that the scaling-and-compression technique helps give SPD approximations. In the projection approach, compute a tall matrix V_i with orthonormal columns to approximate $C_{ii^c}^{(k-1)}$ by $V_i V_i^T C_{ii^c}^{(k-1)}$, and thus compress each $C_{ij}^{(k-1)}$ as

$$C_{ij}^{(k-1)} \xrightarrow{\text{compress}} V_i V_i^T C_{ij}^{(k-1)} V_j V_j^T.$$

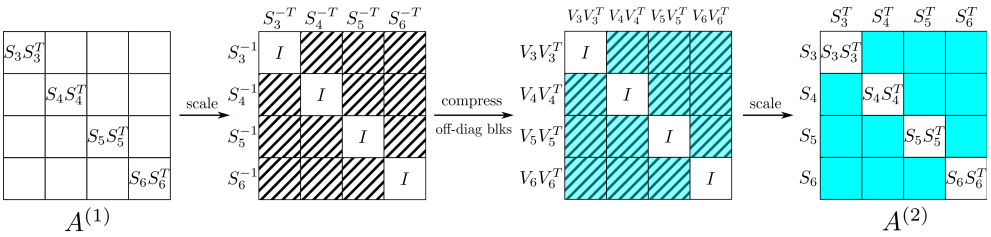


FIG. 3.1. *Illustration of the scaling-and-compression technique to compress off-diagonal blocks of $A^{(1)}$ at level 2 to obtain $A^{(2)}$ for the example in Figure 2.1. Note that the scaling operations are applied to all blocks, and the compression operations are only applied to off-diagonal blocks.*

Lastly, scale these compressed blocks back using S_i and S_j^T to obtain the final low-rank approximation $A_{ij}^{(k)}$ to $A_{ij}^{(k-1)}$ as

$$\begin{aligned} A_{ij}^{(k-1)} \approx A_{ij}^{(k)} &= S_i(V_i V_i^T C_{ij}^{(k-1)} V_j V_j^T) S_j^T \\ &= S_i V_i (V_i^T S_i^{-1} A_{ij}^{(k-1)} S_j^{-T} V_j) V_j^T S_j^T, \end{aligned} \tag{3.2}$$

which we write as

$$A_{ij}^{(k)} = U_i B_{ij} U_j^T, \tag{3.3}$$

where we have defined the basis matrix $U_i = S_i V_i$ and the coefficient matrix $B_{ij} = V_i^T C_{ij}^{(k-1)} V_j = V_i^T S_i^{-1} A_{ij}^{(k-1)} S_j^{-T} V_j$. Thus the same notation as before is used for the basis matrix and the coefficient matrix, regardless of whether the scaling-and-compression technique is used. With scaling and compression, we again require U_i to satisfy the nested form (2.2). For clarity, it is worth comparing the definition (3.3) with the approximation (2.1). Figure 3.1 illustrates the application of the scaling-and-compression technique for compressing $A^{(1)}$ to obtain $A^{(2)}$ at level 2 for the example of Figure 2.1.

3.2. Positive definiteness of $A^{(k)}$. Given an SPD matrix A, to show that the HSS approximation $A^{(L-1)}$ constructed above is SPD, it is sufficient to show that if $A^{(k-1)}$ is SPD, then $A^{(k)}$ is also SPD. To begin, the low-rank approximation (3.2) to an off-diagonal block can be written as

$$A_{ij}^{(k)} = U_i W_i A_{ij}^{(k-1)} W_i^T U_i^T,$$

where we have defined $W_i = V_i^T S_i^{-1}$. Then, the overall approximation at level k is

$$\begin{aligned} A^{(k)} &= \operatorname{diag}(A_{ii}^{(k-1)}) + \operatorname{diag}(U_i W_i)[A^{(k-1)} - \operatorname{diag}(A_{ii}^{(k-1)})]\operatorname{diag}(U_i W_i)^T \\ &= \operatorname{diag}(U_i W_i) A^{(k-1)} \operatorname{diag}(U_i W_i)^T + \operatorname{diag}(A_{ii}^{(k-1)} - U_i W_i A_{ii}^{(k-1)} W_i^T U_i^T) \\ &= \operatorname{diag}(U_i W_i) A^{(k-1)} \operatorname{diag}(U_i W_i)^T + \operatorname{diag}(S_i(I - V_i V_i^T) S_i^T) \end{aligned}$$

which shows that $A^{(k)}$ is at least positive semidefinite. To show that $A^{(k)}$ is SPD, given that $A^{(k-1)}$ is SPD, we prove that $v^T A^{(k)} v > 0$ for any nonzero vector v. Assume $v^T A^{(k)} v = 0$. Since $A^{(k-1)}$ is SPD, we have

$$\operatorname{diag}(U_i W_i)^T v = 0 \quad \text{and} \quad v^T \operatorname{diag}(S_i(I - V_i V_i^T) S_i^T) v = 0.$$

Let v_i denote the subvector of v indexed by I_i. The above two equations can be further written as $S_i^{-T} V_i V_i^T S_i^T v_i = 0$ and $v_i^T S_i (I - V_i V_i^T) S_i^T v_i = 0$, for each node $i \in \operatorname{lvl}(k)$. Plugging $V_i V_i^T S_i^T v_i = 0$ into the latter equation gives $v_i^T S_i S_i^T v_i = 0$ which suggests $v_i = 0$. Thus, $A^{(k)}$ is SPD.

4. Generalized SPD HSS construction. In the following discussion, we assume a fixed rank r for the low-rank approximation of all the off-diagonal blocks in HSS construction. The formal construction in the previous section involves computations with large matrix blocks and leads to $O(N^3)$ computation cost. Its implementation with reduced, $O(N^2 r)$ computation proposed in [34] cannot be applied to nonbinary partition trees. In this section, we generalize this quadratic SPD HSS construction method to general partition trees and retain $O(N^2 r)$ complexity. Section 5 will then demonstrate how to exploit an SPD $\mathcal{H}^2$ representation to reduce the computation cost of the generalized construction method to $O(rN \log N)$.

An HSS approximation has three components: (1) diagonal blocks A_{ii} for each leaf node i, (2) basis matrices U_i for each leaf node i and transfer matrices R_i for each nonleaf node i, and (3) coefficient matrices B_{ij} for each pair of siblings i and j. Note that although only B_{ij} matrices for siblings i and j are used in the final HSS representation, all B_{ij} matrices with any $i \neq j \in \mathrm{lvl}(k)$ are needed during the HSS construction process. With fixed approximation rank r, the matrices U_i, R_i, and B_{ij} are of dimensions $|I_i| \times r$, $mr \times r$, and $r \times r$, respectively.

For each level k from 1 to $L-1$, the following calculations are needed, for $i \neq j \in \mathrm{lvl}(k)$:

- Decomposition: $A_{ii}^{(k-1)} = S_i S_i^T$.
- Scale: $C_{ij}^{(k-1)} = S_i^{-1} A_{ij}^{(k-1)} S_j^{-T}$.
- Compute V_i to approximate $C_{ii^c}^{(k-1)}$ by $V_i V_i^T C_{ii^c}^{(k-1)}$.
- Compute $B_{ij} = V_i^T C_{ij}^{(k-1)} V_j$.
- For leaf levels, compute $U_i = S_i V_i$.
- For nonleaf levels, compute R_i by solving (2.2),

$$R_i = \begin{bmatrix} V_{i_1}^T S_{i_1}^{-1} & & \\ & \ddots & \\ & & V_{i_m}^T S_{i_m}^{-1} \end{bmatrix} S_i V_i \ . \tag{4.1}$$

At the leaf level ($k = 1$), all the matrices in the above calculations are small, and the components of the HSS representation, U_i and B_{ij}, can be computed directly with the above formulas. At nonleaf levels, the matrices S_i, $C_{ij}^{(k-1)}$, and V_i in the calculations can be considered large, with dimension $O(N)$ for levels near the root of the partition tree. However, the HSS components actually needed at each nonleaf level are the small $r \times r$ and $mr \times r$ matrices B_{ij} and R_i.

In the following, we show that these large matrices S_i, $C_{ij}^{(k-1)}$, and V_i at level k can be represented using the matrices $\{B_{ij}\}$ and $\{R_i\}$ previously computed in level $(k-1)$. Further, $\{B_{ij}\}$ and $\{R_i\}$ at level k can be computed directly using $\{B_{ij}\}$ and $\{R_i\}$ from level $(k-1)$. Thus, all calculations involving large matrix blocks can be avoided.

Symmetric decomposition $A_{ii}^{(k-1)} = S_i S_i^T$. For each nonleaf node i at level k with children $i_1, \ldots, i_m$, the block $A_{ii}^{(k-1)}$ can be first split as

$$A_{ii}^{(k-1)} = \begin{bmatrix} A_{i_1 i_1}^{(k-1)} & \cdots & A_{i_1 i_m}^{(k-1)} \\ \vdots & \ddots & \vdots \\ A_{i_m i_1}^{(k-1)} & \cdots & A_{i_m i_m}^{(k-1)} \end{bmatrix} = \begin{bmatrix} S_{i_1} S_{i_1}^T & \cdots & U_{i_1} B_{i_1, i_m} U_{i_m}^T \\ \vdots & \ddots & \vdots \\ U_{i_m} B_{i_m, i_1} U_{i_1}^T & \cdots & S_{i_m} S_{i_m}^T \end{bmatrix}.$$

This matrix can then be decomposed as (using $U_{i_a} = S_{i_a} V_{i_a}$)

$$A_{ii}^{(k-1)} = \begin{bmatrix} S_{i_1} & & \\ & \ddots & \\ & & S_{i_m} \end{bmatrix} \left(I + \mathbf{V}_i \mathbf{B}_{ii} \mathbf{V}_i^T\right) \begin{bmatrix} S_{i_1} & & \\ & \ddots & \\ & & S_{i_m} \end{bmatrix}^T \tag{4.2}$$

with

$$\mathbf{B}_{ii} = \begin{bmatrix} 0 & B_{i_1,i_2} & \dots & B_{i_1,i_m} \\ B_{i_2,i_1} & 0 & \dots & B_{i_2,i_m} \\ \vdots & \vdots & \ddots & \vdots \\ B_{i_m,i_1} & B_{i_m,i_2} & \dots & 0 \end{bmatrix}, \quad \mathbf{V}_i = \begin{bmatrix} V_{i_1} & & \\ & \ddots & \\ & & V_{i_m} \end{bmatrix}.$$

We use bold typeface to denote concatenations of children blocks, e.g., $\mathbf{B}_{ii}$ is made up of children blocks B_{i_a,i_b} from level $(k-1)$.

As can be verified, a symmetric factorization $I + \mathbf{V}_i \mathbf{B}_{ii} \mathbf{V}_i^T = \bar{S}_i \bar{S}_i^T$ exists with

$$\begin{aligned} \bar{S}_i &= I + \mathbf{V}_i((I + \mathbf{B}_{ii})^{1/2} - I)\mathbf{V}_i^T, \\ \bar{S}_i^{-1} &= I + \mathbf{V}_i((I + \mathbf{B}_{ii})^{-1/2} - I)\mathbf{V}_i^T \end{aligned} \tag{4.3}$$

which are derived from a formula in [3]. These are the key equations that we use to generalize the SPD HSS construction method of [34] for binary partition trees to non-binary partition trees. The positive definiteness of $A_{ii}^{(k-1)}$ guarantees the existence of $(I+\mathbf{B}_{ii})^{\pm 1/2}$. Matrix $\mathbf{B}_{ii}$ is of dimension $mr \times mr$, and $(I+\mathbf{B}_{ii})^{\pm 1/2}$ can be computed by the direct eigendecomposition of $\mathbf{B}_{ii}$. A symmetric factorization $A_{ii}^{(k-1)} = S_i S_i^T$ can be formally computed based on (4.2) and (4.3) with

$$S_i = \begin{bmatrix} S_{i_1} & & \\ & \ddots & \\ & & S_{i_m} \end{bmatrix} \bar{S}_i. \tag{4.4}$$

Scaled off-diagonal blocks $C_{ij}^{(k-1)} = S_i^{-1} A_{ij}^{(k-1)} S_j^{-T}$. For nonleaf nodes $i \neq j$ at level k with children $i_1, \dots, i_m$ and $j_1, \dots, j_m$, the quantity $A_{ij}^{(k-1)}$ can be written as

$$\begin{bmatrix} A_{i_1 j_1}^{(k-1)} & \dots & A_{i_1 j_m}^{(k-1)} \\ \vdots & \ddots & \vdots \\ A_{i_m j_1}^{(k-1)} & \dots & A_{i_m j_m}^{(k-1)} \end{bmatrix} = \begin{bmatrix} U_{i_1} & & \\ & \ddots & \\ & & U_{i_m} \end{bmatrix} \begin{bmatrix} B_{i_1 j_1} & \dots & B_{i_1 j_m} \\ \vdots & \ddots & \vdots \\ B_{i_m j_1} & \dots & B_{i_m j_m} \end{bmatrix} \begin{bmatrix} U_{j_1} & & \\ & \ddots & \\ & & U_{j_m} \end{bmatrix}^T,$$

where the middle matrix is denoted as $\mathbf{B}_{ij} \in \mathbb{R}^{mr \times mr}$. Noting its difference from B_{ij}, this bold typeface $\mathbf{B}_{ij}$ consists of children blocks B_{i_a,j_b} in level $(k-1)$. Based on (4.3), (4.4), $U_i = S_i V_i$, and the above equation, the scaled block $C_{ij}^{(k-1)}$ by definition can be computed as

$$\begin{aligned} C_{ij}^{(k-1)} = S_i^{-1} A_{ij}^{(k-1)} S_j^{-T} &= \bar{S}_i^{-1} \mathbf{V}_i \mathbf{B}_{ij} \mathbf{V}_j^T \bar{S}_j^{-T} \\ &= \mathbf{V}_i (I + \mathbf{B}_{ii})^{-1/2} \mathbf{B}_{ij} (I + \mathbf{B}_{jj})^{-1/2} \mathbf{V}_j^T. \end{aligned} \tag{4.5}$$

Calculation of V_i. Recall that we desire to calculate V_i such that $\mathrm{col}(V_i)$ approximates $\mathrm{col}(C_{ii^c}^{(k-1)})$. Further, V_i must satisfy $\mathrm{col}(V_i) \subset \mathrm{col}(C_{ii^c}^{(k-1)})$ in order to

guarantee the nested form of U_i in (2.2); see [34]. From (4.5), each scaled block $C_{ij}^{(k-1)}$ has its column space contained in $\mathrm{col}(\mathbf{V}_i)$. Thus, V_i can be represented by

$$V_i = \mathbf{V}_i \bar{V}_i = \begin{bmatrix} V_{i_1} & & \\ & \ddots & \\ & & V_{i_m} \end{bmatrix} \bar{V}_i, \tag{4.6}$$

where the small matrix $\bar{V}_i \in \mathbb{R}^{mr \times r}$ is computed with orthonormal columns to minimize the error of the required approximation $C_{ii^c}^{(k-1)} \approx V_i V_i^T C_{ii^c}^{(k-1)}$. Noting that all V_{i_a} blocks have orthonormal columns and using (4.5) and (4.6), the minimization problem can be converted as

$$\min_{V_i} \|C_{ii^c}^{(k-1)} - V_i V_i^T C_{ii^c}^{(k-1)}\|_F = \min_{\bar{V}_i} \left\|E_{ii^c} - \bar{V}_i \bar{V}_i^T E_{ii^c}\right\|_F, \tag{4.7}$$

where E_{ii^c} is the horizontal concatenation of all blocks $(I+\mathbf{B}_{ii})^{-1/2}\mathbf{B}_{ij}(I+\mathbf{B}_{jj})^{-1/2}$ with nodes $j \in i^c$. We note that E_{ii^c} is a small matrix of dimension $mr \times (|\mathrm{lvl}(k)|-1)r$. Thus, $\bar{V}_i$ can be directly computed to capture the principal column space of E_{ii^c}. In subsection 5.2.1, we will discuss how to more efficiently compute $\bar{V}_i$.

Calculation of $B_{ij} = V_i^T C_{ij}^{(k-1)} V_j$. Based on the calculations of $C_{ij}^{(k-1)}$ in (4.5) and V_i in (4.6), B_{ij} can be directly computed as

$$\begin{aligned} B_{ij} &= \left(\bar{V}_i^T \mathbf{V}_i^T\right) \mathbf{V}_i (I+\mathbf{B}_{ii})^{-1/2}\mathbf{B}_{ij}(I+\mathbf{B}_{jj})^{-1/2} \mathbf{V}_j^T \left(\mathbf{V}_j \bar{V}_j\right) \\ &= \bar{V}_i^T (I+\mathbf{B}_{ii})^{-1/2}\mathbf{B}_{ij}(I+\mathbf{B}_{jj})^{-1/2} \bar{V}_j, \end{aligned} \tag{4.8}$$

where all matrices in the second equation are of small dimensions. In subsection 5.2.2, we will discuss how to reduce the number of matrices B_{ij} that need to be computed.

Calculation of R_i. Based on the above calculation of S_i in (4.4) and V_i in (4.6), R_i defined by (4.1) can be directly computed as

$$\begin{aligned} R_i &= \begin{bmatrix} V_{i_1}^T S_{i_1}^{-1} & & \\ & \ddots & \\ & & V_{i_m}^T S_{i_m}^{-1} \end{bmatrix} \begin{bmatrix} S_{i_1} & & \\ & \ddots & \\ & & S_{i_m} \end{bmatrix} \bar{S}_i \begin{bmatrix} V_{i_1} & & \\ & \ddots & \\ & & V_{i_m} \end{bmatrix} \bar{V}_i \\ &= (I+\mathbf{B}_{ii})^{1/2} \bar{V}_i, \end{aligned} \tag{4.9}$$

where, again, all matrices in the second equation are of small dimensions.

To summarize, the actual computations needed at level k include the calculation of $(I+\mathbf{B}_{ii})^{\pm 1/2}$ in (4.3), $\bar{V}_i$ in (4.7), B_{ij} in (4.8), and R_i in (4.9). The pseudocode of this generalized SPD HSS construction process based on scaling and compression is shown in Algorithm 4.1. The overall computation and peak storage costs of Algorithm 4.1 are both $O(N^2 r)$, and the constructed SPD HSS representation has $O(Nr)$ storage cost.

5. Accelerated SPD HSS construction with quasilinear computation. The generalized SPD HSS construction algorithm of the previous section has quadratic computation cost. In this section, we show how to reduce the cost to quasilinear if we can utilize an existing $\mathcal{H}^2$ representation of the SPD matrix. Below, we first give necessary background on $\mathcal{H}^2$ representations.

Algorithm 4.1. Generalized SPD HSS construction.

Input: HSS rank r, an SPD matrix A

Output: an SPD HSS approximation with $\{A_{ii}\}, \{B_{ij}\}, \{U_i\}, \{R_i\}$

At the leaf level

compute the Cholesky decomposition $A_{ii} = S_i S_i^T \ \forall i \in \text{lvl}(1)$

compute the scaled off-diagonal block $C_{ij}^{(0)} = S_i^{-1} A_{ij} S_j^{-T} \ \forall i \neq j \in \text{lvl}(1)$

compute $V_i \in \mathbb{R}^{|I_i| \times r}$ satisfying

– V_i has orthonormal columns and $\text{col}(V_i) \subset \text{col}(C_{ii^c}^{(0)})$

– V_i should minimize $\|C_{ii^c}^{(0)} - V_i V_i^T C_{ii^c}^{(0)}\|_F$

compute $B_{ij} = V_i^T C_{ij}^{(0)} V_j \ \forall i \neq j \in \text{lvl}(1)$

set $U_i = S_i V_i \ \forall i \in \text{lvl}(1)$

for $k = 2, 3, \ldots, L-1$ **do**

compute $(I + \mathbf{B}_{ii})^{\pm 1/2}$ via the eigendecomposition of $\mathbf{B}_{ii} \ \forall i \in \text{lvl}(k)$

compute $(I + \mathbf{B}_{ii})^{-1/2} \mathbf{B}_{ij} (I + \mathbf{B}_{jj})^{-1/2} \ \forall i \neq j \in \text{lvl}(k)$

assemble E_{ii^c} in (4.7) and compute $\bar{V}_i$ satisfying

– $\bar{V}_i$ has orthonormal columns and $\text{col}(\bar{V}_i) \subset \text{col}(E_{ii^c})$

– $\bar{V}_i$ should minimize $\|E_{ii^c} - \bar{V}_i \bar{V}_i^T E_{ii^c}\|_F$

compute $B_{ij} = \bar{V}_i^T (I + \mathbf{B}_{ii})^{-1/2} \mathbf{B}_{ij} (I + \mathbf{B}_{jj})^{-1/2} \bar{V}_j \ \forall i \neq j \in \text{lvl}(k)$

set $R_i = (I + \mathbf{B}_{ii})^{1/2} \bar{V}_i \ \forall i \in \text{lvl}(k)$

end for

5.1. $\mathcal{H}^2$ representation. Like the HSS representation, the $\mathcal{H}^2$ representation of a matrix A is based on a partition tree $\mathcal{T}$ and a hierarchical index set $\{I_i\}_{i \in \mathcal{T}}$. For each node i at each level k of a partition tree, we define a node set $\mathcal{F}_i \subset \text{lvl}(k)$ that contains all the nodes in level k that are in the "far field" of node i. More precisely, if the indices are associated with points in space, $\mathcal{F}_i$ can be defined as the set of nodes $j \in \text{lvl}(k)$ such that the points associated with I_j are well separated from the points associated with I_i. In particular, the HSS representation is a specific $\mathcal{H}^2$ representation with $\mathcal{F}_i = i^c = \text{lvl}(k) \setminus \{i\}$.

The node set $\mathcal{F}_i$ specifies which blocks will be compressed in the $\mathcal{H}^2$ representation of A. At each level k, all blocks A_{ij} with $i \in \text{lvl}(k)$ and $j \in \mathcal{F}_i \subset \text{lvl}(k)$ are compressed as

$$A_{ij} = U_i^{\mathcal{H}^2} B_{ij}^{\mathcal{H}^2} (U_j^{\mathcal{H}^2})^T \tag{5.1}$$

assuming that the $\mathcal{H}^2$ representation is exact. Here we use the superscript "$\mathcal{H}^2$" to distinguish the corresponding components of $\mathcal{H}^2$ from those of HSS. Like for HSS, the basis matrix $U_i^{\mathcal{H}^2}$ is shared by all blocks A_{ij} with $j \in \mathcal{F}_i$ and is computed to capture the principal column space of $A_{i\mathcal{F}_i} = [A_{ij}]_{j \in \mathcal{F}_i}$ (corresponding to A_{ii^c} in HSS). Further, $U_i^{\mathcal{H}^2}$ satisfies the nested form (2.2) as well.

An $\mathcal{H}^2$ representation consists of (1) dense blocks A_{ij} with $j \notin \mathcal{F}_i$ at the leaf level and (2) low-rank representations (5.1) of blocks A_{ij} with $j \in \mathcal{F}_i$ at various levels that are not contained in larger low-rank blocks. Such a low-rank block A_{ij} is associated with i, j satisfying the condition $j \in \mathcal{F}_i$ but $\text{par}(j) \notin \mathcal{F}_{\text{par}(i)}$ ($\text{par}(i)$ denotes the parent of i). Figure 5.1 gives an illustration of an $\mathcal{H}^2$ matrix with a binary partition tree.

In practical problems, a proper definition of $\mathcal{F}_i$ can guarantee that all compressed blocks $A_{i\mathcal{F}_i}$ have numerical ranks bounded by a small constant independent of the matrix size and thus the $\mathcal{H}^2$ representation can have linear-scaling matrix-vector multiplications. In the case of HSS, by defining $\mathcal{F}_i = i^c$, the maximum numerical rank

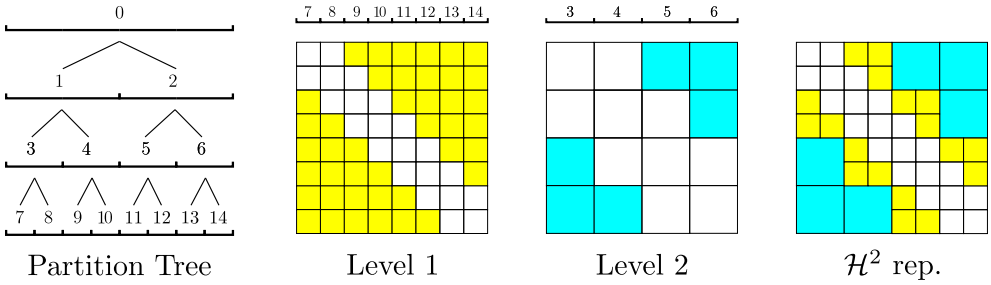


FIG. 5.1. *Illustration of an $\mathcal{H}^2$ representation with a binary partition tree. The colored blocks A_{ij} at different levels satisfy $j \in \mathcal{F}_i$ and are compressed into low-rank form. For each node i, this example defines $\mathcal{F}_i$ as the set of nodes in the same level that are not adjacent to i.*

of all $A_{i\mathcal{F}_i}$ blocks usually increases with the matrix size and thus leads to superlinear complexities in HSS construction and other HSS computations.

5.2. Quasilinear SPD HSS construction. In the generalized SPD HSS construction algorithm of section 4, the computation and storage costs are dominated by those related to the coefficient matrices, B_{ij}. In each level k, there are $|\mathrm{lvl}(k)|(|\mathrm{lvl}(k)|-1)$ such matrices. Each B_{ij} is computed recursively from the leaf level to level k using (4.8) and is ultimately computed from the original matrix block A_{ij}.

Before proceeding, we define Φ_i, which will be used in this section. Observe that, at level k,

$$B_{ij} = V_i^T S_i^{-1} A_{ij}^{(k-1)} S_j^{-T} V_j, \quad i \neq j \in \mathrm{lvl}(k),$$

and that B_{ij} is computed recursively by applying multiple matrices to A_{ij} on its left and right. To emphasize this relationship, we define Φ_i such that

$$B_{ij} = \Phi_i A_{ij} \Phi_j^T, \quad i \neq j \in \mathrm{lvl}(k) \tag{5.2}$$

$$\Phi_i = \begin{cases} V_i^T S_i^{-1}, & i \text{ is a leaf node}, \\ \bar{V}_i^T (I + \mathbf{B}_{ii})^{-1/2} \begin{bmatrix} \Phi_{i_1} & & \\ & \ddots & \\ & & \Phi_{i_m} \end{bmatrix}, & i \text{ has children } i_1, \ldots, i_m, \end{cases} \tag{5.3}$$

where this nested representation of Φ_i is derived from (4.8).

To compute the matrices B_{ij}, the matrices V_i for leaf nodes i and $\bar{V}_i$ for nonleaf nodes i are needed. We discuss how V_i and $\bar{V}_i$ are computed using a randomized algorithm in subsection 5.2.1. The matrix-vector products required in the randomized algorithm are performed efficiently using an $\mathcal{H}^2$ representation of the SPD matrix that we assume to be available.

To compute a B_{ij} matrix at level k, the B_{ij} matrices that are needed from lower levels may already have a low-rank form in the $\mathcal{H}^2$ representation. Thus, the recursion for computing B_{ij} at level k can stop and does not need to proceed to the leaf level. We discuss this in subsection 5.2.2.

5.2.1. Calculation of V_i and $\bar{V}_i$. In the generalized SPD HSS construction algorithm (Algorithm 4.1), V_i and $\bar{V}_i$ are computed as follows. At leaf nodes i, the matrix V_i is computed to approximate $C_{ii^c}^{(0)}$ by $V_i V_i^T C_{ii^c}^{(0)}$ with the constraint that V_i has orthonormal columns and $\mathrm{col}(V_i) \subset \mathrm{col}(C_{ii^c}^{(0)})$. At nonleaf nodes i, the matrix $\bar{V}_i$ is computed to approximate E_{ii^c} by $\bar{V}_i \bar{V}_i^T E_{ii^c}$ with the constraint that $\bar{V}_i$ has orthonormal columns and $\mathrm{col}(\bar{V}_i) \subset \mathrm{col}(E_{ii^c})$.

For the accelerated SPD HSS algorithm, we will compute V_i and $\bar{V}_i$ using a randomized algorithm [18]. (For completeness, we give the randomized algorithm in Algorithm 5.1.) However, instead of using matrix-vector products with $C_{ii^c}^{(0)}$ and E_{ii^c}, respectively, which would be the standard approach, we will use matrix-vector products with alternative matrices that have almost the same column spaces as $C_{ii^c}^{(0)}$ and E_{ii^c}, respectively, to reduce cost.

To see what alternative matrices we propose using, we first write the matrices $C_{ii^c}^{(0)}$ and E_{ii^c} explicitly as

$$\begin{aligned}
C_{ii^c}^{(0)} &= \left[S_i^{-1} A_{ij} S_j^{-T}\right]_{j\in i^c} \\
&= S_i^{-1} A_{ii^c} \operatorname{diag}(\{S_j^{-T}\}_{j\in i^c}), \\
E_{ii^c} &= \left[(I+\mathbf{B}_{ii})^{-1/2}\mathbf{B}_{ij}(I+\mathbf{B}_{jj})^{-1/2}\right]_{j\in i^c} \\
&= (I+\mathbf{B}_{ii})^{-1/2}\left\{\begin{bmatrix} B_{i_1 j_1} & \cdots & Bi_1,j_m \\ \vdots & \ddots & \vdots \\ B_{i_m j_1} & \cdots & B_{i_m j_m}\end{bmatrix}\right\}_{j\in i^c} \operatorname{diag}(\{(I+\mathbf{B}_{jj})^{-1/2}\}_{j\in i^c}) \\
&= (I+\mathbf{B}_{ii})^{-1/2}\begin{bmatrix} \Phi_{i_1} & & \\ & \ddots & \\ & & \Phi_{i_m}\end{bmatrix} A_{ii^c} \operatorname{diag}(\Phi_{j_s}) \operatorname{diag}(\{(I+\mathbf{B}_{jj})^{-1/2}\}_{j\in i^c}),
\end{aligned}$$

where $\operatorname{diag}(\Phi_{j_s})$ denotes the block diagonal matrix made up of all Φ_{j_s} with j_s being a child of any node $j \in i^c$. The last equation above is from substituting (5.2) into $B_{i_a j_b}$ of its previous equation.

Instead of approximating the column spaces of $C_{ii^c}^{(0)}$ (when i is a leaf node) and E_{ii^c} (when i is a nonleaf node), we approximate the column spaces of Λ_{ii^c}, defined as

$$\Lambda_{ii^c} = \begin{cases} S_i^{-1} A_{ii^c}, & i \text{ is a leaf node}, \\ (I+\mathbf{B}_{ii})^{-1/2}\begin{bmatrix} \Phi_{i_1} & & \\ & \ddots & \\ & & \Phi_{i_m}\end{bmatrix} A_{ii^c}, & i \text{ has children } i_1,\ldots,i_m. \end{cases}$$

Block Λ_{ii^c} differs from $C_{ii^c}^{(0)}$ and E_{ii^c} in that there is no matrix applied to the right of A_{ii^c}. This choice of Λ_{ii^c} is for the efficiency of computing the corresponding matrix-vector products in the randomized algorithm. It is theoretically possible that computing V_i and $\bar{V}_i$ using Λ_{ii^c} may affect the approximation accuracy of $C_{ii^c}^{(0)} \approx V_i V_i^T C_{ii^c}^{(0)}$ and $E_{ii^c} \approx V_i V_i^T E_{ii^c}$. Since our goal is to construct a low-accuracy SPD HSS preconditioner, this possible slight deterioration of the approximation accuracy may be tolerable.

The product of Λ_{ii^c} and a block of random vectors involves first computing the product of A_{ii^c} with random vectors. Thus we first compute the products,

Algorithm 5.1. Randomized algorithm for computing a basis matrix U for H.

Input: $H \in \mathbb{R}^{n\times m}$, rank r, oversampling parameter p

Output: U from the rank-r approximation $H \approx UU^T H$ with $U^T U = I$

Step 1: Generate an $m\times(p+r)$ random matrix Ω whose entries follow the standard normal distribution

Step 2: Compute $\Psi = H\Omega$

Step 3: Compute the pivoted QR decomposition $\Psi P = QR$, and set U to be the first r columns of Q

$$Y^{(k)} = \left(A - \mathrm{diag}(\{A_{ii}\}_{i\in \mathrm{lvl}(k)})\right)\Omega, \quad k = 1, 2, \ldots, L-1, \tag{5.4}$$

(one for each nonroot level) where $\Omega \in \mathbb{R}^{N\times(r+p)}$ is a random matrix, given that we desire rank-r approximations using an oversampling parameter p. The quantity in the outer brackets of (5.4) is just the matrix A without its block diagonal part at each level k. These products $Y^{(k)}$ can be computed efficiently using the $\mathcal{H}^2$ representation of A and just neglecting the block diagonal parts during multiplication. The desired products $A_{ii^c}\Omega_{i^c}$, where Ω_{i^c} denotes the row subset of Ω associated with i^c, can be extracted as the row subsets of $Y^{(k)}$ associated with each $i \in \mathrm{lvl}(k)$, denoted by $Y_i^{(k)}$.

To complete the multiplication by Λ_{ii^c}, we now apply S_i^{-1} (if i is a leaf node) or $(I + \mathbf{B}_{ii})^{-1/2}\mathrm{diag}(\{\Phi_{i_1}, \ldots, \Phi_{i_m}\})$ (if i is a nonleaf node) to $Y_i^{(k)}$ to obtain the product $\Lambda_{ii^c}\Omega_{i^c}$ needed in step 2 of Algorithm 5.1. The product $S_i^{-1}Y_i^{(1)}$ for each leaf node can be directly computed. The product $(I + \mathbf{B}_{ii})^{-1/2}\mathrm{diag}(\{\Phi_{i_1}, \ldots, \Phi_{i_m}\})Y_i^{(k)}$ for each nonleaf node at level k needs to be recursively computed from level 1 to level $(k-1)$, since Φ_{i_s} is recursively defined in (5.3). This recursive computation can be unfolded into local computations at each descendant of node i from level 1 to level $(k-1)$ as shown in Algorithm 5.2.

The complexity of computing each $Y^{(k)}$ in (5.4) is $O((r+p)N)$ due to the linear-scaling of $\mathcal{H}^2$ matrix-vector multiplication. Since there are a logarithmic number of levels, the overall complexity for the randomized algorithm is $O(rN\log N)$ for both computation and storage, assuming p is a small constant. The cost of the pivoted QR decompositions in the randomized algorithm is small because $\Lambda_{ii^c}\Omega_{i^c}$ is a small matrix of dimension $|I_i| \times (r+p)$ for a leaf node and $mr \times (r+p)$ for a nonleaf node.

Algorithm 5.2. Level-by-level computation of the special products.

Input: $\{S_i, V_i\}$ at level 1, $\{\mathbf{B}_{ii}, \bar{V}_i\}$ at levels $2, \ldots, k-1$, $\{\mathbf{B}_{ii}\}$ at level k

Output: $(I + \mathbf{B}_{ii})^{-1/2}\mathrm{diag}(\{\Phi_{i_1}, \ldots, \Phi_{i_m}\})Y_i^{(k)}$ for each $i \in \mathrm{lvl}(k)$

At the leaf level

Let $T_i = Y_i^{(k)}$ for each $i \in \mathrm{lvl}(1)$ be the row subset of $Y^{(k)}$ indexed by I_i

Compute $T_i = V_i^T S_i^{-1} T_i$ (in-place computation)

for $l = 2, 3, \ldots, k-1$ **do**

Let $T_i = [T_{i_1}^T, \ldots, T_{i_m}^T]^T$ $\forall i \in \mathrm{lvl}(l)$ be the vertical concatenation of $T_{i_1}, \ldots, T_{i_m}$

Compute $T_i = \bar{V}_i^T (I + \mathbf{B}_{ii})^{-1/2} T_i$

end for

Define $(I + \mathbf{B}_{ii})^{-1/2}[T_{i_1}^T, \ldots, T_{i_m}^T]^T$ for each $i \in \mathrm{lvl}(k)$ as the output

5.2.2. Calculation of B_{ij}. We first define some nomenclature for the blocks in an $\mathcal{H}^2$ representation. At each level k, we categorize all the blocks $\{A_{ij}\}$ with $i, j \in \mathrm{lvl}(k)$ into three types as follows. The colors for each type refer to the colors in Figure 5.2 which illustrates the categorization.

- Type-1 (white): $j \notin \mathcal{F}_i$.
- Type-2 (yellow): $j \in \mathcal{F}_i$ and A_{ij} is contained in a larger low-rank block at some upper level, i.e., $\mathrm{par}(j) \in \mathcal{F}_{\mathrm{par}(i)}$.
- Type-3 (green): $j \in \mathcal{F}_i$, and A_{ij} is represented in low-rank form, i.e., $\mathrm{par}(j) \notin \mathcal{F}_{\mathrm{par}(i)}$.

Type-1 blocks are either stored in dense form or consist of Type-1 and Type-3 blocks at the next lower level. Type-2 blocks are contained in larger Type-3 blocks.

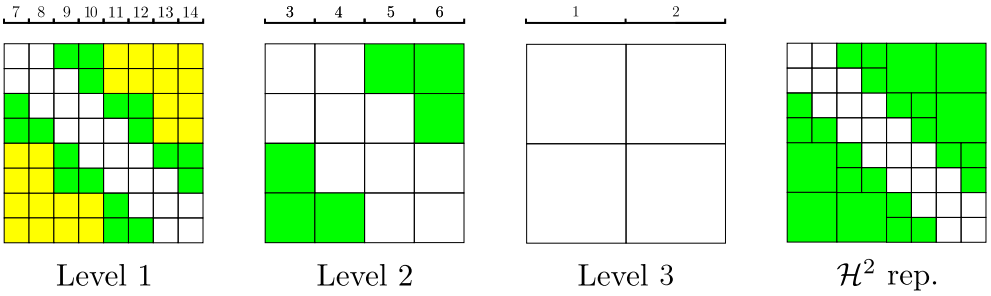


FIG. 5.2. *Illustration of three types of blocks at each partition level of the matrix from Figure* 5.1. *The white blocks are Type-*1*, the yellow blocks are Type-*2*, and the green blocks are Type-*3*. An* $\mathcal{H}^2$ *representation is made up of Type-*1 *blocks from the leaf level and Type-*3 *blocks from all levels.*

Recall from (4.8) that, in the generalized SPD HSS construction algorithm, B_{ij} at a nonleaf level is recursively calculated using

$$B_{ij} = \bar{V}_i^T (I + \mathbf{B}_{ii})^{-1/2} \mathbf{B}_{ij} (I + \mathbf{B}_{jj})^{-1/2} \bar{V}_j.$$

We now discuss how the $\mathcal{H}^2$ representation can be used to reduce the number of B_{ij} matrices that need to be calculated in HSS construction. There are three cases, corresponding to the three types of blocks.

Case 1. If we need the HSS coefficient matrix $B_{ij} = \Phi_i A_{ij} \Phi_j^T$ and the corresponding A_{ij} is a Type-3 block in the $\mathcal{H}^2$ representation, i.e.,

$$A_{ij} = U_i^{\mathcal{H}^2} B_{ij}^{\mathcal{H}^2} (U_j^{\mathcal{H}^2})^T$$

(from (5.1)), then B_{ij} can be computed as

$$B_{ij} = (\Phi_i U_i^{\mathcal{H}^2}) B_{ij}^{\mathcal{H}^2} (\Phi_j U_j^{\mathcal{H}^2})^T. \tag{5.5}$$

Thus, instead of recursively computing B_{ij}, we can first compute $\Phi_i U_i^{\mathcal{H}^2}$ for each i and use (5.5) to immediately compute B_{ij}, which only contains products of small matrices. By this approach, we do not have to compute any $B_{i_a j_b}$ for the descendants i_a of i and j_b of j at lower levels. Any B_{ij} in the HSS representation that corresponds to a Type-2 block A_{ij} in the $\mathcal{H}^2$ representation is no longer needed since Type-2 blocks are enclosed in Type-3 blocks.

For each node i, $\Phi_i U_i^{\mathcal{H}^2}$ can be recursively computed as (utilizing (2.2))

$$\begin{aligned}
\Phi_i U_i^{\mathcal{H}^2} &= \bar{V}_i^T (I + \mathbf{B}_{ii})^{-1/2} \begin{bmatrix} \Phi_{i_1} & & \\ & \ddots & \\ & & \Phi_{i_m} \end{bmatrix} \begin{bmatrix} U_{i_1}^{\mathcal{H}^2} & & \\ & \ddots & \\ & & U_{i_m}^{\mathcal{H}^2} \end{bmatrix} R_i^{\mathcal{H}^2} \\
&= \bar{V}_i^T (I + \mathbf{B}_{ii})^{-1/2} \begin{bmatrix} \Phi_{i_1} U_{i_1}^{\mathcal{H}^2} & & \\ & \ddots & \\ & & \Phi_{i_m} U_{i_m}^{\mathcal{H}^2} \end{bmatrix} R_i^{\mathcal{H}^2},
\end{aligned}$$

which involves computations of $\{\Phi_{i_p} U_{i_p}^{\mathcal{H}^2}\}$ at level $(k-1)$.

Case 2. If we need the HSS coefficient matrix B_{ij} and the corresponding A_{ij} is a Type-1 block not at the leaf level, then B_{ij} must be computed by recursion, using

children blocks in the $\mathcal{H}^2$ representation that are either Type-1 or Type-3. If they are Type-3, then the recursion stops (we have the case above), but if they are Type-1, then the recursion continues unless the Type-1 block is stored in dense format (i.e., at the leaf level, such a B_{ij} is directly computed).

Case 3. The case that we need the HSS coefficient matrix B_{ij} and the corresponding A_{ij} is a Type-2 block is impossible (see the comment on Type-2 blocks in Case 1); such B_{ij} are never needed.

Overall, at each level k of the accelerated SPD HSS construction, we only need to construct the small subset of all B_{ij} blocks that are associated with either Type-1 or Type-3 blocks A_{ij}. There are in total only $O(|\mathrm{lvl}(k)|)$ such blocks at level k. Finally, we only require B_{ij} for each pair of sibling nodes for the final HSS representation. The computation of all such B_{ij} is $O(Nr^2)$.

5.2.3. Summary. The complete algorithm that exploits an $\mathcal{H}^2$ representation to efficiently construct an SPD HSS approximation is shown in Algorithm 5.3. Note that only B_{ij} corresponding to Type-3 and Type-1 blocks need to be computed. The computation proceeds level-by-level from the leaves toward the root in order to satisfy the data dependencies implicit in B_{ij}, $\mathbf{B}_{ii}$, and Φ_i.

In the algorithm, the major computation and storage costs come from those related to $Y^{(k)}$. Algorithm 5.3 thus has $O(rN \log N)$ computation and peak storage cost. The constructed SPD HSS approximation has $O(Nr)$ storage cost.

Algorithm 5.3 can be extended to construct an SPD HSS approximation with a given approximation error threshold by adaptively adding more vectors to Ω and by compressing $\Lambda_{ii^c}\Omega_{i^c}$ with this error threshold. In this case, the approximation ranks for $\Lambda_{ii^c}\Omega_{i^c}$ could increase with the overall matrix size for many problems, leading to more expensive computation cost.

6. Numerical experiments. The SPD HSS approximation constructed by Algorithm 5.3 will be denoted as "SPDHSS." In comparison, the regular HSS representation that neither uses the scaling-and-compression technique nor considers positive definiteness will be referred to as "regular HSS" or, simply, "HSS" in the tables and figures below.

SPDHSS is tested using SPD kernel matrices. A kernel matrix $K(X,X) = (K(x_i,x_j))_{x_i,x_j\in X}$ is defined by a kernel function $K(x,y)$ and a set of points X. Kernel matrices appear in many applications, such as Gaussian processes and Brownian dynamics, and usually can be effectively represented in $\mathcal{H}^2$ form when defined in low-dimensional spaces, e.g., two-dimensional and three-dimensional (3D) spaces. We consider four kernel functions:

- Matérn-3/2 kernel, $K(x,y) = \left(1+\sqrt{3}\,l\,|x-y|\right)\exp\left(-\sqrt{3}\,l\,|x-y|\right)$.
- Gaussian kernel, $K(x,y) = \exp\left(-l\,|x-y|^2\right)$.
- Inverse multiquadric (IMQ) kernel, $K(x,y) = 1/\sqrt{1+l\,|x-y|^2}$.
- Rotne–Prager–Yamakawa (RPY) kernel [27, 36],

$$K(x,y) = \begin{cases} \dfrac{1}{a}I_3 & \text{if } |r| = 0, \\ \dfrac{3}{4|r|}\left(I_3 + \dfrac{rr^T}{|r|^2}\right) + \dfrac{3a^2}{2|r|^3}\left(\dfrac{1}{3}I_3 - \dfrac{rr^T}{|r|^2}\right) & \text{if } |r| \geqslant 2a, \\ \dfrac{1}{a}\left(1 - \dfrac{9}{32}\dfrac{|r|}{a}\right)I_3 + \dfrac{3}{32}\dfrac{|r|}{a}\dfrac{rr^T}{|r|^2} & \text{if } |r| < 2a \end{cases}$$

with $r = x - y$. The kernel is a 3×3 tensor and is defined for points in three dimensions.

Algorithm 5.3. Accelerated SPD HSS construction with quasilinear computation.

Input: HSS rank r, oversampling parameter p, an SPD $\mathcal{H}^2$ representation of A
Output: an SPD HSS approximation of A with $\{A_{ii}\}, \{B_{ij}\}, \{U_i\}, \{R_i\}$

compute $Y^{(k)} = (A - \mathrm{diag}(\{A_{ii}\}_{i \in \mathrm{lvl}(k)}))\Omega, \quad k = 1, 2, \ldots, L-1$
At the leaf level
compute the Cholesky decomposition $A_{ii} = S_i S_i^T \ \forall i \in \mathrm{lvl}(1)$
compute $\Lambda_{ii^c}\Omega_{i^c} = S_i^{-1} Y_i^{(1)} \ \forall i \in \mathrm{lvl}(1)$
compute V_i via the pivoted QR decomposition of $\Lambda_{ii^c}\Omega_{i^c}$ by Algorithm 5.1
compute $\Phi_i U_i^{\mathcal{H}^2} = V_i^T S_i^{-1} U_i^{\mathcal{H}^2} \ \forall i \in \mathrm{lvl}(1)$
compute $B_{ij} = V_i^T S_i^{-1} A_{ij} S_j^{-T} V_j^T$ for all Type-1 $i, j \in \mathrm{lvl}(1)$
compute $B_{ij} = (\Phi_i U_i^{\mathcal{H}^2}) B_{ij}^{\mathcal{H}^2} (\Phi_j U_j^{\mathcal{H}^2})^T$ for all Type-3 $i, j \in \mathrm{lvl}(1)$
set $U_i = S_i V_i \ \forall i \in \mathrm{lvl}(1)$
for $k = 2, 3, \ldots, L-1$ **do**
compute $(I + \mathbf{B}_{ii})^{\pm 1/2}$ via eigendecomposition of $\mathbf{B}_{ii}$, $\forall i \in \mathrm{lvl}(k)$
compute

$$\Lambda_{ii^c}\Omega_{i^c} = (I + \mathbf{B}_{ii})^{-1/2} \begin{bmatrix} \Phi_{i_1} & & \\ & \ddots & \\ & & \Phi_{i_m} \end{bmatrix} Y_i^{(k)}$$

via Algorithm 5.2 $\forall i \in \mathrm{lvl}(k)$
compute $\bar{V}_i$ via the pivoted QR decomposition of $\Lambda_{ii^c}\Omega_{i^c}$ by Algorithm 5.1
compute

$$\Phi_i U_i^{\mathcal{H}^2} = \bar{V}_i^T (I + \mathbf{B}_{ii})^{-1/2} \begin{bmatrix} \Phi_{i_1} U_{i_1}^{\mathcal{H}^2} & & \\ & \ddots & \\ & & \Phi_{i_m} U_{i_m}^{\mathcal{H}^2} \end{bmatrix} R_i^{\mathcal{H}^2} \ \forall i \in \mathrm{lvl}(k)$$

compute $B_{ij} = \bar{V}_i^T (I + \mathbf{B}_{ii})^{-1/2} \mathbf{B}_{ij} (I + \mathbf{B}_{jj})^{-1/2} \bar{V}_j$, for all Type-1 $i, j \in \mathrm{lvl}(k)$
compute $B_{ij} = (\Phi_i U_i^{\mathcal{H}^2}) B_{ij}^{\mathcal{H}^2} (\Phi_j U_j^{\mathcal{H}^2})^T$ for all Type-3 $i, j \in \mathrm{lvl}(k)$
set $R_i = (I + \mathbf{B}_{ii})^{1/2} \bar{V}_i \ \forall i \in \mathrm{lvl}(k)$
end for

The first three kernels are commonly used in statistical models with spatial data, such as Gaussian processes for geoscience problems [19], as well as many other numerical methods that rely on radial basis functions, such as in the numerical solution of partial differential equations [10]. In these kernels, l is a length-scale parameter that is optimized to fit the data. The RPY kernel describes the hydrodynamic interactions between spherical particles in a viscous fluid. In this kernel, a is the particle radius. In practice, the parameters l and a in these kernel functions and the distribution of the data points affect the conditioning of the resulting kernel matrices.

For all tests in this section, we consider two types of 3D point sets for X: uniform random distributions of N points on a sphere of radius $\sqrt{N/(4\pi)}$ in three dimensions (*sphere* point set) and uniform random distributions of N points in a ball of radius $\sqrt[3]{3N/(4\pi)}$ in three dimensions (*ball* point set). The radii of the sphere and the ball are selected to make the point density on the sphere and in the ball remain constant with different N.

We use the H2Pack library [22] for general computations related to $\mathcal{H}^2$ and HSS representations of kernel matrices. H2Pack can efficiently construct an $\mathcal{H}^2$ representation of a kernel matrix with linear-scaling computation by using a hybrid compression

technique called the proxy point method [35]. H2Pack also provides efficient *regular* HSS construction for kernel matrices using the proxy point method. This regular HSS construction [33] does not use the scaling-and-compression technique and instead exploits analytic information of a kernel function to reduce the construction cost. It resembles recursive skeletonization [20] but works for general kernel matrices and requires an additional ULV decomposition for matrix inversion. Note that the ULV decomposition has relatively cheap computation cost compared to the corresponding HSS construction. All timings of regular HSS and SPDHSS construction reported below include those for ULV decomposition.

Given a kernel matrix $K(X, X)$ with X in d-dimensional space, a 2^d-ary partition tree is constructed by recursively partitioning a box enclosing all the points X (by bisecting each dimension) until each finest box has fewer than 400 points. This partition tree is used to construct the $\mathcal{H}^2$ representation of the kernel matrix. The regular HSS and SPDHSS representations use the same partition tree. The preconditioned conjugate gradient (PCG) method is used to solve kernel matrix systems. The systems have random right-hand side vectors with entries chosen from the uniform distribution on $[-0.5, 0.5]$. The PCG relative residual norm stopping threshold is 10^{-4}.

The test calculations are carried out on a dual Intel Xeon Gold 6226 CPU computer with a total of 24 cores and 180 GB memory. One hyperthread per core is used. All codes are implemented in C and parallelized using OpenMP.

6.1. Computational efficiency. Consider the Matérn kernel with parameter $l = 0.1$. Ball and sphere point sets are generated with the number of points N ranging from 4×10^4 to 2.56×10^6. Figure 6.1 plots the timings for constructing $\mathcal{H}^2$, SPDHSS, and regular HSS representations, as well as timings for $\mathcal{H}^2$ matrix-vector multiplication and the SPDHSS/HSS solve operation (with a ULV decomposition). The $\mathcal{H}^2$ representations are constructed with relative error threshold 10^{-8} here and in the results that follow. The SPDHSS/HSS approximations use fixed $r = 100$ and $r = 200$.

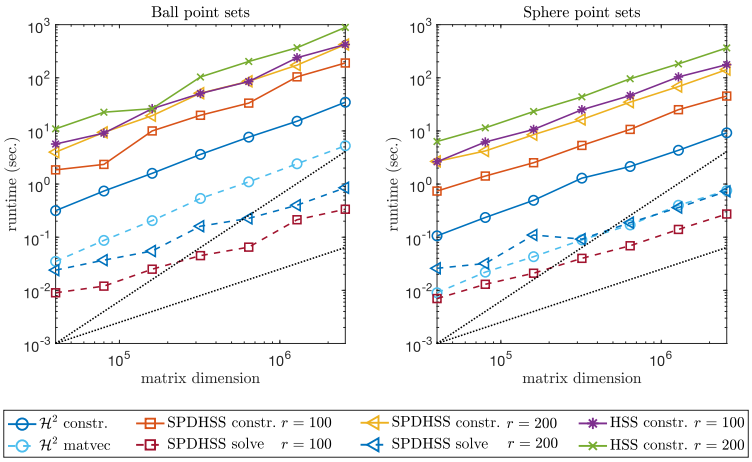


FIG. 6.1. *Timings for constructing (solid lines) and applying (dashed lines) $\mathcal{H}^2$ representations and HSS/SPDHSS approximations. Results are for Matérn kernel matrices with ball and sphere point sets. Linear scaling of the regular HSS construction is due to the fixed approximation rank and the use of the proxy point method. Dotted reference lines show linear and quadratic scaling.*

Approximate linear scaling is observed in all cases. In terms of absolute cost, note that SPDHSS construction formally requires $(r+p)(L-1)$ matrix-vector multiplications using the $\mathcal{H}^2$ representation to compute $Y^{(k)}$ in (5.4), not to mention other operations. For our range of N, the number of levels L ranges from 4 to 7. The oversampling parameter p is set to 10 for all tests. Despite requiring $(r+p)(L-1)$ matrix-vector multiplications, Figure 6.1 shows that the SPDHSS construction time can be faster than r times the cost of a single $\mathcal{H}^2$ matrix-vector multiplication. This is due to the use of level 3 BLAS operations when performing these multiplications and points to the computational efficiency of blocked matrix multiplication that can be used in randomized algorithms.

Figure 6.1 also shows that SPDHSS construction is faster than regular HSS construction with the same rank r. This is due to efficient use of the $\mathcal{H}^2$ representation for SPDHSS construction as presented in this paper. Note that the $\mathcal{H}^2$ construction cost is relatively very small. We also note that the cost of the SPDHSS solves is comparable to or smaller than the cost of a $\mathcal{H}^2$ matrix-vector multiplication in these examples.

Figure 6.2 plots the storage costs of the $\mathcal{H}^2$ representation and SPDHSS approximation (after ULV decomposition). As can be shown analytically [30], an HSS approximation using a fixed rank has linearly scaling storage cost. An SPDHSS approximation has the same storage cost as the corresponding regular HSS approximation for the same approximation rank r.

To estimate the accuracy of the regular HSS and SPDHSS approximations, we measure the accuracy of sample matrix-vector multiplications with these approximations, where we assume that the matrix-vector multiplication with the $\mathcal{H}^2$ representation is the exact value. Figure 6.3 plots the average relative error using a sample of 10 matrix-vector multiplications by Gaussian random vectors. As expected in 3D problems, with a fixed rank r, the relative errors of both the SPDHSS and regular HSS approximations increase with the problem size. SPDHSS has slightly larger approximation errors than regular HSS. However, the regular HSS approximations in all these examples are not SPD.

6.2. Preconditioning performance. We test the SPDHSS approximation as a preconditioner and compare it with the following preconditioners.

- The block Jacobi preconditioner (BJ) is a block diagonal matrix consisting of the diagonal blocks associated with the leaf nodes in the partition tree.

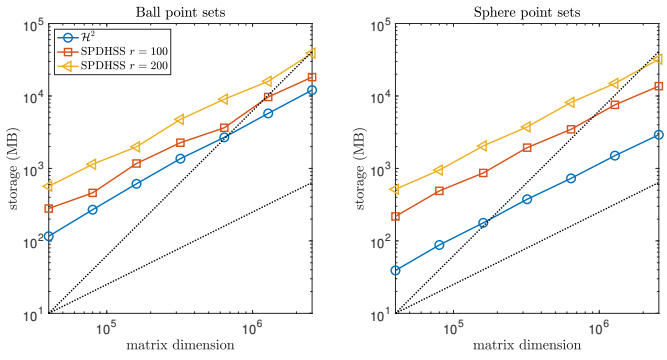


FIG. 6.2. *Storage cost of $\mathcal{H}^2$ representations and SPDHSS approximations of Matérn kernel matrices with ball and sphere point sets. Dotted reference lines show linear and quadratic scaling.*

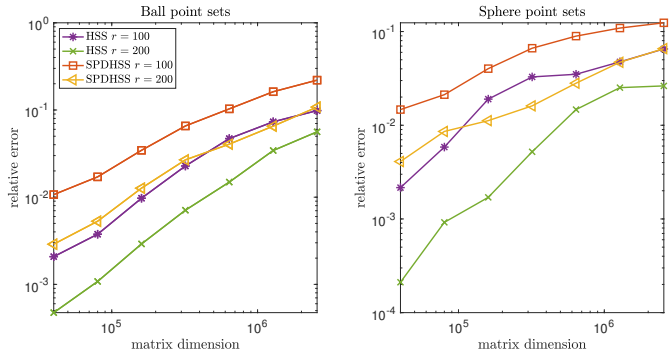


FIG. 6.3. *Average relative matrix-vector multiplication errors of regular HSS and SPDHSS approximations. Each data point is the average relative error (in* 2*-norm) of matrix-vector multiplications by* 10 *Gaussian random vectors. The reference results for matrix-vector multiplication are computed using the* $\mathcal{H}^2$ *representation.*

- The factorized sparse approximate inverse preconditioner (FSAI) is $G^T G$, where G is a sparse approximation to the inverted Cholesky factor of an SPD matrix [23]. The nonzero pattern used for row i of G has k nonzero entries and corresponds to the k nearest neighbors of point i for scalar kernels, or to the $k/3$ nearest neighbors of point i for the RPY kernel. Constructing G requires only selected entries of the SPD matrix, which depends on the chosen sparsity pattern, and thus FSAI can be efficient for dense kernel matrices.
- The regular HSS approximation with a fixed rank r, but only when the approximation happens to be SPD.

6.2.1. Kernel functions with varying parameters. We consider the Matérn, Gaussian, and IMQ kernel functions with varying parameter l. In Gaussian process estimation, l changes in each optimization step and each l corresponds to a system to solve involving the kernel matrix, denoted here as $K_l(X, X)$. For all three kernel functions, when l is close to zero, $K_l(X, X)$ is close to low-rank; when l is sufficiently large, $K_l(X, X)$ is close to sparse. In practice, a diagonal shift is added, i.e., $\sigma I + K_l(X, X)$, to account for noise in the Gaussian process model. Numerically, this diagonal shift is also necessary to keep the linear system from being extremely ill-conditioned when l is small. We set $\sigma = 10^{-2}$ in the following tests.

Table 6.1 lists the number of PCG iterations for solves involving the three types of kernel matrices with various parameters l, generated by a ball point set of size 3.2×10^5. For all the different types of kernels, FSAI performs very well when l is large, corresponding to kernel matrices that are close to sparse. For smaller values of l, the performance of FSAI deteriorates.

In comparison, SPDHSS has more consistent preconditioning performance for this wide range of parameters, although it takes more iterations than FSAI for large l in most cases. This consistency is an advantage of SPDHSS over FSAI, since the parameter l changes during optimization, and it could be difficult to quantitatively decide when to use FSAI, particularly for more complicated kernel functions.

We also observe that SPDHSS has better performance than BJ in all the tests. The SPDHSS preconditioner can be viewed as the combination of a BJ preconditioner (the diagonal blocks) with some off-diagonal approximations. Finally, the computed regular HSS preconditioner in most cases is not SPD.

TABLE 6.1

Number of PCG iterations for systems with diagonal-shifted kernel matrices $\sigma I + K_l(X, X)$ with different kernels and parameters l. All tests use the same $N = 3.2 \times 10^5$ points in a ball. The notation "−" means that PCG fails to converge within 3000 iterations; "/" means that a regular HSS preconditioner is not SPD.

Parameter l	0.0010	0.0025	0.005	0.010	0.025	0.05	0.10	0.25	0.5	1.0
Unpreconditioned	41	119	297	687	1896	-	-	1684	634	210
BJ	459	1202	2504	-	-	-	2765	707	172	82
FSAI $k = 200$	2659	2623	2045	1518	960	569	266	63	18	6
FSAI $k = 400$	1734	1531	1111	831	511	266	108	28	10	4
SPDHSS $r = 100$	2	3	5	14	62	159	287	245	117	56
SPDHSS $r = 200$	1	2	4	5	20	58	131	171	92	50
HSS $r = 100$	2	3	6	/	/	/	/	/	/	/
HSS $r = 200$	2	2	3	6	/	/	/	/	/	/

(a) Matérn kernel

Parameter l	0.0001	0.0005	0.001	0.005	0.01	0.05	0.1	0.5	1.0
Unpreconditioned	103	394	838	2498	2396	1098	695	232	150
BJ	1851	-	-	-	-	989	487	147	85
FSAI $k = 200$	-	-	-	2718	1371	253	106	16	7
FSAI $k = 400$	-	-	-	1498	687	93	43	9	4
SPDHSS $r = 100$	1	3	21	577	855	563	378	118	65
SPDHSS $r = 200$	1	2	6	116	284	415	314	106	59
HSS $r = 100$	2	3	/	/	/	/	/	/	/
HSS $r = 200$	2	2	2	/	/	/	/	/	/

(b) Gaussian kernel

Parameter l	0.001	0.005	0.01	0.05	0.1	0.5	1	5	10	50	100
Unpreconditioned	1239	2656	-	-	2812	1958	1576	915	724	394	284
BJ	-	-	-	2300	1605	529	322	195	151	97	73
FSAI $k = 200$	2839	1092	619	201	121	45	32	24	24	23	23
FSAI $k = 400$	1598	535	266	73	51	24	21	18	18	18	17
SPDHSS $r = 100$	53	249	328	355	291	112	74	38	28	13	10
SPDHSS $r = 200$	12	71	129	212	195	84	60	33	23	10	8
HSS $r = 100$	/	/	/	/	/	/	/	/	/	/	/
HSS $r = 200$	/	/	/	/	/	/	/	/	/	/	/

(c) IMQ kernel

Table 6.2 shows the time required to construct and apply (solve with) the various preconditioners. The table also shows the time required to construct and apply (multiply by) the $\mathcal{H}^2$ representation. The storage requirements for the $\mathcal{H}^2$ representation and for the preconditioners are also shown. The construction cost of an SPDHSS approximation depends on the efficiency of the corresponding $\mathcal{H}^2$ representation and thus varies for different kernel functions. The application of the SPDHSS preconditioners, although more expensive than for FSAI preconditioners, is relatively fast in comparison to corresponding $\mathcal{H}^2$ matrix-vector multiplications.

6.2.2. Kernel matrices with varying sizes. We now consider the iterative solution of the Matérn and RPY kernel matrix systems for systems of different sizes. For the RPY kernel, particle radii $a = 0.29$ and $a = 0.42$ are selected such that each ball point set has corresponding volume fraction of particles around 0.1 and 0.3, respectively. These two volume fractions are representative for macromolecular

TABLE 6.2

Timings (in sec.) for constructing and applying (solves) the preconditioner and storage (in GB) for the BJ, FSAI, and SPDHSS preconditioners. The table also includes timings for constructing and applying (matrix-vector multiplication) the $\mathcal{H}^2$ representation. The matrices are from those in Table 6.1 defined by the Matérn with $l = 0.025$, Gaussian with $l = 0.01$, and IMQ with $l = 1$. The $\mathcal{H}^2$ representations of the three test matrices require 0.7, 1.8, and 1.2 GB of storage, respectively.

		Matérn		Gaussian		IMQ	
	Storage	Constr.	Apply	Constr.	Apply	Constr.	Apply
$\mathcal{H}^2$ representation	0.7/1.8/1.2	1.7	0.30	10.8	0.36	4.6	0.92
BJ	0.3	0.3	0.016	2.9	0.0055	0.1	0.0091
FSAI $k = 200$	0.7	8.6	0.0070	6.7	0.0052	7.8	0.0083
FSAI $k = 400$	1.4	13.1	0.013	12.3	0.0081	15.3	0.023
SPDHSS $r = 100$	2.2	21.1	0.046	22.5	0.041	23.6	0.043
SPDHSS $r = 200$	4.6	38.3	0.15	44.0	0.11	43.9	0.11

TABLE 6.3

Number of PCG iterations for kernel matrices defined by different point sets.

N ($\times 10^4$)	Ball point sets					Sphere point sets				
	4	8	16	32	64	4	8	16	32	64
Matérn $l = 0.01$										
Unpreconditioned	134	217	397	689	1235	370	713	1252	2195	-
BJ	1209	2249	-	-	-	1472	2642	-	-	-
FSAI $k = 200$	762	1215	1205	1503	-	504	651	877	1340	1791
FSAI $k = 400$	347	693	656	849	1474	247	272	380	470	634
SPDHSS $r = 100$	3	4	7	14	29	4	4	7	17	34
SPDHSS $r = 200$	2	3	3	5	10	3	3	3	4	8
Matérn $l = 0.25$										
Unpreconditioned	1123	1358	1542	1681	1790	549	559	566	569	570
BJ	574	568	763	704	650	169	197	184	199	197
FSAI $k = 200$	56	83	53	63	114	16	23	21	24	28
FSAI $k = 400$	26	36	22	28	46	7	9	8	10	12
SPDHSS $r = 100$	100	148	196	236	294	38	55	71	85	101
SPDHSS $r = 200$	45	76	122	172	216	11	21	36	51	71
RPY $a = 0.29$										
Unpreconditioned	432	510	1055	1151	1653	549	707	1741	1448	1706
BJ	95	142	181	238	282	51	104	100	139	147
FSAI $k = 200$	89	113	137	174	222	31	35	42	50	60
FSAI $k = 400$	84	98	122	162	212	24	28	34	37	48
SPDHSS $r = 100$	40	51	62	77	99	16	22	27	28	31
SPDHSS $r = 200$	27	36	46	58	75	12	14	22	22	25
RPY $a = 0.42$										
Unpreconditioned	632	762	1571	1723	2428	809	1063	2566	2132	2515
BJ	150	218	237	328	436	66	140	128	178	191
FSAI $k = 200$	137	165	206	266	338	38	43	52	62	77
FSAI $k = 400$	125	151	176	244	307	28	32	37	43	56
SPDHSS $r = 100$	64	79	101	121	157	23	30	35	39	45
SPDHSS $r = 200$	45	59	74	95	122	16	20	30	29	36

simulations of conditions within biological cells [7]. For the Matérn kernel, $l = 0.25$ and $l = 0.01$ are tested based on the previous results in Table 6.1, where FSAI performs better than SPDHSS for $l = 0.25$ and vice versa for $l = 0.01$. No diagonal shift is added to RPY kernel matrices, while a shift of $\sigma = 10^{-2}$ is added to Matérn kernel matrices as before.

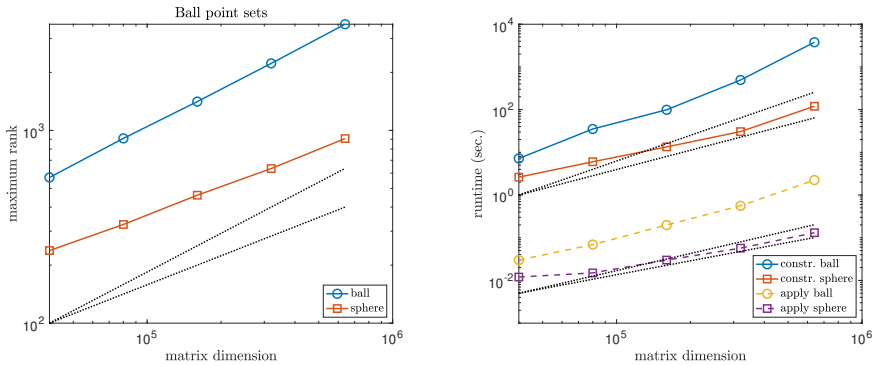
Table 6.3 shows PCG convergence for systems using the two kernel functions with different point sets. As expected, iteration numbers increase with matrix sizes

for the FSAI and SPDHSS preconditioners since a fixed approximation rank r and sparsity parameter k are used. Related to this is the increasing relative approximation error in the SPDHSS approximation with increasing matrix size when r is fixed, as observed earlier in Figure 6.3. As to be shown next, it is possible to obtain scalable preconditioning performance by constructing SPDHSS preconditioners with a fixed relative error threshold but at the sacrifice of asymptotically more expensive cost in SPDHSS construction and solve.

6.2.3. SPDHSS with a fixed relative error threshold. To demonstrate the preconditioning performance and computational complexity of SPDHSS with a fixed relative error threshold, we consider the Matérn kernel with $l = 0.25$ which was previously tested with fixed ranks (Table 6.3). Applying SPDHSS with two relative error thresholds $\tau = 10^{-1}$ and $\tau = 10^{-2}$, Table 6.4 shows PCG iteration counts and average matrix-vector multiplication errors of SPDHSS approximations for systems with different point sets. The iteration counts are roughly constant with different-sized problems, suggesting scalable preconditioning performance. The results in Table 6.4 show that the error of an SPDHSS approximation is well controlled by the relative error threshold for our test problems. However, we note that an error threshold is applied to the compression of scaled blocks in SPDHSS construction, which only indirectly controls the overall matrix approximation error.

TABLE 6.4
Number of PCG iterations and average matrix-vector multiplication errors for kernel matrices defined by the Matérn kernel with $l = 0.25$ and different point sets.

N ($\times 10^4$)	Ball point sets					Sphere point sets				
	4	8	16	32	64	4	8	16	32	64
Iteration counts										
SPDHSS $\tau = 10^{-1}$	85	79	90	88	89	43	43	46	50	49
SPDHSS $\tau = 10^{-2}$	10	10	10	13	10	7	9	12	10	10
Matvec errors										
SPDHSS $\tau = 10^{-1}$	0.05	0.05	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
SPDHSS $\tau = 10^{-2}$	0.004	0.003	0.003	0.003	0.003	0.005	0.004	0.005	0.005	0.005



(a) Maximum ranks (b) Timings for construction and application

FIG. 6.4. *Maximum approximation ranks and timings for construction and application (solves) of SPDHSS with fixed relative error threshold $\tau = 10^{-2}$ for ball and sphere point sets. In* (a)*, the two dotted reference lines show $O(N^{1/2})$ and $O(N^{2/3})$ scaling. In* (b)*, the four dotted reference lines show $O(N \log(N))$, $O(N^{4/3})$, $O(N^{3/2})$, and $O(N^2)$ scaling.*

Figure 6.4 plots the maximum approximation ranks and SPDHSS construction and application costs with different point sets for $\tau = 10^{-2}$. For these ball (sphere) point sets, it is well known (e.g., see [20]) that an HSS approximation with a fixed relative error, whether it is regular HSS or SPDHSS, can have at least $O(N^{2/3})$ ($O(N^{1/2})$) maximum approximation ranks and have at least $O(N^2)$ ($O(N^{3/2})$) factorization and $O(N^{4/3})$ ($O(N \log N)$) solve cost. Our numerical results with SPDHSS are consistent with this theoretical analysis.

Overall, although the number of PCG iterations remains roughly constant when the problem size increases, the costs of precomputation of SPDHSS construction and the application of SPDHSS as a preconditioner both increase superlinearly. It is thus more practical to use a proper combination of a maximum rank threshold and a relative error threshold for the application of SPDHSS.

7. Conclusion. Fast direct solvers and rank-structured preconditioners, such as those using the HSS representation, impose a block structure on a matrix that provides for fast solve operations, but the rigid block structure (arising from so-called weak admissibility) also results in large block ranks, especially if an accurate representation is desired. This leads to high construction cost.

On the other hand, more general rank-structured matrix representations, such as $\mathcal{H}^2$, have a flexible block structure (arising from so-called strong admissibility) that allows for an accurate representation with smaller block ranks, and thus these representations have relatively low construction cost. However, the general structure does not admit fast solve operations.

This paper, in a way, combines these two types of rank-structured matrix representations. The paper shows how to accelerate the construction of an SPD HSS approximation to an SPD matrix by exploiting and only using an $\mathcal{H}^2$ representation of the SPD matrix that is assumed to be available, for example, in the context of a preconditioned iterative solve. The acceleration results from (i) using fast $\mathcal{H}^2$ matrix-vector multiplication to compute scaled basis matrices, V_i and $\bar{V}_i$, needed in constructing the HSS representation and from (ii) using existing low-rank blocks in the $\mathcal{H}^2$ representation to reduce the number of coefficient matrices B_{ij} that need to be computed in the HSS representation.

While we only tested SPDHSS as a preconditioner on kernel matrices, its application to linear systems from the numerical solution of integral equations is straightforward. Further, although we only considered dense SPD matrices, our proposed algorithms, Algorithm 4.1 and Algorithm 5.3, can also be applied directly to sparse SPD matrices that are ubiquitous in the numerical solution of partial differential equations. Both methods can still guarantee the positive definiteness of the constructed preconditioners in the sparse case, but it is worthy to study whether it is possible to exploit matrix sparsity directly to accelerate SPD HSS construction. It is also worthy to study whether or not the FSAI and SPDHSS preconditioners can be beneficially combined, i.e., augmenting a sparse preconditioner with a dense one.

REFERENCES

[1] S. Ambikasaran and E. Darve, *An $\mathcal{O}(n \log n)$ fast direct solver for partial hierarchically semi-separable matrices*, J. Sci. Comput., 57 (2013), pp. 477–501.

[2] S. Ambikasaran and E. Darve, *The Inverse Fast Multipole Method*, arXiv preprint, arXiv:1407.1572 [math. NA], 2014.

[3] S. Ambikasaran, M. O'Neil, and K. R. Singh, *Fast Symmetric Factorization of Hierarchical Matrices with Applications*, arXiv preprint, arXiv:1405.0223 [math. NA], 2014.

[4] S. Börm, L. Grasedyck, and W. Hackbusch, *Hierarchical Matrices*, Lecture, Note 21/2003, max-planch-Institut für mathematik in den Naturwissen schaften, 2003.
[5] L. Cambier, C. Chen, E. G. Boman, S. Rajamanickam, R. S. Tuminaro, and E. Darve, *An algebraic sparsified nested dissection algorithm using low-rank approximations*, SIAM J. Matrix Anal. Appl., 41 (2020), pp. 715–746.
[6] S. Chandrasekaran, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.
[7] E. Chow and J. Skolnick, *Effects of confinement on models of intracellular macromolecular dynamics*, Proc. Natl. Acad. Sci. USA, 112 (2015), pp. 14846–14851.
[8] P. Coulier, H. Pouransari, and E. Darve, *The inverse fast multipole method: Using a fast approximate direct solver as a preconditioner for dense linear systems*, SIAM J. Sci. Comput., 39 (2017), pp. A761–A796.
[9] J. Feliu-Fabà, K. L. Ho, and L. Ying, *Recursively preconditioned hierarchical interpolative factorization for elliptic partial differential equations*, Commun. Math. Sci., 18 (2020), pp. 91–108.
[10] B. Fornberg and N. Flyer, *Solving PDEs with radial basis functions*, Acta Numer., 24 (2015), p. 215.
[11] P. Ghysels, X. S. Li, F.-H. Rouet, S. Williams, and A. Napov, *An efficient multicore implementation of a novel HSS-structured multifrontal solver using randomized sampling*, SIAM J. Sci. Comput., 38 (2016), pp. S358–S384.
[12] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.
[13] L. Greengard and V. Rokhlin, *A new version of the fast multipole method for the Laplace equation in three dimensions*, Acta Numer., 6 (1997), pp. 229–269.
[14] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part* I: *Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.
[15] W. Hackbusch and S. Börm, *Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices*, Computing, 69 (2002), pp. 1–35.
[16] W. Hackbusch, B. Khoromskij, and S. A. Sauter, *On $\mathcal{H}^2$-matrices*, in Lectures on Applied Mathematics, Springer-Verlag, Berlin, 2000, pp. 9–29.
[17] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic. Part* II: *Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47.
[18] N. Halko, P. Martinsson, and J. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.
[19] M. J. Heaton, A. Datta, A. O. Finley, R. Furrer, J. Guinness, R. Guhaniyogi, F. Gerber, R. B. Gramacy, D. Hammerling, M. Katzfuss, F. Lindgren, D. W. Nychka, F. Sun, and Zammit-Mangion, *A case study competition among methods for analyzing large spatial data*, J. Agric., Biol. Environ. Stat., 24 (2019), pp. 398–425.
[20] K. L. Ho and L. Greengard, *A fast direct solver for structured linear systems by recursive skeletonization*, SIAM J. Sci. Comput., 34 (2012), pp. A2507–A2532.
[21] K. L. Ho and L. Ying, *Hierarchical interpolative factorization for elliptic operators: Integral equations*, Comm. Pure Appl. Math, 69 (2016), pp. 1314–1353.
[22] H. Huang, X. Xing, and E. Chow, *H2Pack: High-performance $\mathcal{H}^2$ matrix package for kernel matrices using the proxy point method*, ACM Trans. Math. Software, 47 (2020), 3.
[23] L. Y. Kolotilina and A. Y. Yeremin, *Factorized sparse approximate inverse preconditionings* i. *Theory*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58.
[24] Y. Li, H. Yang, E. R. Martin, K. L. Ho, and L. Ying, *Butterfly factorization*, Multiscale Model. Simul., 13 (2015), pp. 714–732.
[25] Y. Liu, X. Xing, H. Guo, E. Michielssen, P. Ghysels, and X. S. Li, *Butterfly Factorization via Randomized Matrix-Vector Multiplications*, arXiv preprint, arXiv:2002.03400 [math. NA], 2020.
[26] P.-G. Martinsson, *A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1251–1274.
[27] J. Rotne and S. Prager, *Variational treatment of hydrodynamic interaction in polymers*, J. Chem. Phys., 50 (1969), pp. 4831–4837.
[28] F.-H. Rouet, X. S. Li, P. Ghysels, and A. Napov, *A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization*, ACM Trans. Math. Software, 42 (2016), pp. 1–35.
[29] J. Xia, *Robust and Effective eSIF Preconditioning for General SPD Matrices*, arXiv preprint, arXiv:2007.03729 [math. NA], 2020.
[30] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Nume. Linear Algebra Appl., 17 (2010), pp. 953–976.

[31] J. Xia and Z. Xin, *Effective and robust preconditioning of general SPD matrices via structured incomplete factorization*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 1298–1322.
[32] Z. Xin, J. Xia, S. Cauley, and V. Balakrishnan, *Effectiveness and robustness revisited for a preconditioning technique based on structured incomplete factorization*, Nume. Linear Algebra Appl., 27 (2020), e2294.
[33] X. Xing and E. Chow, *An Efficient Method for Block Low-Rank Approximations for Kernel Matrix Systems*, arXiv preprint, arXiv:1811.04134 [math. NA], 2018.
[34] X. Xing and E. Chow, *Preserving positive definiteness in hierarchically semiseparable matrix approximations*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 829–855.
[35] X. Xing and E. Chow, *Interpolative decomposition via proxy points for kernel matrices*, SIAM J. Matrix Anal. Appl., 41 (2020), pp. 221–243.
[36] H. Yamakawa, *Transport properties of polymer chains in dilute solution: Hydrodynamic interaction*, J. Chem. Phys., 53 (1970), pp. 436–443.