THE PROXY POINT METHOD FOR RANK-STRUCTURED MATRICES

A Dissertation
Presented to
The Academic Faculty

By

Xin Xing

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Mathematics

Georgia Institute of Technology

December 2019

THE PROXY POINT METHOD FOR RANK-STRUCTURED MATRICES

To my parents and my wife.

ACKNOWLEDGEMENTS

I would like to first express my sincere gratitude to my advisor Edmond Chow for his guidance and support in my PhD study and research. His professionalism, wisdom, and patience have guided me to get through this long and occasionally tough PhD journey and have also motivated and inspired me to do better in my life and career. I feel truly blessed to work with him.

I would like to thank all other members of my committee: David Sherrill, Jianlin Xia, Yuanzhe Xi, Haomin Zhou. Haomin Zhou is my mentor in the math department and has offered me a lot of encouragement, support, and guidance since I started my PhD study. He is the one who I can always resort to. Jianlin Xia has taught me a lot about HSS matrices and has also provided me many opportunities to present my work in conferences. Yuanzhe Xi has shared with me his valuable experience in pursuing an academic career and many interesting academia stories. Discussing with him is always rewarding.

I would like to thank my collaborators Hua Huang and Yang Liu. With his passion and excellent HPC skills, Hua Huang helped me turn my toy $\mathcal{H}^2$ code into a true library for real application which is one of the most exciting PhD works I have got. Yang Liu introduced me to the butterfly method and the butterfly factorization and we had many enlightening discussions about rank-structured matrices.

This work was partially supported by NSF under grants ACI-1306573 and ACI-1609842.

I would also like to thank all my friends for all these good days we spent together in Georgia Tech, especially to Fan, Haoyan, Qianli, Jiangning, Xiaowen, Hua, Jordi, Lucas, Ruilin, Tongzhou, Xiao, Yian. Furthermore, I would like to thank all my volleyball and badminton partners for all these wonderful games we played, especially to Yanxi, Peizheng, Peijue, Xinran, Fan, Ruilin, Xiao, Yian, Brian, Congshi.

Lastly, I would like to express my deepest gratitude to my parents and my wife for their love, accompany, and support. I wouldn't be here without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Rank-structured matrix representations, e.g., $\mathcal{H}^2$ and HSS, are commonly used to reduce computation and storage cost for dense matrices defined by interactions between many bodies. The main bottleneck for their application is the expensive computation required to represent a matrix in a rank-structured matrix format which involves compressing specific matrix blocks into low-rank form. This dissertation is mainly about the study and application of a hybrid analytic-algebraic compression method, called *the proxy point method*. This work uncovers the full strength of this presently underutilized method that could potentially resolve the above bottleneck for all rank-structured matrix techniques. As a result, this work could extend the applicability and improve the performance of rank-structured matrix techniques and thus facilitate dense matrix computations in a wider range of scientific computing problems, such as particle simulations, numerical solution of integral equations, and Gaussian processes.

Application of the proxy point method in practice is presently very limited. Only two special instances of the method have been used heuristically to compress interaction blocks defined by specific kernel functions over points. We address several critical problems of the proxy point method which limit its applicability. A general form of the method is then proposed, paving the way for its wider application in the construction of different rank-structured matrix representations with kernel functions that are more general than those usually used.

In addition to kernel-defined interactions between points, we further extend the applicability of the proxy point method to compress the interactions between charge distributions in quantum chemistry calculations. Specifically, we propose a variant of the proxy point method to efficiently construct an $\mathcal{H}^2$ matrix representation of the four-dimensional electron repulsion integral tensor. The linear-scaling matrix-vector multiplication algorithm for the constructed $\mathcal{H}^2$ matrix is then used for fast Coulomb matrix construction which is an important step in many quantum chemical methods.

Two additional contributions related to $\mathcal{H}^2$ and HSS matrices are also presented.

First, we explain the exact equivalence between $\mathcal{H}^2$ matrices and the fast multipole method (FMM). This equivalence has not been rigorously studied in the literature. Numerical comparisons between FMM and $\mathcal{H}^2$ matrices based on the proxy point method are also provided, showing the relative advantages and disadvantages of the two methods.

Second, we consider the application of HSS approximations as preconditioners for symmetric positive definite (SPD) matrices. Preserving positive definiteness is essential for rank-structured matrix approximations to be used efficiently in various algorithms and applications, e.g., the preconditioned conjugate gradient method. We propose two methods for constructing HSS approximations to an SPD matrix that preserve positive definiteness.

CHAPTER 1
INTRODUCTION

Large dense matrices appear in many scientific computing problems, such as particle simulations, numerical solution of integral equations, and Gaussian processes. Usually, these dense matrices are defined by non-compact interactions between many bodies (e.g., points and distributions) and have specific *block low-rank structures*, meaning that certain blocks of the matrices are numerically low-rank. For such a matrix, representing these blocks in low-rank form can reduce the quadratic cost for matrix storage and matrix-vector multiplications and is referred to as representing the matrix in a *rank-structured matrix* format. Depending on different block low-rank structures, there are different rank-structured matrix formats, such as $\mathcal{H}$ [1, 2], $\mathcal{H}^2$ [3, 4], HSS [5], HODLR [6], directional $\mathcal{H}^2$ [7, 8], and butterfly factorization [9]. The main bottleneck of applying all these rank-structured matrix techniques is the expensive computation required to compress all the matrix blocks characterized by block low-rank structure into low-rank form. For example, simply evaluating all the matrix entries takes quadratic computation cost, making algebraic compression methods such as SVD or QR decompositions unfavorable. As a result, efficient compression methods are critical for the construction and application of rank-structured matrix techniques.

In this dissertation, we focus on a hybrid analytic-algebraic compression method, called the proxy point method. This work uncovers the full strength of this presently underutilized method that could potentially resolve the above bottleneck for all the rank-structured matrix techniques. As a result, this work could further extend the applicability and improve the performance of rank-structured matrix techniques and thus facilitate dense matrix computations in a wider range of scientific computing problems.

For kernel functions from potential theory, such as the Laplace kernel and Stokes kernel, Martinsson and Rokhlin [10] introduced *the proxy surface method* to efficiently compress specific kernel matrix blocks into a low-rank form called interpolative decomposition [11] (ID, to be explained in Section 1.2). This technique has been applied in a class of fast direct solvers [10, 12, 13, 14] for kernel matrices based on HSS format and dramatically reduces the HSS construction cost. Methods closely related to the proxy surface method also exist which differ in their selection of so-called proxy points [15, 16]. Together, all these methods, including the proxy surface method, have a general form that we refer to as *the proxy point method*. While the proxy point method has its advantages compared to purely algebraic or analytic compression methods, several important problems concerning the method still remain unsolved, limiting its present application only to specific kernel functions.

This dissertation focuses on the rigorous study and extensive application of the proxy point method. We first address the unsolved problems concerning the method, paving the way for its wider application in the construction of different types of rank-structured matrices with kernel functions that are more general than those usually used. In addition to kernel matrices (which are associated with kernel-defined interactions between points), we further extend the applicability of the proxy point method to compress specific interaction blocks defined by charge distributions in quantum chemistry calculations. Specifically, we propose a variant of the proxy point method to efficiently construct an $\mathcal{H}^2$ matrix representation of the four-dimensional electron repulsion integral (ERI) tensor. The linear-scaling matrix-vector multiplication algorithm for the constructed $\mathcal{H}^2$ matrix is then used for Coulomb matrix construction which is an important step in many quantum chemical methods.

Two additional contributions related to $\mathcal{H}^2$ and HSS matrices are also presented. (These are not closely connected to the proxy point method and can be read independently.) In the first contribution, we provide a detailed explanation of the exact equivalence between $\mathcal{H}^2$ matrices and the fast multipole method (FMM) [17, 18]. This equivalence has not been rigorously studied in the literature. We also compare the numerical performance of two commonly used FMM libraries and our own $\mathcal{H}^2$ matrix library based on the proxy point method. In the second contribution, we consider the application of HSS approximations as preconditioners for symmetric positive definite (SPD) matrices. We propose two methods for constructing HSS approximations to an SPD matrix that preserve positive definiteness which is essential for the HSS approximations to be used as preconditioners in various algorithms.

1.1 Outline and contributions

In Chapter 2, we provide a thorough review of the $\mathcal{H}^2$ matrix representation from a new viewpoint. The structure of $\mathcal{H}^2$ matrices was originally proposed and explained as a combination of the structure of $\mathcal{H}$ matrices, a simpler rank-structured matrix format, with specific restrictions on the low-rank approximation of various matrix blocks in $\mathcal{H}$ matrices. In comparison, the new viewpoint gives a more direct explanation of $\mathcal{H}^2$ matrices. From this viewpoint, we also propose modifications to improve the original structure of $\mathcal{H}^2$ matrices. This chapter serves as a background chapter and provides the basic concepts and notation for $\mathcal{H}^2$ matrices that are used in the thesis.

In Chapter 3, we present and study a general form of the proxy point method for the low-rank approximation of kernel matrices defined by kernel functions over points. Specifically, we address several critical unsolved problems about the proxy point method: how the method works, under what conditions the method works, and how to select a proper set of proxy points. We note that the set of proxy points here is a key component in the proxy point method and is critical to the effectiveness of the method. However, it is only selected heuristically in practice. These problems limit the present application of the proxy point method only to the construction of $\mathcal{H}^2$ matrices with specific kernel functions. In this chapter, we provide a rigorous error analysis for the proxy point method under a general problem setting. Moreover, we propose a systematic and adaptive scheme to select a proper set of proxy points under different problem settings.

In Chapter 4, we extend the applicability of both the proxy point method and $\mathcal{H}^2$ matrix representation to the four-dimensional electron repulsion integral (ERI) tensors in quantum chemistry for fast Coulomb matrix construction. Constructing the Coulomb matrix is equivalent to computing a matrix-vector multiplication where the matrix (referred to as the ERI matrix) has entries defined as the Coulomb interactions between continuous charge distributions. We propose to construct an explicit $\mathcal{H}^2$ matrix representation of the ERI matrix to accelerate the Coulomb matrix construction. The main challenge of this approach is still the expensive computation required in the $\mathcal{H}^2$ matrix construction to compress specific blocks of the ERI matrix into low-rank form. A variant of the proxy point method is proposed to tackle this challenge and also to avoid computing all the ERI entries. This variant helps reduce the $\mathcal{H}^2$ matrix construction cost to nearly linear in the ERI matrix dimension. As a result, this new $\mathcal{H}^2$-based approach to constructing the Coulomb matrix is fast and has linear computation and storage cost.

In Chapter 5, we provide a detailed explanation of the exact equivalence between $\mathcal{H}^2$ matrices and FMM. It is known that FMM for the fast matrix-vector multiplication of a kernel matrix is algebraically equivalent to multiplying by the matrix in $\mathcal{H}^2$ format where all the low-rank approximations are analytically constructed by multipole expansions. However, this equivalence has not been rigorously studied in the literature. Moreover, we experimentally compare the numerical performance of two state-of-the-arts FMM libraries and our own $\mathcal{H}^2$ matrix library currently under development with collaborators based on the proxy point method. These numerical experiments demonstrate the relative advantages and disadvantages between FMM and $\mathcal{H}^2$ matrices. It is worth noting that the proxy point method plays a critical role in reducing $\mathcal{H}^2$ matrix construction cost in our $\mathcal{H}^2$ matrix library, leading to competitive performance compared to these FMM libraries.

In Chapter 6, we consider the application of approximate HSS representations as preconditioners for SPD matrices. Given an SPD matrix, it is desirable to compute an HSS approximation that is also positive definite, which is essential for the approximation to be used as a preconditioner in various algorithms. However, positive definiteness is not guaranteed as HSS approximations generally only focus on compressing matrix blocks into low-rank form. We first provide a new recursive description of the construction process of HSS approximations. Based on this new description, we propose two methods for constructing HSS approximations to an SPD matrix that preserve positive definiteness. Compared to existing SPD rank-structured preconditioners, our proposed methods have similar computation cost but are much more parallel and also are more stable in terms of preserving positive definiteness.

1.2 Mathematical preliminaries

The two cornerstones of rank-structured matrix representations, including $\mathcal{H}^2$ and HSS we mainly studied, are the block low-rank structure of matrices from applications and the low-rank approximation of matrices. This section introduces the basic notation, facts, and methods related to these two aspects.

1.2.1 Basic concepts

Given a matrix $A \in \mathbb{R}^{m\times n}$, a rank-$r$ approximation of A with accuracy $O(\varepsilon)$ can be written in the form

$$A = \underbrace{U}_{m\times r}\,\underbrace{V^T}_{r\times n} + O(\varepsilon).$$

If $r \ll \min(m, n)$, A is said to be *numerically low-rank* with accuracy $O(\varepsilon)$ and UV^T is a *low-rank approximation* of A. In this case, $(m+n)r \ll mn$ and representing A by UV^T dramatically reduces the cost for matrix storage and matrix-vector multiplications. Given a rank r, the optimal rank-r approximation of A in the 2-norm or in the Frobenius norm is characterized by the *singular value decomposition* (SVD) of A. There exists an accurate low-rank approximation of A if and only if A has fast-decaying singular values.

1.2.2 Low-rank property of kernel matrices

Most dense matrices from applications that have block low-rank structures are associated with interactions between many bodies. Of all these dense matrices, the most common and the most fundamental ones are dense *kernel matrices* in the form $K(X,Y)$ where $K(x,y)$ is a non-compact bivariate function (called a kernel function), X and Y are two sets of points, and $K(X,Y)$ has entries $K(x_i,y_j)$ with all pairs of x_i in X and y_j in Y. In this subsection, we characterize the block low-rank structure of kernel matrices defined by a kernel function $K(x,y)$.

Degenerate function approximation Let $K(x,y)$ be a kernel function and $\mathcal{X}$ and $\mathcal{Y}$ be two compact domains. For two sets of points, $X_0 \in \mathcal{X}$ and $Y_0 \in \mathcal{Y}$, the low-rank property of $K(X_0,Y_0)$ is closely related to the existence of an accurate low-degree *degenerate approximation* of $K(x,y)$ in $\mathcal{X} \times \mathcal{Y}$. A degenerate approximation of $K(x,y)$ in $\mathcal{X} \times \mathcal{Y}$ is formally defined as follows.

Definition 1. *$K(x,y)$ is said to have an r-term ε-expansion in $\mathcal{X} \times \mathcal{Y}$ if there exist functions $\{\psi_i(x)\}_{i=1}^r$ and $\{\phi_i(y)\}_{i=1}^r$ such that*

$$\left| K(x,y) - \sum_{i=1}^{r} \psi_i(x)\phi_i(y) \right| \leqslant \varepsilon, \quad x \in \mathcal{X},\ y \in \mathcal{Y}. \tag{1.1}$$

The summation in eq. (1.1) *is called a degenerate approximation (a.k.a. separated representation) of $K(x,y)$ with degree r and accuracy ε in $\mathcal{X} \times \mathcal{Y}$.*

Let $\Psi(x)$ and $\Phi(y)$ denote the r-dimensional vectors consisting of functions $\{\psi_i(x)\}_{i=1}^r$ and $\{\phi_i(y)\}_{i=1}^r$ from eq. (1.1), respectively. The approximation eq. (1.1) can be written as

$$K(x,y) = \Psi(x)^T \Phi(y) + O(\varepsilon). \tag{1.2}$$

Substituting all pairs of $x_i \in X_0$ and $y_j \in Y_0$ into the above equation gives $K(X_0,Y_0) \approx$

$\Psi(X_0)^T\Phi(Y_0)$ where $\Psi(X_0) \in \mathbb{R}^{r\times|X_0|}$ denotes the matrix of column vectors $\Psi(x_i)$ for all $x_i \in X_0$ and $\Phi(Y_0)$ is similarly defined. As can be noted, $\Psi(X_0)^T\Phi(Y_0)$ is a rank-r approximation to $K(X_0, Y_0)$ with $O(\varepsilon)$ error.

This observation shows that if there exists an accurate low-degree degenerate approximation of $K(x, y)$ in $\mathcal{X} \times \mathcal{Y}$, i.e., r is $O(1)$ and ε is sufficiently small, $K(X_0, Y_0)$ with any $X_0 \in \mathcal{X}$ and $Y_0 \in \mathcal{Y}$ is numerically low-rank.

Admissibility conditions To locate the numerically low-rank blocks of a kernel matrix, algebraically checking the numerical rank of a block $K(X_0, Y_0)$ (e.g., by SVD) is expensive and impractical. Instead, it is more common and efficient to check the pair of domains $\mathcal{X}$ and $\mathcal{Y}$ to decide whether there exists an accurate low-degree degenerate approximation of $K(x, y)$ in $\mathcal{X} \times \mathcal{Y}$ and thus to decide whether $K(X_0, Y_0)$ is numerically low-rank.

For kernel functions commonly used in practice, there exist geometric criteria for a pair of domains $\mathcal{X}$ and $\mathcal{Y}$ that can decide whether such a degenerate approximation of $K(x, y)$ in $\mathcal{X}\times\mathcal{Y}$ exists. Such geometric criteria are referred to as *admissibility conditions* and a pair of domains that meet the criteria are said to be *admissible*. Depending on kernel functions, different rank-structured matrix formats exploit different admissibility conditions (and thus different block low-rank structures) to locate the numerically low-rank blocks inside a kernel matrix. Figure 1.1 shows examples of admissible pairs of domains in three different situations.

1.2.3 Low-rank approximation methods

Algebraic methods A low-rank approximation of $A \in \mathbb{R}^{m\times n}$ can usually be constructed by truncating a specific matrix decomposition of A, such as SVD, pivoted QR decomposition, and pivoted LU decomposition. These decomposition-based methods usually must access all the matrix entries and thus have computation cost at least $O(mn)$.

The main low-rank form we will use in this dissertation is the *interpolative decomposi-*

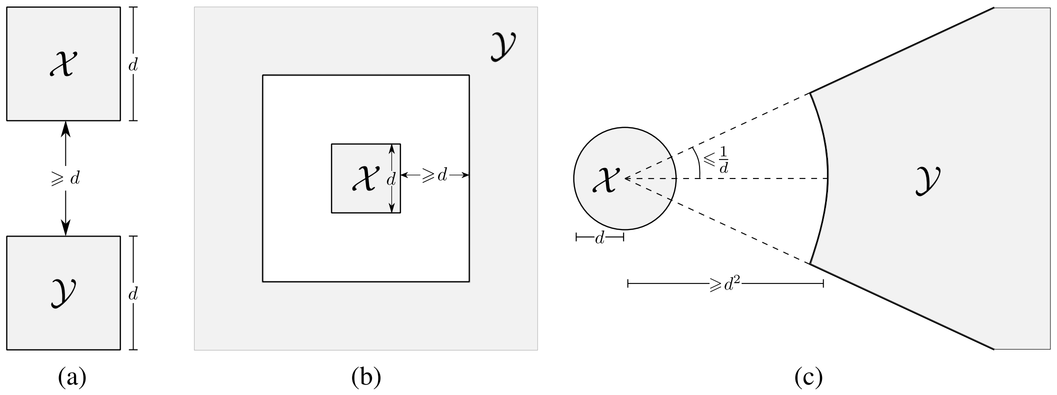


Figure 1.1: Examples of admissible pairs of domains $\mathcal{X} \times \mathcal{Y}$ in three different situations: (a) $K(x, y) = \log(|x - y|)$ in $\mathcal{H}$ format [2], (b) $K(x, y) = \log(|x - y|)$ in $\mathcal{H}^2$ format [4], (c) 2D Helmholtz kernel function in the fast directional multilevel algorithm [19]. Admissibility conditions generally have constraints on the relative size of certain geometric features of $\mathcal{X}$ and $\mathcal{Y}$ which is characterized by the parameter d in the figures.

tion (ID) [12]. A rank-r ID approximates or represents A as

$$A \approx \underbrace{U}_{m\times r}\underbrace{A_J}_{r\times n}, \tag{1.3}$$

where U has bounded entries and A_J contains r rows of A. An ID approximation defined in this way is said to have error below the error threshold ε_0 if the 2-norm of each row of $A - UA_J$ is bounded by ε_0.

Using an algebraic approach, an ID approximation with a given rank or a given error threshold can be calculated using the *strong rank-revealing QR* (SRRQR) decomposition [11] with typical computation cost $O(mnr)$ (or $O(mnr\log r)$ in rare cases). The matrix U computed by SRRQR can have all its entries bounded by a prespecified parameter $C_{qr} \geqslant 1$. In most cases, the QR decomposition with greedy column pivoting can also be used to construct an ID approximation and can usually obtain a well-bounded U.

Analytic methods As illustrated previously in Section 1.2.2, a degenerate approximation $\Psi(x)^T\Phi(y)$ of $K(x, y)$ in $\mathcal{X} \times \mathcal{Y}$ provides an efficient way to construct a low-rank approximation of $K(X_0, Y_0)$ with any $X_0 \in \mathcal{X}$ and $Y_0 \in \mathcal{Y}$. Specifically, if this degenerate approximation has degree r and accuracy ε, a rank-r approximation of $K(X_0, Y_0)$ with error $O(\varepsilon)$ is constructed by evaluating $\Psi(X_0)$ and $\Phi(Y_0)$. Such an approach, referred to be *analytic*, has computation cost $O(r(m+n))$. Analytic low-rank approximation methods have been commonly used in many fast matrix-vector multiplication algorithms for kernel matrices that are equivalent to multiplying the matrices in specific rank-structured matrix formats, such as FMM [18, 20, 21], the panel clustering method [22], the fast directional multilevel algorithm [19], and the butterfly method [23]. Common examples of degenerate approximations used in practice include multipole expansions [18], polynomial interpolations [20, 24], and pseudoskeleton approximations [19, 21, 25].

Comparison between algebraic and analytic methods In general, analytic methods require much less computation cost than algebraic methods. Also, the obtained approximation factors, i.e., $\Psi(X_0)$ and $\Phi(Y_0)$, by analytic methods do not have to be stored and can be dynamically computed when needed. In algebraic methods, the approximation factors must be stored.

On the other hand, the degrees of the degenerate approximations used in analytic methods, which equal to the ranks of the obtained low-rank approximations, have to be manually selected to obtain a given approximation accuracy. Given an accuracy threshold, the approximation rank obtained by analytic methods is always larger than the actual numerical rank of $K(X_0, Y_0)$. The difference between this approximation rank and the numerical rank can be significant, especially when X_0 or Y_0 lie in a much smaller subdomain of $\mathcal{X}$ or $\mathcal{Y}$. In comparison, algebraic methods can usually better capture the numerical rank of $K(X_0, Y_0)$.

CHAPTER 2
REVIEW OF THE $\mathcal{H}^2$ MATRIX REPRESENTATION

The basic idea of $\mathcal{H}^2$ matrix technique is to locate numerically low-rank blocks of a matrix by specific *admissibility condition* and then compresses these blocks into low-rank form by a *nested approach*. Usually, dense matrices defined by non-oscillatory interactions between many bodies in low-dimension spaces can be effectively represented in $\mathcal{H}^2$ format. Such an $\mathcal{H}^2$ matrix representation can have cost for both matrix storage and matrix-vector multiplications linear in the matrix dimension. Originally, $\mathcal{H}^2$ format was introduced in Ref [4] as an improvement of $\mathcal{H}$ format [1, 2] to further reduce the storage and multiplication cost in numerical solution of integral equations. It later turns out that the pioneering fast multipole method (FMM) [17, 18] for the fast matrix-vector multiplication of a specific kernel matrix is algebraically equivalent to multiplying by the matrix in $\mathcal{H}^2$ format. $\mathcal{H}^2$ matrix technique is thus sometimes referred to as the algebraic FMM as well. Moreover, although derived by different approaches, hierarchically semi-separable (HSS) [5] matrix format is also equivalent to a special $\mathcal{H}^2$ format that applies the weak admissibility condition (to be explained in Section 2.1). In addition to the reduced storage and multiplication cost, an HSS matrix can also be efficiently factorized and solved [5, 26].

In most existing literature, the structure of $\mathcal{H}^2$ matrices is explained as a combination of the structure of $\mathcal{H}$ matrices with specific restrictions on the low-rank approximation of various matrix blocks in $\mathcal{H}$ matrices, which can be unnatural and difficult to understand. In this chapter, we review the structure of $\mathcal{H}^2$ matrices from a more direct viewpoint. Based on this new viewpoint, we also propose modifications to further improve the original structure of $\mathcal{H}^2$ matrices. These modifications are necessary for the exact equivalence between FMM and $\mathcal{H}^2$ matrices (to be discussed in Chapter 5).

For ease of understanding, this chapter focuses on a kernel matrix $K(X, X)$ defined by a non-oscillatory scalar kernel function $K(x, y)$ that is smooth when $x \neq y$ and a set of points X in a low-dimensional space. Note that $K(X, X)$ can be non-symmetric if $K(x, y)$ is not symmetric. The discussion in this chapter can be easily extended to the general case $K(X, Y)$ with $X \neq Y$. If only reading this chapter for background knowledge, it would be sufficient to read Sections 2.1 to 2.3 and 2.6.

The rest of the chapter is organized as follows.

- Section 2.1 describes the admissibility condition used by $\mathcal{H}^2$ matrices to locate numerically low-rank blocks in $K(X, X)$.
- Section 2.2 describes the hierarchical partitioning of the points in X and the associated hierarchical partitioning of the matrix $K(X, X)$.
- Section 2.3 describes the $\mathcal{H}^2$ matrix representation of $K(X, X)$ in the case of a perfect hierarchical partitioning of X and $K(X, X)$.
- Section 2.4 describes the $\mathcal{H}^2$ matrix representation of $K(X, X)$ in the general case, i.e., with a non-perfect hierarchical partitioning of X and $K(X, X)$, and introduces novel modifications to the original $\mathcal{H}^2$ format.
- Section 2.5 describes the fast matrix-vector multiplication for an $\mathcal{H}^2$ matrix.
- Section 2.6 describes the construction of an $\mathcal{H}^2$ matrix representation.

2.1 Admissibility condition

For a non-oscillatory kernel function $K(x, y)$ that is smooth when $x \neq y$, e.g.,

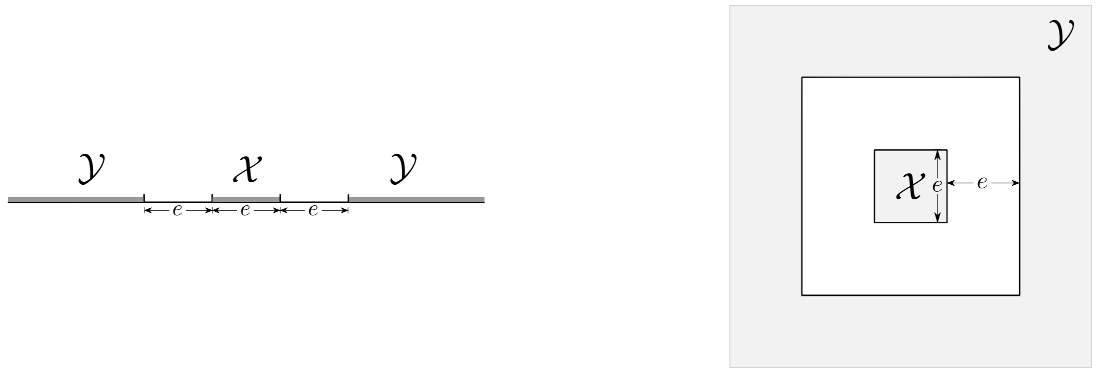
$$K(x,y) = \frac{1}{|x-y|} \quad \text{or} \quad K(x,y) = e^{-|x-y|^2},$$

it is usually the case in a low-dimensional space (e.g., 2D and 3D) that there exists an accurate low-degree degenerate approximation of $K(x, y)$ in a pair of domains $\mathcal{X} \times \mathcal{Y}$ satisfying the geometric condition,

$$\eta \min(\text{diam}(\mathcal{X}), \text{diam}(\mathcal{Y})) \leqslant \text{dist}(\mathcal{X}, \mathcal{Y}), \quad \eta > 0, \tag{2.1}$$

with a reasonably large parameter η, e.g., $\eta = 1/2$. In other words, eq. (2.1) is an admissibility condition for $K(x, y)$ and such a pair of domains $\mathcal{X} \times \mathcal{Y}$ is admissible for $K(x, y)$. A larger η, requiring $\mathcal{X}$ and $\mathcal{Y}$ to be more separated, gives a stronger admissibility condition and requires a smaller degree for an accurate degenerate approximation in $\mathcal{X} \times \mathcal{Y}$. Rigorous study of this observation, including the exact description of applicable kernel functions and the effectiveness of this admissibility condition, can be found in Ref [27].

As to be shown in the next section, an $\mathcal{H}^2$ matrix representation of $K(X, X)$ first partitions X into small boxes of points. Thus, to construct an $\mathcal{H}^2$ matrix, a special box version of eq. (2.1) is used as the *admissibility condition* to locate numerically low-rank blocks in $K(X, X)$. The commonly used one is that $\mathcal{X}$ and $\mathcal{Y}$ are admissible if $\mathcal{X}$ is a box and $\mathcal{Y}$ is the complement of the union of $\mathcal{X}$ and all its adjacent same-sized boxes, or vice versa. Domain $\mathcal{Y}$ is also referred to as the *far field* of $\mathcal{X}$. Examples of such $\mathcal{X} \times \mathcal{Y}$ in 1D and 2D are illustrated in Figure 2.1. This admissibility condition is referred to as the *strong admissibility condition* for $\mathcal{H}^2$ format. An even stronger admissibility condition requires more layers of same-sized boxes between a box domain and its far field. The *weak admissibility condition* simply defines the far field of a box as its complement.



(a) 1D (b) 2D

Figure 2.1: Illustration of the strong admissibility condition in 1D and 2D exploited by $\mathcal{H}^2$ matrices. These admissible pairs of domains are characterized by the edge length parameter $e > 0$ in the figures.

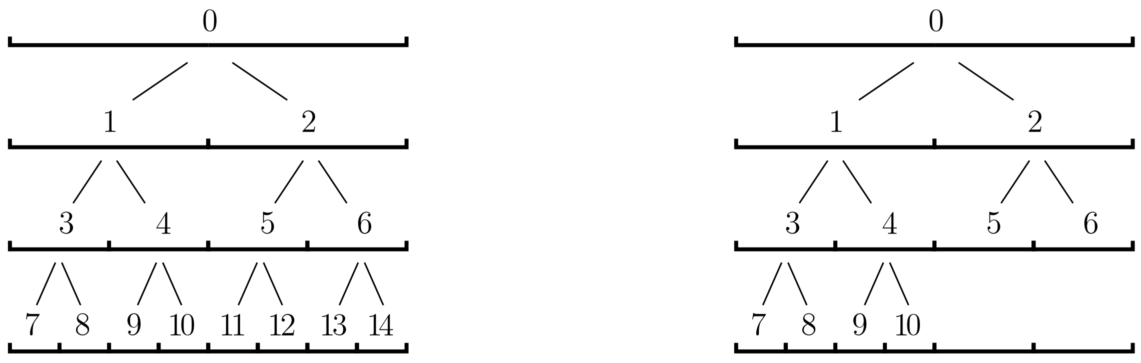
In this chapter, we use the strong admissibility condition to review the structure of $\mathcal{H}^2$ matrices. Using this condition, blocks $K(X_0, Y_0)$ and $K(Y_0, X_0)$ in $K(X, X)$ are identified to be numerically low-rank and are to be compressed into low-rank form for an $\mathcal{H}^2$ matrix representation if X_0 is in a box and Y_0 is in the far field of the box. All the discussion in this chapter can be easily adapted to describe $\mathcal{H}^2$ formats with other admissibility conditions. The main difference is that a stronger admissibility condition locates fewer numerically low-rank blocks but these blocks have smaller numerical ranks. It is worth noting that $\mathcal{H}^2$ format with the weak admissibility condition is exactly HSS format which is to be discussed in more detail in Chapter 6.

For the rest of this chapter, we fix an accuracy threshold ε_0 when describe a matrix to be numerically low-rank and when discuss low-rank approximations of matrices. Moreover, we assume that $K(X_0, Y_0)$ and $K(Y_0, X_0)$ with $X_0 \times Y_0$ in any $\mathcal{X} \times \mathcal{Y}$ satisfying the strong admissibility condition can always be approximated in low-rank form with accuracy $O(\varepsilon_0)$ with a fixed rank r_0. For kernel functions commonly used in practice, this assumption holds true experimentally and can also be analytically justified in some cases. (However, we note that, with the weak admissibility condition, the numerical rank of $K(X_0, Y_0)$ may increase with the absolute sizes of $\mathcal{X}$ and $\mathcal{Y}$.)

2.2 Hierarchical partitioning

The first step to construct an $\mathcal{H}^2$ matrix representation of $K(X, X)$ is to hierarchically partition the points in X into small clusters. Assume X is in a d-dimensional space and $\mathcal{B}$ is a box with equal edges that encloses all the points in X. A hierarchical partitioning of X can be obtained by an adaptive and recursive partitioning of $\mathcal{B}$.

First, $\mathcal{B}$ is partitioned into 2^d smaller same-sized boxes by bisecting all its edges. Within these 2^d boxes, boxes without any point inside are discarded, boxes with the number of points inside less than a prescribed constant n_0 stay untouched hereafter, and any box with the number of points inside greater than n_0 is partitioned into 2^d even-smaller boxes. Recursively, the newly obtained even-smaller boxes are further partitioned in the same manner till all the finest boxes have the number of points inside less than n_0. This hierarchical partitioning of $\mathcal{B}$ can be represented by a 2^d-ary tree whose nodes correspond to the boxes. We choose to number the nodes level-by-level from the root to the leaves of the tree. Figure 2.2 shows two examples of such a partitioning and numbering for points in 1-dimensional space and an associated binary tree.



(a) perfect partition tree (b) non-perfect partition tree

Figure 2.2: Examples of a hierarchical partitioning of points in 1-dimensional space and an associated binary tree: (a) a perfect binary tree for uniformly distributed points (b) a non-perfect binary tree for non-uniformly distributed points.

Let $\mathcal{T}$ denote the partition tree. Let X_i denote the set of points lying in box i and corresponding to node i in the tree. Using this notation, $K(X_i, X_j)$ denotes the block in

$K(X,X)$ corresponding to the interactions between the points in box i and box j. With the partition tree $\mathcal{T}$, X is hierarchically partitioned into $\{X_i\}_{i\in\mathcal{T}}$ and $K(X,X)$ is hierarchically partitioned into blocks $\{K(X_i,X_j)\}_{i,j\in\mathcal{T}}$.

Some basic facts and notation related to the partition tree are listed as follows:

- The root node of $\mathcal{T}$ is at level 0, its children are at level 1, and etc. Denote the bottom level as level L.
- Each leaf node in $\mathcal{T}$ corresponds to a box that has less than n_0 points and that is not further partitioned.
- "A lower level" refers to a level that has nodes further away from the root, or that has a larger level number.
- Boxes in the same level are all of the same size. Boxes in upper levels are larger.
- Let level(l) denote the set of nodes in level l.
- Let level$^+(l)$ denote the union of level(l) and all the leaf nodes above level l. When the tree is full at level l, then level$^+(l) =$ level(l).
- At each level l, all the point sets X_i with $i \in$ level$^+(l)$ are disjoint and their union is exactly X, i.e.,

$$X_i \cap X_j = \emptyset \text{ for } i \neq j \in \text{level}^+(l) \qquad \text{and} \qquad \bigcup_{i\in\text{level}^+(l)} X_i = X.$$

For ease of understanding, we first describe an $\mathcal{H}^2$ matrix representation of $K(X,X)$ with a perfect partition tree in the next section, which has the same structure as the original $\mathcal{H}^2$ format. We then discuss the general case with a non-perfect partition tree in Section 2.4 which contains new modifications to the original $\mathcal{H}^2$ format.

2.3 $\mathcal{H}^2$ matrix with a perfect partition tree

Consider a perfect partition tree $\mathcal{T}$, i.e., every level of $\mathcal{T}$ is full and all the leaves of $\mathcal{T}$ are in level L. In this case, at each level l, level(l) = level$^+(l)$ and there are 2^{dl} nodes in total corresponding to the 2^{dl} same-sized boxes partitioning box $\mathcal{B}$ that encloses X.

At each level l, X is partitioned into 2^{dl} subsets $\{X_i\}_{i\in\text{level}(l)}$. Accordingly, $K(X,X)$ is partitioned into $2^{dl}\times 2^{dl}$ blocks $\{K(X_i,X_j)\}_{i,j\in\text{level}(l)}$. Figure 2.3a illustrates such a matrix partitioning at each level for the 1D example in Figure 2.2a. According to the admissibility condition, for each box $i\in$ level(l), boxes in level(l) can be split into two subsets:

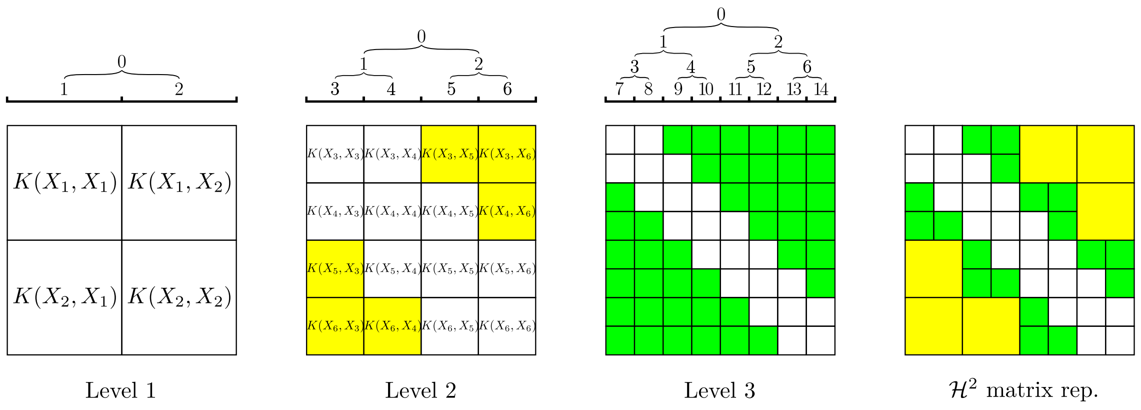
$$\begin{aligned}&\mathcal{F}_i = \{k\in \text{level}(l) \mid \text{box } k \text{ is in the far field of box } i\}\\&\mathcal{N}_i = \text{level}(l)\setminus \mathcal{F}_i.\end{aligned}$$

For example, for node 7 in Figure 2.3a, $\mathcal{F}_7 = \{9,10,11,12,13,14\}$ and $\mathcal{N}_7 = \{7,8\}$. Since $\mathcal{F}_i$ contains all the boxes in level l that are in the far field of box i, X_i and $\cup_{k\in\mathcal{F}_i}X_k$ contain the points in box i and all the points that are in the far field of box i, respectively, and have their associated matrix blocks,

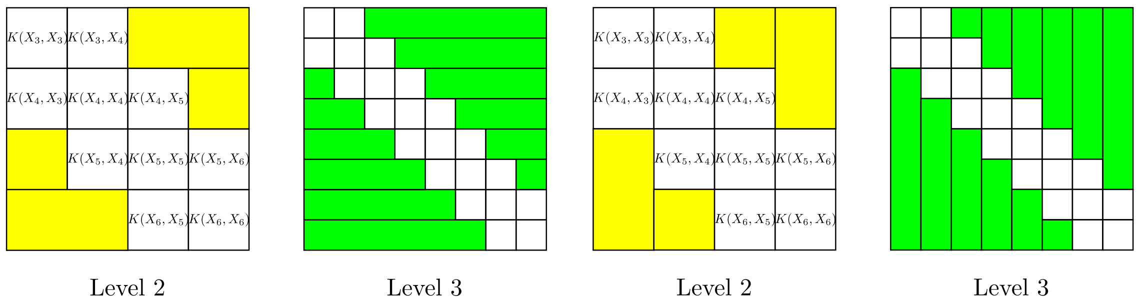
$$K(X_i,\cup_{k\in\mathcal{F}_i}X_k) \quad \text{and} \quad K(\cup_{k\in\mathcal{F}_i}X_k, X_i),$$

to be both numerically low-rank with the rank bounded by r_0 (by the assumption in Section 2.1). Figures 2.3b and 2.3c plot these numerically low-rank blocks in the 1D example.

The two blocks $K(X_i,\cup_{k\in\mathcal{F}_i}X_k)$ and $K(\cup_{k\in\mathcal{F}_i}X_k, X_i)$ having numerical ranks bounded by r_0 suggests that their subblocks $K(X_i,X_j)$ and $K(X_j,X_i)$ with $j\in\mathcal{F}_i$ also have numerical ranks bounded by r_0. In the case of a perfect partition tree $\mathcal{T}$, it can be proved that $j\in\mathcal{F}_i$ is equivalent to $i\in\mathcal{F}_j$ and $j\in\mathcal{N}_i$ is equivalent to $i\in\mathcal{N}_j$. Thus, we only need to consider $K(X_i,X_j)$ with $j\in\mathcal{F}_i$ or $j\in\mathcal{N}_i$ as a generic block in the level-l partitioning of $K(X,X)$.



(a) hierarchical partitioning of $K(X, X)$ and an $\mathcal{H}^2$ matrix representation of $K(X, X)$



(b) blocks $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ (c) blocks $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$

Figure 2.3: Illustration of a 3-level $\mathcal{H}^2$ matrix representation for points in 1-dimensional space with a perfect partition tree. Subfigure (a) plots the admissible blocks (colored) and inadmissible blocks (white) at each level. The final $\mathcal{H}^2$ matrix representation is composed of all the inadmissible blocks at level 3 and some of the admissible blocks at levels 2 and 3. Subfigures (b) and (c) plot the numerically low-rank blocks $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ at each level located by the admissibility condition. There is no low-rank block for nodes 1 and 2 in level 1 since $\mathcal{F}_1$ and $\mathcal{F}_2$ are both empty.

We call $K(X_i, X_j)$ with $j \in \mathcal{F}_i$ an *admissible block* and $K(X_i, X_j)$ with $j \in \mathcal{N}_i$ an *inadmissible block*. Admissible blocks are numerically low-rank based on the above discussion and inadmissible blocks are always assumed to be full-rank. At level l, the $2^{dl} \times 2^{dl}$ blocks in the level-l partitioning of $K(X, X)$ can thus be categorized into admissible blocks $\cup_{i\in\text{level}(l)}\{K(X_i, X_j)\}_{j\in\mathcal{F}_i}$ and inadmissible blocks $\cup_{i\in\text{level}(l)}\{K(X_i, X_j)\}_{j\in\mathcal{N}_i}$. Figure 2.3a plots the admissible and inadmissible blocks at each level for the 1D example.

At any non-leaf level $l \neq L$, an inadmissible block consists of inadmissible blocks and possible admissible blocks at level $l + 1$, while an admissible block consists of admissible blocks in level $l + 1$ (see the example in Figure 2.3a). To get the most storage and multiplication cost reduction by compressing as many and as large admissible blocks as possible into low-rank form, an $\mathcal{H}^2$ matrix representation of $K(X, X)$ with a perfect partition tree $\mathcal{T}$ consists of two parts: (1) dense inadmissible blocks at the leaf level and (2) low-rank approximations of all the admissible blocks at any level that are not contained in larger admissible blocks. Figure 2.3a illustrates these two parts in the 1D example.

The low-rank approximations of different admissible blocks in an $\mathcal{H}^2$ matrix representation are *not independent* but closely connected to each other via two restrictions: (1) the *uniform basis property* and (2) the *nested basis property*. It is worth noting that, if these low-rank approximations are independent and without the two restrictions, the obtained representation of $K(X, X)$ is exactly an $\mathcal{H}$ matrix.

2.3.1 Low-rank approximations of admissible blocks

From the above discussion, $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ for each node $i \in \mathcal{T}$ with nonempty $\mathcal{F}_i$ are numerically low-rank according to the admissibility condition. An $\mathcal{H}^2$ matrix representation essentially focuses on compressing these identified blocks instead of each individual admissible block $K(X_i, X_j)$, which naturally leads to the uniform basis property and the nested basis property.

In this subsection, we introduce the definitions of the two properties and explain how they are derived from the viewpoint of compressing $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ into low-rank form by a nested approach. For simplicity, all these blocks are to be approximated in rank-r_0 form.

Uniform basis property For the uniform basis property, the low-rank approximation of an admissible block $K(X_i, X_j)$ is in the form

$$K(X_i, X_j) \approx U_i B_{i,j} V_j^T, \tag{2.2}$$

where U_i and V_j are matrices consisting of r_0 basis vectors for approximating columns and rows of $K(X_i, X_j)$, respectively, and $B_{i,j}$ is an intermediate matrix that makes this approximation as accurate as possible.

The key observation in eq. (2.2) is that U_i does not depend on X_j and V_j does not depend on X_i. In other words, the low-rank approximation of $K(X_i, X_j)$ shares the same column basis matrix U_i as other admissible blocks that have rows associated with X_i. Similarly, the approximation shares the same row basis matrix V_j as other admissible blocks that have columns associated with X_j. For example, in Figure 2.3a, the low-rank approximations of $K(X_7, X_9)$ and $K(X_7, X_{10})$ share the same column basis matrix U_7. These shared matrices U_i and V_i associated with node i are referred to as the *uniform column basis matrix and uniform row basis matrix*, respectively.

For a node $i \in \mathcal{T}$, observe that $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ is the concatenation of all the admissible blocks that have rows associated with X_i and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ is the concatenation of all the admissible blocks that have columns associated with X_i. For example, in Figure 2.3a, $K(X_7, \cup_{k\in\mathcal{F}_7} X_k)$ is the concatenation of the blocks $K(X_7, X_9)$, $K(X_7, X_{10})$, ..., $K(X_7, X_{14})$. The low-rank approximations of these latter blocks need to share the same column basis matrix U_7 according to the uniform basis property.

The uniform basis property can be naturally derived by constructing the low-rank approximation of an admissible block $K(X_i, X_j)$ based on the low-rank approximations of $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_j} X_k, X_j)$. Here, $K(X_i, X_j)$ is exactly the intersection block of the latter two blocks, as exemplified in Figure 2.4. This specific low-rank approximation approach for all the admissible blocks $K(X_i, X_j)$ is described as follows.

At each level l, consider a rank-r_0 approximation of $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ for each node i at level l as,

$$K(X_i, \cup_{k\in\mathcal{F}_i} X_k) \approx U_i E_i^T = U_i \begin{pmatrix} E_{k_1,i}^T & \dots & E_{k_{f_i},i}^T \end{pmatrix}, \quad \mathcal{F}_i = \{k_1, \dots, k_{f_i}\}, \tag{2.3}$$

where U_i is of dimension $|X_i| \times r_0$ and each $E_{k,i}$ is of dimension $|X_k| \times r_0$ and associated with node $k \in \mathcal{F}_i$. This approximation gives a rank-r_0 approximation to each admissible block $K(X_i, X_k)$ with $k \in \mathcal{F}_i$ as

$$K(X_i, X_k) \approx U_i E_{k,i}^T, \tag{2.4}$$

where U_i does not vary with different nodes k. As a result, all the blocks $K(X_i, X_k)$ with $k \in \mathcal{F}_i$ have their columns close to col(U_i) with the computed U_i in eq. (2.3).

Similarly, consider a rank-r_0 approximation of $K(\cup_{k\in\mathcal{F}_i} X_k, X_j)$ associated with each node j at level l as

$$K(\cup_{k\in\mathcal{F}_j} X_k, X_j) \approx G_j V_j^T = \begin{pmatrix} G_{k_1,j} \\ \vdots \\ G_{k_{f_j},j} \end{pmatrix} V_j^T, \quad \mathcal{F}_j = \{k_1, \ldots, k_{f_j}\}, \tag{2.5}$$

where V_j is of dimension $|X_j| \times r_0$ and each $G_{k,j}$ is of dimension $|X_k| \times r_0$ and associated with node $k \in \mathcal{F}_j$. This approximation gives a rank-r_0 approximation to each admissible block $K(X_k, X_j)$ with $k \in \mathcal{F}_j$ as

$$K(X_k, X_j) \approx G_{k,j} V_j^T, \tag{2.6}$$

where V_j does not vary with different nodes k. Thus, all the blocks $K(X_k, X_j)$ with $k \in \mathcal{F}_j$ have their rows close to col(V_j) with the computed V_j in eq. (2.5).

Recall that $j \in \mathcal{F}_i$ is equivalent to $i \in \mathcal{F}_j$. The above discussion shows that an admissible block $K(X_i, X_j)$ has all the columns close to col(U_i) and all the rows close to col(V_j). Thus, $K(X_i, X_j)$ can be approximated in a rank-r_0 form as

$$K(X_i, X_j) \approx U_i B_{i,j} V_j^T, \tag{2.7}$$

with an $r_0 \times r_0$ *intermediate matrix* $B_{i,j}$ to be computed, which is in the same form as required by the uniform basis property. To minimize the approximation error in eq. (2.7), the optimal $B_{i,j}$ can be computed as $U_i^\dagger K(X_i, X_j)(V_j^T)^\dagger$ by which eq. (2.7) projects the columns of $K(X_i, X_j)$ onto $\operatorname{col}(U_i)$ and the rows onto $\operatorname{col}(V_j)$.

The approach to compressing an admissible block $K(X_i, X_j)$ based on the low-rank approximation of $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_j} X_k, X_j)$ is exemplified in Figure 2.4.

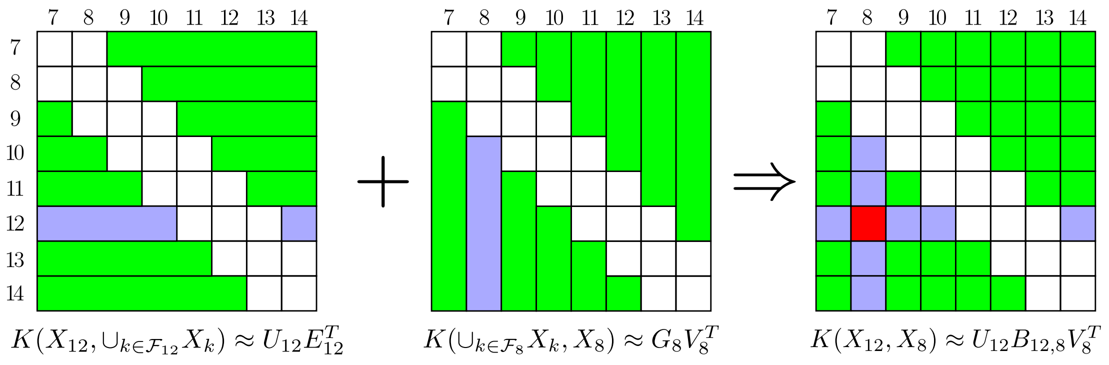

7 8 9 10 11 12 13 14
7 8 9 10 11 12 13 14
$K(X_{12}, \cup_{k\in\mathcal{F}_{12}} X_k) \approx U_{12}E_{12}^T$
$K(\cup_{k\in\mathcal{F}_8} X_k, X_8) \approx G_8 V_8^T$
$K(X_{12}, X_8) \approx U_{12}B_{12,8}V_8^T$

Figure 2.4: Illustration of the low-rank approximation to an admissible block $K(X_{12}, X_8)$ that leads to the uniform basis property based on the 1D example in Figure 2.3.

Nested basis property For the nested basis property, each non-leaf node i with children $\{i_1, i_2, \ldots, i_s\}$ ($s = 2^d$ denotes the number of children) has its uniform column and row basis matrices (associated with the uniform basis property) represented in *nested forms*,

$$U_i = \begin{pmatrix} U_{i_1} & & & \\ & U_{i_2} & & \\ & & \ddots & \\ & & & U_{i_s} \end{pmatrix} R_i, \quad \text{and} \quad V_i = \begin{pmatrix} V_{i_1} & & & \\ & V_{i_2} & & \\ & & \ddots & \\ & & & V_{i_s} \end{pmatrix} S_i, \tag{2.8}$$

for some *transfer matrices* R_i and S_i to be computed. Equation (2.8) is also equivalent to the fact that the column space of U_i is a subspace of a specific concatenation of the column spaces of $U_{i_1}, U_{i_2}, \dots, U_{i_s}$, i.e.,

$$\text{col}(U_i) \subset \left\{ x = \begin{pmatrix} x_1 \\ \vdots \\ x_s \end{pmatrix} \in \mathbb{R}^{|X_i|} \,\middle|\, x_1 \in \text{col}(U_{i_1}), \dots, x_s \in \text{col}(U_{i_s}) \right\}.$$

The same interpretation applies to V_i and $V_{i_1}, V_{i_2}, \dots, V_{i_s}$. Recall that U_i and V_i are basis matrices obtained by compressing $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$, respectively. The nested basis property essentially restricts the selection of basis matrices U_i and V_i for compressing these two associated blocks .

With the nested basis property, the basis matrices at parent nodes are expressed in terms of the basis matrices of their children nodes via the transfer matrices R_i and S_i. Thus, in an $\mathcal{H}^2$ matrix representation, the basis matrices for non-leaf nodes are not explicitly formed and can be recovered recursively from quantities at lower levels of the tree.

The nested basis property can be naturally derived by a nested approach to constructing the low-rank approximations of $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ for all non-leaf nodes i based on the low-rank approximations of $K(X_{i_a}, \cup_{k\in\mathcal{F}_{i_a}} X_k)$ and $K(\cup_{k\in\mathcal{F}_{i_a}} X_k, X_{i_a})$ that are associated with all the children i_a of i. This nested low-rank approximation approach is described as follows.

For a non-leaf node i with its children $\{i_1, i_2, \dots, i_s\}$, it holds that $X_i = X_{i_1} \cup X_{i_2} \cup \dots \cup X_{i_s}$ and thus $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ can be split as

$$K(X_i, \cup_{k\in\mathcal{F}_i} X_k) = \begin{pmatrix} K(X_{i_1}, \cup_{k\in\mathcal{F}_i} X_k) \\ K(X_{i_2}, \cup_{k\in\mathcal{F}_i} X_k) \\ \vdots \\ K(X_{i_s}, \cup_{k\in\mathcal{F}_i} X_k) \end{pmatrix}.$$

Note that any box $k \in \mathcal{F}_i$ is in the far field of box i by definition and thus is also in the far field of each child box i_a of box i. As a result, the set of points in all the boxes $k \in \mathcal{F}_i$ is a subset of all the points in the far field of box i_a, i.e.,

$$\bigcup_{k\in\mathcal{F}_i} X_k \subset \bigcup_{k\in\mathcal{F}_{i_a}} X_k.$$

Therefore, $K(X_{i_a}, \cup_{k\in\mathcal{F}_i} X_k)$ consists of a subset of the columns in $K(X_{i_a}, \cup_{k\in\mathcal{F}_{i_a}} X_k)$. For example, for node 3 and its child node 7 in the 1D example Figure 2.5, $\cup_{k\in\mathcal{F}_3} X_k = X_5 \cup X_6$ is a subset of $\cup_{k\in\mathcal{F}_7} X_k = X_9 \cup X_{10} \ldots \cup X_{14}$ and $K(X_7, X_5 \cup X_6)$ consists of a subset of the columns in $K(X_7, X_9 \cup X_{10} \ldots \cup X_{14})$.

For each child i_a of i, assume that $K(X_{i_a}, \cup_{k\in\mathcal{F}_{i_a}} X_k)$ has been approximated by $U_{i_a} E_{i_a}^T$ as in eq. (2.3) which gives a rank-r_0 approximation of its subblock $K(X_{i_a}, \cup_{k\in\mathcal{F}_i} X_k)$ as

$$K(X_{i_a}, \cup_{k\in\mathcal{F}_i} X_k) \approx U_{i_a} E_{\mathcal{F}_i, i_a}^T.$$

where $E_{\mathcal{F}_i, i_a}^T$ denotes the columns of $E_{i_a}^T$ associated with $\cup_{k\in\mathcal{F}_i} X_k$. Plug this approximation with each child node i_a into $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and we obtain

$$K(X_i, \cup_{k\in\mathcal{F}_i} X_k) \approx \begin{pmatrix} U_{i_1} E_{\mathcal{F}_i, i_1}^T \\ U_{i_2} E_{\mathcal{F}_i, i_2}^T \\ \vdots \\ U_{i_s} E_{\mathcal{F}_i, i_s}^T \end{pmatrix} = \begin{pmatrix} U_{i_1} & & & \\ & U_{i_2} & & \\ & & \ddots & \\ & & & U_{i_s} \end{pmatrix} \begin{pmatrix} E_{\mathcal{F}_i, i_1}^T \\ E_{\mathcal{F}_i, i_2}^T \\ \vdots \\ E_{\mathcal{F}_i, i_s}^T \end{pmatrix}. \tag{2.9}$$

Instead of directly compressing $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$, we can compute a rank-$r_0$ approximation of the last matrix above as

$$\begin{pmatrix} E_{\mathcal{F}_i, i_1} & E_{\mathcal{F}_i, i_2} & \ldots & E_{\mathcal{F}_i, i_s} \end{pmatrix}^T \approx R_i E_i^T. \tag{2.10}$$

Plugging eq. (2.10) into eq. (2.9), we have a rank-r_0 approximation of $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ as

$$K(X_i, \cup_{k\in\mathcal{F}_i} X_k) \approx \begin{pmatrix} U_{i_1} & & & \\ & U_{i_2} & & \\ & & \ddots & \\ & & & U_{i_s} \end{pmatrix} R_i E_i^T = U_i E_i^T,$$

where the uniform column basis matrix U_i for node i is exactly defined as in eq. (2.8) using the computed R_i in eq. (2.10).

The approximated matrix in eq. (2.10) has far fewer rows than $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and, in fact, the number of rows equals $r_0 s$ which does not depend on the size of X_i. As a result, this nested approach to compressing $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ for a non-leaf node i is much cheaper than directly compressing $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$. Further, the computed U_i by this approach is exactly in the same form as required by the nested basis property for U_i.

Similarly, to compress $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$, the matrix can be first split and approximated based on the low-rank approximations associated with the children of i as

$$\begin{aligned} K(\cup_{k\in\mathcal{F}_i} X_k, X_i) &= \begin{pmatrix} K(\cup_{k\in\mathcal{F}_i} X_k, X_{i_1}) & K(\cup_{k\in\mathcal{F}_i} X_k, X_{i_2}) & \cdots & K(\cup_{k\in\mathcal{F}_i} X_k, X_{i_s}) \end{pmatrix} \\ &\approx \begin{pmatrix} G_{\mathcal{F}_i,i_1} V_{i_1}^T & G_{\mathcal{F}_i,i_2} V_{i_2}^T & \cdots & G_{\mathcal{F}_i,i_s} V_{i_s}^T \end{pmatrix}. \end{aligned} \tag{2.11}$$

Then compute a rank-r_0 approximation of $\left(G_{\mathcal{F}_i,i_1},\ G_{\mathcal{F}_i,i_2},\ \cdots,\ G_{\mathcal{F}_i,i_s}\right)$ as

$$\left(G_{\mathcal{F}_i,i_1},\ G_{\mathcal{F}_i,i_2},\ \cdots,\ G_{\mathcal{F}_i,i_s}\right) \approx G_i S_i^T. \tag{2.12}$$

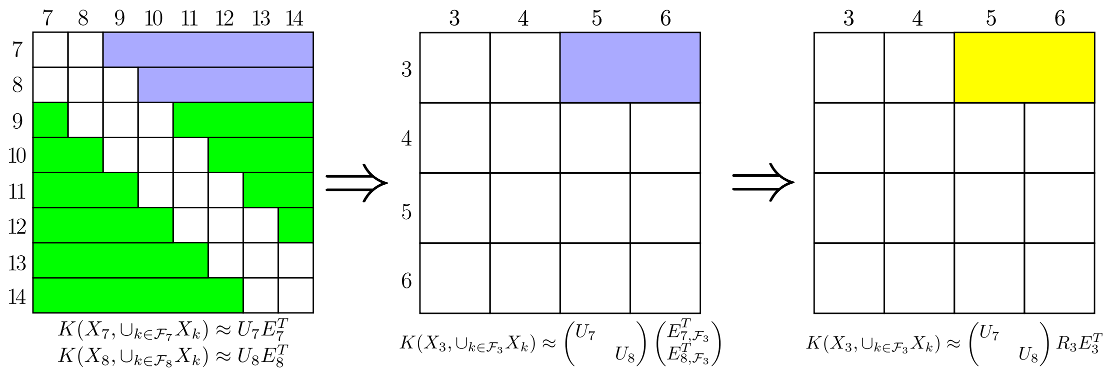
Plugging eq. (2.12) into eq. (2.11), $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ is approximated in the rank-r_0 form

$$K(\cup_{k\in\mathcal{F}_i} X_k, X_i) = G_i S_i^T \begin{pmatrix} V_{i_1}^T & & & \\ & V_{i_2}^T & & \\ & & \ddots & \\ & & & V_{i_s}^T \end{pmatrix} = G_i V_i^T, \tag{2.13}$$

where the uniform row basis matrix V_i for node i is exactly defined as in eq. (2.8) using the computed S_i in eq. (2.12).

Again, the approximated matrix in eq. (2.12) has $r_0 s$ columns and this nested approach to compressing $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ is much cheaper than directly compressing the matrix. The computed V_i is exactly in the form as required by the nested basis property for V_i.

This nested approach to compressing $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ is exemplified in Figure 2.5.



7 8 9 10 11 12 13 14
$K(X_7, \cup_{k\in\mathcal{F}_7} X_k) \approx U_7 E_7^T$
$K(X_8, \cup_{k\in\mathcal{F}_8} X_k) \approx U_8 E_8^T$
3 4 5 6
$K(X_3, \cup_{k\in\mathcal{F}_3} X_k) \approx \begin{pmatrix} U_7 & \\ & U_8 \end{pmatrix} \begin{pmatrix} E_{7,\mathcal{F}_3}^T \\ E_{8,\mathcal{F}_3}^T \end{pmatrix}$
3 4 5 6
$K(X_3, \cup_{k\in\mathcal{F}_3} X_k) \approx \begin{pmatrix} U_7 & \\ & U_8 \end{pmatrix} R_3 E_3^T$

Figure 2.5: Illustration of the nested low-rank approximation of block $K(X_3, \cup_{k\in\mathcal{F}_3} X_k)$ that leads to the nested basis property based on the 1D example in Figure 2.3a.

2.3.2 Summary of an $\mathcal{H}^2$ matrix representation

An $\mathcal{H}^2$ matrix representation of $K(X, X)$ constructs the low-rank approximations of blocks $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ for all the nodes $i \in \mathcal{T}$ with nonempty $\mathcal{F}_i$ using a nested approach from the leaves to the root of the tree (see Algorithm 1). Based on these low-rank approximations, each admissible block $K(X_i, X_j)$ can be approximated as

$$K(X_i, X_j) \approx U_i B_{ij} V_j^T, \quad j \in \mathcal{F}_i \tag{2.14}$$

where U_i and V_i for each non-leaf node i are not formed but recursively defined as

$$U_i = \begin{pmatrix} U_{i_1} & & & \\ & U_{i_2} & & \\ & & \ddots & \\ & & & U_{i_s} \end{pmatrix} R_i \quad \text{and} \quad V_i = \begin{pmatrix} V_{i_1} & & & \\ & V_{i_2} & & \\ & & \ddots & \\ & & & V_{i_s} \end{pmatrix} S_i \tag{2.15}$$

with the associated transfer matrices R_i and S_i. The optimal intermediate matrix $B_{i,j}$ can be computed as $U_i^\dagger K(X_i, X_j)(V_j^T)^\dagger$. As to be shown in Section 2.6, this optimal $B_{i,j}$ can be efficiently constructed without accessing all the entries in $K(X_i, X_j)$.

Algorithm 1 Low-rank approximations of $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$

```
1: for $l = L, L-1, \dots, 1$ do
2:     for all node $i$ at level $l$ do
3:         if $i$ is a leaf node then
4:             • compute $K(X_i, \cup_{k\in\mathcal{F}_i} X_k) \approx U_i E_i^T$.
5:             • compute $K(\cup_{k\in\mathcal{F}_i} X_k, X_i) \approx G_i V_i^T$.
6:         else
7:             • compute $\begin{pmatrix} E_{\mathcal{F}_i,i_1} & E_{\mathcal{F}_i,i_2} & \dots & E_{\mathcal{F}_i,i_s} \end{pmatrix}^T \approx R_i E_i^T$.
8:             • compute $\begin{pmatrix} G_{\mathcal{F}_i,i_1} & G_{\mathcal{F}_i,i_2} & \cdots & G_{\mathcal{F}_i,i_s} \end{pmatrix} \approx G_i S_i^T$.
9:         end if
10:     end for
11: end for
```

An $\mathcal{H}^2$ matrix representation is made up by dense inadmissible blocks at level L and low-rank approximations eq. (2.14) of the admissible blocks at any level that are not contained in larger admissible blocks. The components stored by an $\mathcal{H}^2$ matrix include:

- Uniform column and row basis matrices U_i and V_i for each leaf node i.
- Transfer matrices R_i and S_i for each non-leaf node i with a nonempty $\mathcal{F}_i$.
- Intermediate matrices $B_{i,j}$ for the low-rank approximation eq. (2.14) of each admissible block $K(X_i, X_j)$ at any level that is not contained in a larger admissible block.
- Inadmissible blocks $K(X_i, X_j)$ with i and j being leaf nodes.

Note that if $K(x, y)$ is symmetric, we have $K(X_i, \cup_{k\in\mathcal{F}_i} X_k) = K(\cup_{k\in\mathcal{F}_i} X_k, X_i)^T$. In this case, we only need to compute the low-rank approximation of each $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and set $V_i = U_i$ for leaf nodes and $S_i = R_i$ for non-leaf nodes.

2.3.3 Storage cost

We estimate the storage cost of the above $\mathcal{H}^2$ matrix components in a simplified setting. Consider a perfect L-level 2^d-ary partition tree. Assume that there are exactly n_0 points inside each leaf box and thus there are $N = 2^{dL} n_0$ points in total. Also, it is common in practice to set the parameter n_0 bounding the number of points in each leaf box to $O(r_0)$.

Uniform basis matrices U_i and V_i for each node i at the leaf level are of dimension $n_0 \times r_0$ and thus their total storage cost is

$$2 \times \underbrace{n_0 r_0}_{\text{storage per node}} \times \underbrace{2^{dL}}_{\text{\# of nodes}} = 2Nr_0.$$

Transfer matrices R_i and W_i for each node i at level l with $L < l \leqslant 2$ ($\mathcal{F}_i$ is empty for any node i in level 1) are of dimension $2^d r_0 \times r_0$ and thus their total storage cost is

$$2 \sum_{l=2}^{L-1} \underbrace{2^d r_0^2}_{\text{storage per node}} \times \underbrace{2^{dl}}_{\text{\# of nodes}} \approx 2r_0^2 2^{dL} \approx 2Nr_0.$$

For each node i, $K(X_i, X_j)$ with $j \in \mathcal{F}_i$ is not contained in a larger admissible block only if the parents of i and j are not in the far field of each other. There are at most $6^d - 3^d$ boxes j satisfying that box j is at the same level as box i and is non-adjacent to box i while their parents are. Thus, the total storage cost of $B_{i,j}$ is estimated as

$$\sum_{l=2}^{L} \underbrace{(6^d - 3^d)}_{\text{\# of boxes}} \times \underbrace{r_0^2}_{\text{storage of } B_{i,j}} \times \underbrace{2^{dl}}_{\text{\# of nodes}} \approx (6^d - 3^d) r_0^2 2^{Ld} \approx (6^d - 3^d) N r_0.$$

For each leaf node i, there are at most 3^d boxes j in $\mathcal{N}_i$. Thus, the total storage cost of inadmissible blocks at the leaf level is bounded by

$$\underbrace{2^{dL}}_{\text{\# of nodes}} \times \underbrace{3^d}_{\text{\# of boxes}} \times \underbrace{n_0^2}_{\text{storage per block}} \approx 3^d N r_0.$$

In total, the storage cost for an $\mathcal{H}^2$ matrix representation is around $2Nr_0 + 2Nr_0 + (6^d - 3^d)Nr_0 + 3^d N r_0 \approx 6^d N r_0$ which is linear in the matrix dimension N.

2.4 $\mathcal{H}^2$ matrix with a non-perfect partition tree

This section describes the structure of an $\mathcal{H}^2$ matrix representation with a non-perfect partition tree, which contains novel modifications to the original structure of $\mathcal{H}^2$ matrices [4, 27]. These modifications are naturally derived from the viewpoint of $\mathcal{H}^2$ matrix structure used in the last subsection. Only with these modifications, the $\mathcal{H}^2$ matrix representation can be exactly equivalent to FMM (see Chapter 5).

Consider a non-perfect partition tree $\mathcal{T}$. Recall that level$^+(l)$ is the union of nodes at level l, i.e., level(l), and the leaf nodes above level l. At each level l, X is partitioned into subsets that correspond to the nodes in level$^+(l)$, i.e.,

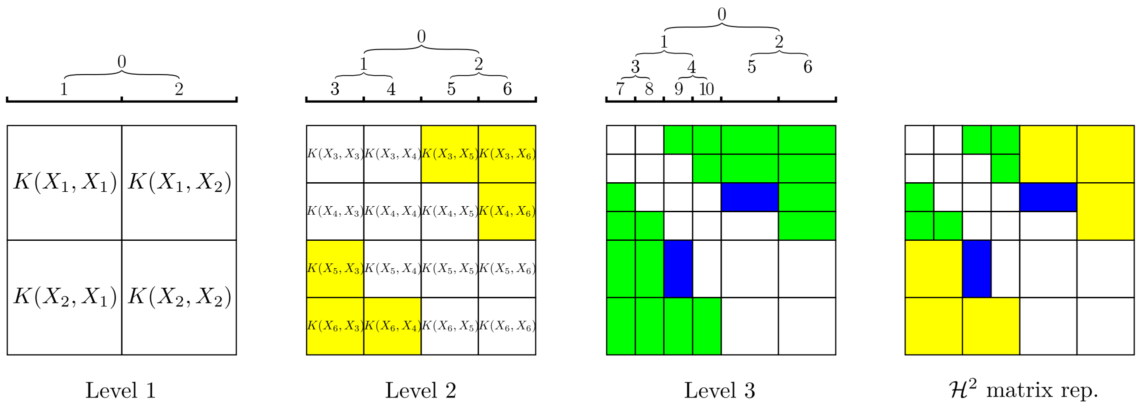
$$X_i \cap X_j = \emptyset, \quad \text{for } i \neq j \in \text{level}^+(l), \qquad \text{and} \quad X = \bigcup_{i \in \text{level}^+(l)} X_i,$$

and $K(X, X)$ is partitioned into $\{K(X_i, X_j)\}_{i,j \in \text{level}^+(l)}$. Figure 2.6a illustrates such a matrix partitioning in a 1D example. In this example, level$(3) = \{7, 8, 9, 10\}$ and level$^+(3) = \{5, 6, 7, 8, 9, 10\}$. According to the admissibility condition, for each box $i \in$ level(l) (not level$^+(l)$), boxes in level$^+(l)$ can be split into two subsets:

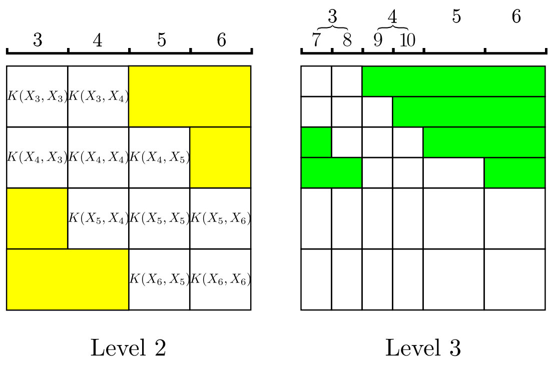
$$\begin{aligned}
\mathcal{F}_i &= \{k \in \text{level}^+(l) \mid \text{box } k \text{ is in the far field of box } i\} \\
\mathcal{N}_i &= \text{level}^+(l) \setminus \mathcal{F}_i.
\end{aligned}$$

For each node $k \in$ level$^+(l) \setminus$ level(l), i.e., a leaf node above level l, $\mathcal{N}_k$ and $\mathcal{F}_k$ are defined at the level where node k lies, say level l_k, and they split level$^+(l_k)$ instead. We note that $j \in \mathcal{F}_i$ may not lead to $i \in \mathcal{F}_j$ anymore since j might be in level$^+(l) \setminus$ level(l). In this case, $\mathcal{F}_j$ only contains nodes at or above the level where j lies and node i is not in $\mathcal{F}_j$.

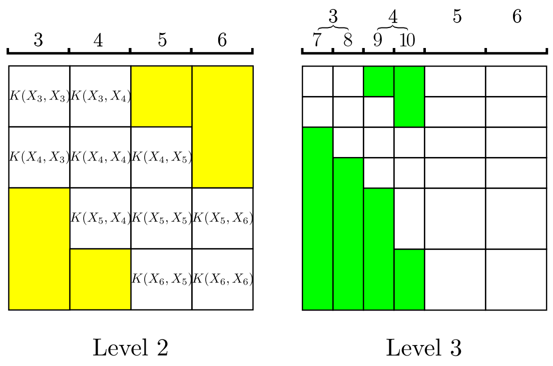
With this newly defined $\mathcal{F}_i$, the numerically low-rank blocks associated with each node $i \in$ level(l) are still $K(X_i, \cup_{j \in \mathcal{F}_i} X_j)$ and $K(\cup_{j \in \mathcal{F}_i} X_j, X_i)$. Note that these two blocks associated with a node $i \in$ level$^+(l) \setminus$level(l) are characterized at an upper level. Figures 2.6b



(a) hierarchical partitioning of $K(X, X)$ and the $\mathcal{H}^2$ matrix representation of $K(X, X)$



(b) blocks $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ with $i \in$ level(l)



(c) blocks $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ with $i \in$ level(l)

Figure 2.6: Illustration of a 3-level $\mathcal{H}^2$ matrix representation for points in 1-dimensional space with a non-perfect partition tree. In (a), inadmissible blocks are white at all levels, level 2 has admissible blocks (yellow), and level 3 has admissible blocks (green) and partially admissible blocks (blue). In (b) and (c), $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ are plotted at each level. Note that the partially admissible block $K(X_9, X_5)$ is colored in (b) but not in (c), meaning that $K(X_9, X_5)$ only shares the column basis matrix U_9.

and 2.6c plot these blocks in the 1D example.

Just like in the case of a perfect partition tree, an $\mathcal{H}^2$ matrix representation of $K(X, X)$ compresses blocks $K(X_i, \cup_{j\in\mathcal{F}_i} X_j)$ and $K(\cup_{j\in\mathcal{F}_i} X_j, X_i)$ associated with all the nodes i with non-empty $\mathcal{F}_i$ into the low-rank forms $U_i E_i^T$ and $G_i V_i^T$, respectively. For any non-leaf node i, it still holds that the set of points in all the boxes $k \in \mathcal{F}_i$ is a subset of all the points in the far field of any child box of i, i.e.,

$$\bigcup_{k\in\mathcal{F}_i} X_k \subset \bigcup_{k\in\mathcal{F}_{i_a}} X_k, \quad i_a = i_1, i_2, \ldots, i_s.$$

Thus, blocks $K(X_i, \cup_{j\in\mathcal{F}_i} X_j)$ and $K(\cup_{j\in\mathcal{F}_i} X_j, X_i)$ can still be compressed by the nested approach described in Algorithm 1 with the newly defined $\mathcal{F}_i$ based on level$^+(l)$.

The supplementary modifications to the structure of an $\mathcal{H}^2$ matrix with a non-perfect partition tree is over the low-rank approximations of each individual block $K(X_i, X_j)$ with $j \in \mathcal{F}_i$ or $i \in \mathcal{F}_j$. For node $i \in$ level(l), $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ being numerically low-rank suggests that $K(X_i, X_j)$ and $K(X_j, X_i)$ with $j \in \mathcal{F}_i$ are also numerically low-rank. To further characterize these latter blocks, $\mathcal{F}_i$ is split into to two subsets:

$$\mathcal{F}_i^1 = \{k \in \mathcal{F}_i \mid \text{box } i \text{ is in the far field of box } k\}$$
$$\mathcal{F}_i^2 = \{k \in \mathcal{F}_i \mid \text{box } i \text{ is } \textbf{not in} \text{ the far field of box } k\}.$$

Geometrically, $\mathcal{F}_i^2$ only contains the leaf boxes k above level l that are separated from box i but adjacent to an ancestor of box i. Meanwhile, $\mathcal{F}_i^1$ contains all the boxes in $\mathcal{F}_i$ that are at level l and also contains qualified leaf boxes in $\mathcal{F}_i$ that are above level l. For example, for node 9 in Figure 2.6a, $\mathcal{F}_9^1 = \{7, 6\}$ and $\mathcal{F}_9^2 = \{5\}$. Noting that $\cup_{k\in\mathcal{F}_j} X_k$ is the set of all the points in the far field of a box j, it can thus be verified that

$$X_i \subset \bigcup_{k\in\mathcal{F}_j} X_k, \quad \forall j \in \mathcal{F}_i^1,$$
$$X_i \bigcap \bigcup_{k\in\mathcal{F}_j} X_k = \emptyset, \quad \forall j \in \mathcal{F}_i^2.$$

Given the splitting $\mathcal{F}_i = \mathcal{F}_i^1 \cup \mathcal{F}_i^2$, the numerically low-rank blocks $K(X_i, X_j)$ and $K(X_j, X_i)$ with $j \in \mathcal{F}_i$ are categorized into two classes:

1. *admissible blocks:* $K(X_i, X_j)$ and $K(X_j, X_i)$ with $j \in \mathcal{F}_i^1$

2. *partially admissible blocks:*$K(X_i, X_j)$ and $K(X_j, X_i)$ with $j \in \mathcal{F}_i^2$.

For $j \in \mathcal{F}_i^1$, $K(X_i, X_j)$ is a subblock of both $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_j} X_k, X_j)$. Based on the low-rank approximations of the latter two blocks, $K(X_i, X_j)$ has its columns close to col(U_i) and rows close to col(V_j) and thus can be approximated as

$$K(X_i, X_j) \approx U_i B_{i,j} V_j^T. \tag{2.16}$$

Similarly, $K(X_j, X_i)$ is a subblock of both $K(X_j, \cup_{k\in\mathcal{F}_j} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ and can be approximated as $K(X_j, X_i) \approx U_j B_{j,i} V_i^T$.

For $j \in \mathcal{F}_i^2$, $K(X_i, X_j)$ is a subblock of $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ but not of $K(\cup_{k\in\mathcal{F}_j} X_k, X_j)$ (recall that $X_i \cap \cup_{k\in\mathcal{F}_j} X_k = \emptyset$). As a result, $K(X_i, X_j)$ has its columns close to col(U_i) but may not have its rows close to col(V_j) with the computed V_j for $K(\cup_{k\in\mathcal{F}_j} X_k, X_j)$. Thus, $K(X_i, X_j)$ can only be approximated as

$$K(X_i, X_j) \approx U_i B_{i,j}. \tag{2.17}$$

Similarly, $K(X_j, X_i)$ is not a subblock of $K(X_j, \cup_{k\in\mathcal{F}_j} X_k)$ and can be approximated as

$$K(X_j, X_i) \approx B_{j,i} V_i^T. \tag{2.18}$$

In other words, partially admissible blocks only satisfy the uniform basis property partially, sharing either row basis matrix or column basis matrix with other related blocks.

To summarize, at level l, $K(X_i, X_j)$ with either i or j in level(l) are categorized into (1) inadmissible blocks with $i \in \mathcal{N}_j$ or $j \in \mathcal{N}_i$, (2) admissible blocks with $i \in \mathcal{F}_j^1$ or $j \in \mathcal{F}_i^1$, and (3) partially admissible blocks with $i \in \mathcal{F}_j^2$ or $j \in \mathcal{F}_i^2$. For a perfect partition tree, $\mathcal{F}_i^2$ is always empty and there is no partially admissible block. In the end, an $\mathcal{H}^2$ matrix with a general partition tree $\mathcal{T}$ consists of three parts:

1. dense inadmissible blocks $K(X_i, X_j)$ with both i and j being leaf nodes.

2. low-rank approximations $U_i B_{i,j} V_j^T$ of the admissible blocks $K(X_i, X_j)$ that are not contained in larger admissible blocks.

3. low-rank approximations $U_i B_{i,j}$ (for $j \in \mathcal{F}_i^2$) or $B_{i,j} V_j^T$ (for $i \in \mathcal{F}_j^2$) of the partially inadmissible blocks $K(X_i, X_j)$ that are not contained in larger partially inadmissible blocks.

Denote the sets of the associated pairs of nodes (i, j) for these three sets of blocks $K(X_i, X_j)$ as $\mathcal{D}$, $\mathcal{A}$, and $\mathcal{A}_p$, respectively. These three sets of blocks exactly form a non-overlapping partitioning of $K(X, X)$ (see Figure 2.6a for an example).

In this general $\mathcal{H}^2$ matrix representation, U_i and V_i are constructed by Algorithm 1 and each intermediate matrix $B_{i,j}$ should make the corresponding approximation accurate. In addition to the $\mathcal{H}^2$ components discussed in Section 2.3.1, only intermediate matrices $B_{i,j}$ corresponding to the partially admissible blocks, i.e., $(i, j) \in \mathcal{A}_p$, are newly introduced.

Original $\mathcal{H}^2$ matrix structure In the original $\mathcal{H}^2$ matrix structure, there is no distinction between partially admissible blocks and admissible blocks. For any two boxes i and j in level$^+(l)$ satisfying $i \in \mathcal{F}_j$ or $j \in \mathcal{F}_i$, $K(X_i, X_j)$ is called an "admissible block" and is approximated as $U_i B_{i,j} V_j^T$.

Let $\mathcal{J}_i$ denote the set of boxes j satisfying that $i \in \mathcal{F}_j^2$, i.e., box i is in the far field of box j while box j is not in the far field of box i. Note that $K(X_i, X_j)$ for any $j \in \mathcal{J}_i$ is not a subblock of $K(X_i, \cup_{k \in \mathcal{F}_i} X_k)$. Thus, to approximate $K(X_i, X_j)$ by $U_i B_{i,j} V_j^T$, the original $\mathcal{H}^2$ matrix structure has to compute an effective uniform column basis matrix U_i by compressing $K(X_i, \cup_{k \in \mathcal{F}_i \cup \mathcal{J}_i} X_k)$ instead of $K(X_i, \cup_{k \in \mathcal{F}_i} X_k)$. Since $\cup_{k \in \mathcal{F}_i \cup \mathcal{J}_i} X_k$ contains points not in the far field of box i, $K(X_i, \cup_{k \in \mathcal{F}_i \cup \mathcal{J}_i} X_k)$ can have much larger numerical rank than $K(X_i, \cup_{k \in \mathcal{F}_i} X_k)$, leading to more columns in U_i and thus less cost reduction by the final $\mathcal{H}^2$ matrix representation.

2.5 $\mathcal{H}^2$ matrix-vector multiplication

With an accurate $\mathcal{H}^2$ matrix representation of $K(X, X)$, the multiplication of $K(X, X)$ with a vector can be approximated by the multiplication of the $\mathcal{H}^2$ matrix with the vector. The fast matrix-vector multiplication algorithm for the $\mathcal{H}^2$ matrix (abbreviated as $\mathcal{H}^2$-matvec) is described as follows.

Consider a vector q to be multiplied by $K(X,X)$ and the result vector b. For each $i \in \mathcal{T}$, let q_i and b_i denote the subvectors of q and b, respectively, whose entry indices are the column indices of $K(X, X_i)$ within $K(X,X)$. The basic idea of $\mathcal{H}^2$-matvec is to traverse all the three sets of blocks $K(X_i, X_j)$ in the $\mathcal{H}^2$ matrix, i.e., the blocks characterized by $\mathcal{D}$, $\mathcal{A}$, and $\mathcal{A}_p$, and accumulate the corresponding multiplications, i.e.,

$$b_i = b_i + K(X_i, X_j)q_j, \quad (i,j) \in \mathcal{D} \cup \mathcal{A} \cup \mathcal{A}_p,$$

with $K(X_i, X_j)$ in dense form or in low-rank form.

For inadmissible blocks $K(X_i, X_j)$ with $(i,j) \in \mathcal{D}$, the computation is straightforward with the blocks in dense form as

$$b_i = b_i + \sum_{(i,j)\in\mathcal{D}} K(X_i, X_j)q_j, \quad \text{for each leaf node } i \in \mathcal{T}.$$

For each admissible block $K(X_i, X_j)$ with $(i,j) \in \mathcal{A}$, its multiplication by q_j, i.e.,

$$b_i = b_i + K(X_i, X_j)q_j \approx b_i + U_i B_{i,j} V_j^T q_j,$$

is computed in three steps $V_j^T q_j$, $B_{i,j}(V_j^T q_j)$, and $b_i = b_i + U_i \left(B_{i,j}\left(V_j^T q_j\right)\right)$ which correspond to three consecutive steps of $\mathcal{H}^2$-matvec: forward transformation, intermediate multiplication, and backward transformation.

Forward transformation This phase computes $y_j = V_j^T q_j$ for all the nodes $j \in \mathcal{T}$. Note that y_j can be used for all the admissible blocks with columns defined by X_j. For each leaf node j, y_j is directly computed. For each non-leaf node j with children $\{j_1, j_2, \dots, j_s\}$, y_j

is recursively computed using the computed results $y_{j_1}, y_{j_2}, \dots, y_{j_s}$ as

$$y_j = V_j^T q_j = S_j^T \begin{pmatrix} V_{j_1}^T & & & \\ & V_{j_2}^T & & \\ & & \ddots & \\ & & & V_{j_s}^T \end{pmatrix} \begin{pmatrix} q_{j_1} \\ q_{j_2} \\ \vdots \\ q_{j_s} \end{pmatrix} = S_j^T \begin{pmatrix} V_{j_1}^T q_{j_1} \\ V_{j_2}^T q_{j_2} \\ \vdots \\ V_{j_s}^T q_{j_s} \end{pmatrix} = S_j^T \begin{pmatrix} y_{j_1} \\ y_{j_2} \\ \vdots \\ y_{j_s} \end{pmatrix}.$$

This phase thus computes y_j recursively by traversing the partition tree from the leaves to the root. See the lines 6-14 in Algorithm 2 for the exact computation.

Intermediate multiplication This phase first computes $z_{i,j} = B_{i,j} y_j$ for each admissible block $K(X_i, X_j)$ with $(i,j) \in \mathcal{A}$. Note that all the $z_{i,j}$ that share the index i are to be multiplied by U_i and then added to b_i as

$$b_i = b_i + \sum_{(i,j)\in\mathcal{A}} U_i z_{i,j} = b_i + U_i \sum_{(i,j)\in\mathcal{A}} z_{i,j}.$$

Only multiplying U_i once, it is more efficient to first sum over all these $z_{i,j}$, then apply U_i, and lastly add to b_i. Thus, for each node $i \in \mathcal{T}$, this phase further computes

$$z_i = \sum_{(i,j)\in\mathcal{A}} z_{i,j} = \sum_{(i,j)\in\mathcal{A}} B_{i,j} y_j.$$

Backward transformation This phase computes $b_i = b_i + U_i z_i$ for each node $i \in \mathcal{T}$. For a non-leaf node i with children $\{i_1, i_2, \dots, i_s\}$, b_i is recursively accumulated as

$$b_i = b_i + U_i z_i = b_i + \begin{pmatrix} U_{i_1} & & & \\ & U_{i_2} & & \\ & & \ddots & \\ & & & U_{i_s} \end{pmatrix} R_i z_i = b_i + \begin{pmatrix} U_{i_1}[R_i z_i]_{i_1} \\ U_{i_2}[R_i z_i]_{i_2} \\ \vdots \\ U_{i_s}[R_i z_i]_{i_s} \end{pmatrix},$$

where $[R_i z_i]_{i_a}$ denote the subvector of $R_i z_i$ associated with U_{i_a}. The addition $b_i = b_i + U_i z_i$ is reduced to $b_{i_a} = b_{i_a} + U_{i_a}[R_i z_i]_{i_a}$ with all the children of i. Meanwhile, $b_{i_a} = b_{i_a} + U_{i_a} z_{i_a}$ also has to be computed for each child i_a. Only multiplying U_{i_a} once, it is more efficient to first overwrite z_{i_a} by $z_{i_a} = z_{i_a} + [R_i z_i]_{i_a}$ and then multiply U_{i_a} by z_{i_a}.

Recursively, this phase traverses the tree from the root to the leaves to overwrite each z_i by $z_i = z_i + [R_p z_p]_i$ with p being the parent of i. As a result, for each leaf node i, z_i accumulates the intermediate multiplication results from all its ancestors. Adding $U_i z_i$ to b_i for all the leaf nodes i in $\mathcal{T}$ finishes this phase. See the lines 21-28 in Algorithm 2 for the exact calculation.

For each partially admissible block $K(X_i, X_j)$ with $(i,j) \in \mathcal{A}_p$, its multiplication can be merged into the above three-step multiplication of admissible blocks as follows.

- For $K(X_i, X_j)$ approximated by $U_i B_{i,j}$, i.e., $j \in \mathcal{F}_i^2$, its multiplication by q_j is computed in two steps: $z_{i,j} = B_{i,j} q_j$ and $b_i = b_i + U_i z_{i,j}$. We only need to compute $z_{i,j}$ and add it to the intermediate vector z_i in the multiplication phase above. The backward transformation will compute $b_i = b_i + U_i z_{i,j}$ without extra modifications.

- For $K(X_i, X_j)$ approximated by $B_{i,j} V_j^T$, i.e., $i \in \mathcal{F}_j^2$, its multiplication by q_j is computed in two steps: $y_j = V_j^T q_j$ and $b_i = b_i + B_{i,j} y_j$. Note that y_j has been computed in the forward transformation. Then, directly compute $b_i = b_i + B_{i,j} y_j$.

The overall $\mathcal{H}^2$-matvec is in Algorithm 2. This multiplication accesses all the components of the $\mathcal{H}^2$ matrix, i.e., dense inadmissible blocks $K(X_i, X_j)$, uniform basis matrices U_i and V_i, transfer matrices R_i and S_i, and intermediate matrices $B_{i,j}$, exactly once. Thus, according to the storage cost of these components analyzed in Section 2.3.1, $\mathcal{H}^2$-matvec has computation cost of scale $6^d N r_0$ which is linear in the matrix dimension.

Algorithm 2 $\mathcal{H}^2$ matrix-vector multiplication

```
1: • initialize result vector $b$ to zero.
2: • initialize intermediate vectors $z_i$ for all $i \in \mathcal{T}$ to zero.
3: for all $(i, j) \in \mathcal{D}$ do                                    ▷ Step 1: Dense multiplication
4:     $b_i = b_i + K(X_i, X_j)q_j$.
5: end for
6: for $l = L, L-1, \ldots, 1$ do                                        ▷ Step 2: Forward transformation
7:     for all node $i$ at level $k$ do
8:         if $i$ is a leaf node then
9:             $y_i = V_i^T q_i$.
10:        else
11:            $y_i = S_i^T (y_{i_1}^T, y_{i_2}^T, \ldots, y_{i_s}^T)^T$ with children $\{i_1, i_2, \ldots, i_s\}$ of node $i$.
12:        end if
13:    end for
14: end for
15: for all $(i, j) \in \mathcal{A}$ do                                  ▷ Step 3: Intermediate multiplication
16:     $z_i = z_i + B_{i,j} y_j$.
17: end for
18: for all $(i, j) \in \mathcal{A}_p$ with $j \in \mathcal{F}_i^2$ do     ▷ Partially admissible blocks
19:     $z_i = z_i + B_{i,j} q_j$.
20: end for
21: for $l = 1, 2, \ldots, L$ do                                         ▷ Step 4: Backward transformation
22:     for all non-leaf node $i$ at level $l$ do
23:         $z_{i_a} = z_{i_a} + [R_i z_i]_{i_a}$ with all children $i_a \in \{i_1, i_2, \ldots, i_s\}$ of node $i$.
24:     end for
25: end for
26: for all leaf node $i$ in $\mathcal{T}$ do
27:     $b_i = b_i + U_i z_i$.
28: end for
29: for all $(i, j) \in \mathcal{A}_p$ with $i \in \mathcal{F}_j^2$ do     ▷ Partially admissible blocks
30:     $b_i = b_i + B_{i,j} y_j$.
31: end for
```

2.6 $\mathcal{H}^2$ matrix construction

In this section, we first discuss the general $\mathcal{H}^2$ matrix construction process. Furthermore, we provide a detailed description about the construction of an $\mathcal{H}^2$ matrix with the low-rank approximations of all the blocks $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ in an interpolative decomposition (ID) form. Such an $\mathcal{H}^2$ matrix is referred to as an ID-based $\mathcal{H}^2$ matrix.

General $\mathcal{H}^2$ matrix construction In general, the construction of an $\mathcal{H}^2$ matrix consists of three steps: (1) apply Algorithm 1 to compute the uniform basis matrices U_i, V_i for leaf nodes and the transfer matrices R_i, S_i for non-leaf nodes; (2) compute intermediate matrices $B_{i,j}$ to make the corresponding low-rank approximations of admissible or partial admissible blocks as accurate as possible; (3) compute all the inadmissible blocks $K(X_i, X_j)$ with $(i,j) \in \mathcal{D}$. For each block $K(X_i, X_j)$ with $j \in \mathcal{F}_i$ or $i \in \mathcal{F}_j$, the optimal intermediate matrix $B_{i,j}$ minimizing the approximation error in the Frobenius norm can be computed as

$$B_{i,j} = \begin{cases} U_i^\dagger K(X_i, X_j)(V_j^T)^\dagger & j \in \mathcal{F}_i^1 \text{ or } i \in \mathcal{F}_j^1 \\ U_i^\dagger K(X_i, X_j) & j \in \mathcal{F}_i^2 \\ K(X_i, X_j)(V_j^T)^\dagger & i \in \mathcal{F}_j^2 \end{cases}.$$

The approximation of $K(X_i, X_j)$ defined by this $B_{i,j}$ projects all the matrix columns onto col(U_i) and/or projects all the matrix rows onto col(V_j). These optimal $B_{i,j}$ can be recursively constructed using B_{i_a,j_b} associated with possible children i_a of i and j_b of j. In the end, only $B_{i,j}$ with $(i,j) \in \mathcal{A}$ and $(i,j) \in \mathcal{A}_p$ are kept for the final $\mathcal{H}^2$ matrix representation. Details of this recursive construction are described in Appendix A.

ID-based $\mathcal{H}^2$ matrix construction For each node i with non-empty $\mathcal{F}_i$, we consider compressing $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ into ID form as

$$K(X_i, \cup_{k\in\mathcal{F}_i} X_k) \approx U_i K(X_i^{\text{row-id}}, \cup_{k\in\mathcal{F}_i} X_k) \tag{2.19}$$

$$K(\cup_{k\in\mathcal{F}_i}X_k, X_i) \approx K(\cup_{k\in\mathcal{F}_i}X_k, X_i^{\text{col-id}})V_i^T, \tag{2.20}$$

where $X_i^{\text{row-id}}$ is a subset of X_i and thus $K(X_i^{\text{row-id}}, \cup_{k\in\mathcal{F}_i}X_k)$ contains a subset of rows in $K(X_i, \cup_{k\in\mathcal{F}_i}X_k)$; $X_i^{\text{col-id}}$ is a subset of X_i and thus $K(\cup_{k\in\mathcal{F}_i}X_k, X_i^{\text{col-id}})$ contains a subset of columns in $K(\cup_{k\in\mathcal{F}_i}X_k, X_i)$.

For each non-leaf node i with children $\{i_1, i_2, \ldots, i_s\}$, the nested approach in Algorithm 1 to computing the above two ID approximations can be written as

$$K(X_i, \cup_{k\in\mathcal{F}_i}X_k) \approx \begin{pmatrix} U_{i_1}K(X_{i_1}^{\text{row-id}}, \cup_{k\in\mathcal{F}_i}X_k) \\ \vdots \\ U_{i_s}K(X_{i_s}^{\text{row-id}}, \cup_{k\in\mathcal{F}_i}X_k) \end{pmatrix} = \begin{pmatrix} U_{i_1} & & \\ & \ddots & \\ & & U_{i_s} \end{pmatrix} K(\hat{X}_i^{\text{row-id}}, \cup_{k\in\mathcal{F}_i}X_k)$$

$$\begin{aligned} K(\cup_{k\in\mathcal{F}_i}X_k, X_i) &\approx \begin{pmatrix} K(\cup_{k\in\mathcal{F}_i}X_k, X_{i_1}^{\text{col-id}})V_{i_1}^T & \ldots & K(\cup_{k\in\mathcal{F}_i}X_k, X_{i_s}^{\text{col-id}})V_{i_s}^T \end{pmatrix} \\ &= K(\cup_{k\in\mathcal{F}_i}X_k, \hat{X}_i^{\text{col-id}}) \begin{pmatrix} V_{i_1}^T & & \\ & \ddots & \\ & & V_{i_s}^T \end{pmatrix}, \end{aligned}$$

where $\hat{X}_i^{\text{row-id}} = \cup_{i_a}X_{i_a}^{\text{row-id}}$ and $\hat{X}_i^{\text{col-id}} = \cup_{i_a}X_{i_a}^{\text{col-id}}$. As can be observed, the two matrices to be actually compressed are still kernel matrix blocks. To compute the transfer matrices R_i and S_i, these two matrices are also compressed into ID form as

$$K(\hat{X}_i^{\text{row-id}}, \cup_{k\in\mathcal{F}_i}X_k) \approx R_iK(X_i^{\text{row-id}}, \cup_{k\in\mathcal{F}_i}X_k) \tag{2.21}$$

$$K(\cup_{k\in\mathcal{F}_i}X_k, \hat{X}_i^{\text{col-id}}) \approx K(\cup_{k\in\mathcal{F}_i}X_k, X_i^{\text{col-id}})S_i^T, \tag{2.22}$$

where the computed $X_i^{\text{row-id}}$ and R_i together define the final ID approximation eq. (2.19) of $K(X_i, \cup_{k\in\mathcal{F}_i}X_k)$ with U_i represented in a nested form using R_i and similarly the computed $X_i^{\text{col-id}}$ and S_i together define the final ID approximation eq. (2.20) of $K(\cup_{k\in\mathcal{F}_i}X_k, X_i)$.

The key benefit of using ID approximation to construct an $\mathcal{H}^2$ matrix is that the intermediate matrices $B_{i,j}$ can be computed far more efficiently than the general recursive construction illustrated in Appendix A. For an admissible block $K(X_i, X_j)$ with $j \in \mathcal{F}_i^1$, the columns in the ID approximation $K(X_i, \cup_{k\in\mathcal{F}_i} X_k) \approx U_i K(X_i^{\text{row-id}}, \cup_{k\in\mathcal{F}_i} X_k)$ that correspond to X_j give the approximation $K(X_i, X_j) \approx U_i K(X_i^{\text{row-id}}, X_j)$. Similarly, the ID approximation $K(\cup_{k\in\mathcal{F}_j} X_k, X_j) \approx K(\cup_{k\in\mathcal{F}_j} X_k, X_j^{\text{col-id}})V_j^T$ gives $K(X_i^{\text{row-id}}, X_j) \approx K(X_i^{\text{row-id}}, X_j^{\text{col-id}})V_j^T$ based on the fact that $X_i^{\text{row-id}} \subset X_i \subset \cup_{k\in\mathcal{F}_j} X_k$. Combining these two approximations leads to

$$K(X_i, X_j) \approx U_i K(X_i^{\text{row-id}}, X_j^{\text{col-id}})V_j^T. \tag{2.23}$$

The associated intermediate matrix $B_{i,j}$ can thus be set to $K(X_i^{\text{row-id}}, X_j^{\text{col-id}})$. Similar discussions apply to partially admissible blocks and $B_{i,j}$ can be generally computed as

$$B_{i,j} = \begin{cases} K(X_i^{\text{row-id}}, X_j^{\text{col-id}}) & (i,j) \in \mathcal{A} \\ K(X_i^{\text{row-id}}, X_j) & (i,j) \in \mathcal{A}_p \text{ and } j \in \mathcal{F}_i^2 \\ K(X_i, X_j^{\text{col-id}}) & (i,j) \in \mathcal{A}_p \text{ and } i \in \mathcal{F}_j^2 \end{cases}.$$

As can be observed, $\mathcal{B}_{i,j}$ above can be directly computed using X_i, $X_i^{\text{row-id}}$, X_j, and $X_j^{\text{col-id}}$. From the discussion in Section 2.3.3, it can be noted that the storage cost of these intermediate matrices dominates the total storage cost of an $\mathcal{H}^2$ matrix. As a result, another key benefit of using ID approximation to construct an $\mathcal{H}^2$ matrix is that the above definition of $\mathcal{B}_{i,j}$ gives the option to not store $\mathcal{B}_{i,j}$ but dynamically compute them when needed as a trade-off between computation and storage.

In our own $\mathcal{H}^2$ matrix library currently under development with collaborators, dynamically computing $\mathcal{B}_{i,j}$ can dramatically reduce the storage cost of $\mathcal{H}^2$ matrices. Experimentally, it also turns out that, in the $\mathcal{H}^2$ matrix-vector multiplication, dynamically computing $\mathcal{B}_{i,j}$ can actually lead to similar multiplication runtime as precomputing and storing $\mathcal{B}_{i,j}$. This is mainly due to that the communication cost to load $\mathcal{B}_{i,j}$ from memory can be as expensive as dynamically computing $\mathcal{B}_{i,j}$ in CPU.

The overall construction of an ID-based $\mathcal{H}^2$ matrix is summarized in Algorithm 3.

Algorithm 3 Construct an ID-based $\mathcal{H}^2$ matrix representation of $K(X, X)$

```
 1: • construct a hierarchical partitioning of $X$ which gives a $L$-level partition tree $\mathcal{T}$.
 2: for $l = L, L-1, \dots, 1$ do
 3:     for all nodes $i$ in level$(l)$ do
 4:         if $i$ is a leaf node then
 5:             • compute $U_i$ and $X_i^{\text{row-id}}$ from an ID approximation of $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$.
 6:             • compute $V_i$ and $X_i^{\text{col-id}}$ from an ID approximation of $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$.
 7:         else if $i$ is a non-leaf node with children $\{i_1, i_2, \dots, i_s\}$ then
 8:             • construct $\hat{X}_i^{\text{row-id}} = \cup_{i_a} X_{i_a}^{\text{row-id}}$ and $\hat{X}_i^{\text{col-id}} = \cup_{i_a} X_{i_a}^{\text{col-id}}$.
 9:             • compute $R_i$ and $X_i^{\text{row-id}}$ from an ID approximation of $K(\hat{X}_i^{\text{row-id}}, \cup_{k\in\mathcal{F}_i} X_k)$.
10:             • compute $S_i$ and $X_i^{\text{col-id}}$ from an ID approximation of $K(\cup_{k\in\mathcal{F}_i} X_k, \hat{X}_i^{\text{col-id}})$.
11:         end if
12:     end for
13: end for
14: • (optional, can be dynamically computed) compute inadmissible blocks $K(X_i, X_j)$
    for all $(i, j) \in \mathcal{D}$ and intermediate matrices $B_{i,j}$ for all $(i, j) \in \mathcal{A}$ and all $(i, j) \in \mathcal{A}_p$.
```

CHAPTER 3
INTERPOLATIVE DECOMPOSITION VIA PROXY POINTS

For kernel matrices, the main bottleneck of applying $\mathcal{H}^2$ matrix techniques is the expensive computation required to represent a matrix in $\mathcal{H}^2$ format, which in turn is dominated by the low-rank approximation of all the numerically low-rank blocks identified by the admissibility condition. Compressing these blocks using purely algebraic methods, e.g., SVD and the pivoted QR decomposition, usually leads to prohibitive quadratic construction cost. These approximated blocks are all in the form $K(X_0, Y_0)$ with $X_0 \times Y_0$ in an admissible pair of domains $\mathcal{X} \times \mathcal{Y}$ as illustrated in Figure 3.1. As a result, efficient compression methods for such a block $K(X_0, Y_0)$ are critical for the construction and application of $\mathcal{H}^2$ matrix representations.

For kernel functions from potential theory, such as the Laplace kernel and Stokes kernel, Martinsson and Rokhlin [10] introduced *the proxy surface method* to efficiently compress $K(X_0, Y_0)$ in interpolative decomposition (ID) form,

$$K(X_0, Y_0) \approx U K(X_{\text{id}}, Y_0), \tag{3.1}$$

where X_{id} is a subset of points in X_0 and $K(X_{\text{id}}, Y_0)$ contains a subset of rows in $K(X_0, Y_0)$. This technique is applied in a class of fast direct solvers [10, 12, 13, 14] for kernel matrices based on HSS format and dramatically reduces the HSS construction cost. Methods closely related to the proxy surface method also exist which differ in their selection of so-called proxy points [15, 16]. Together, all these methods, including the proxy surface method, have a general form that we refer to as *the proxy point method*. As to be discussed later, while the proxy point method is more efficient than computing the ID using SRRQR alone, several important problems concerning the method still remain unsolved which limits its present application to specific kernel functions. In this chapter, we address these problems and generalize the application of the method to the construction of different types of rank-structured matrices with kernel functions that are more general than those usually used.

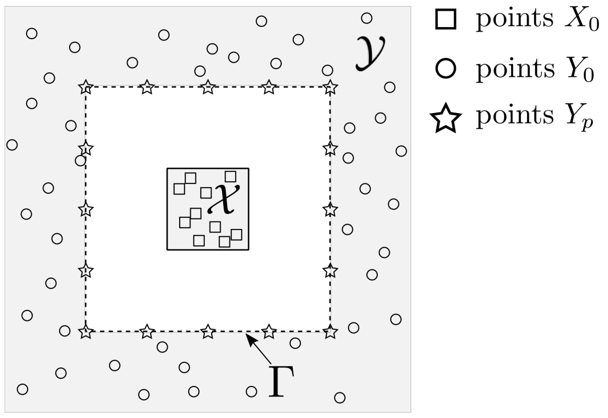
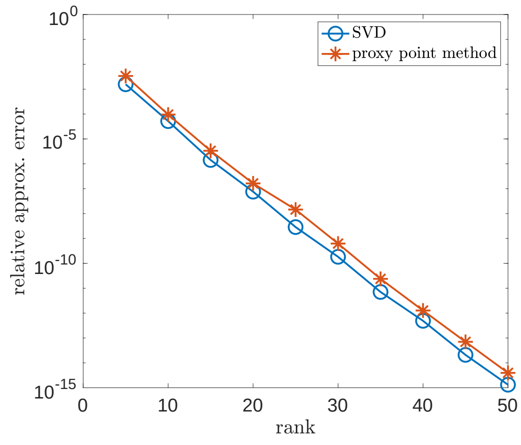


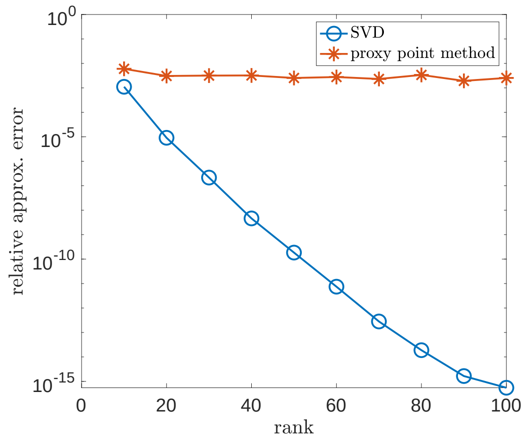
Figure 3.1: Illustration of the proxy surface method. The proxy points Y_p are uniformly selected on the interior boundary Γ of $\mathcal{Y}$. The ID approximation of $K(X_0, Y_0)$ is efficiently constructed via an algebraic ID approximation of $K(X_0, Y_p)$.

Figure 3.1 gives a simple illustration of the proxy surface method for $K(X_0, Y_0)$ with the Laplace kernel $K(x, y)$. The method first uniformly selects a small set of *proxy points* Y_p on the interior boundary of $\mathcal{Y}$. An ID approximation of $K(X_0, Y_p)$ is then computed algebraically as $K(X_0, Y_p) \approx UK(X_{\text{id}}, Y_p)$ using SRRQR. The obtained U and X_{id} are directly used in eq. (3.1) to construct the ID approximation of $K(X_0, Y_0)$. Usually, $K(X_0, Y_p)$ is a much smaller matrix than $K(X_0, Y_0)$. The proxy surface method can thus be much faster than computing the ID approximation of $K(X_0, Y_0)$ using SRRQR alone. The proxy point method generalizes the proxy surface method, where the proxy points Y_p can be selected in the whole domain $\mathcal{Y}$ and the domains $\mathcal{X}$ and $\mathcal{Y}$ can also be chosen adaptively for different kernel functions and different rank-structured matrix formats.

The proxy point method, however, has limited application due to several unsolved problems. First, the effectiveness of the method is, so far, only supported by numerical results. A rigorous explanation of how the proxy point method works and under what conditions is still missing. Second, the proper selection of proxy points Y_p is critical to controlling the approximation accuracy of the method, and such a selection varies for different kernel functions. However, Y_p is only heuristically selected in practice. Figure 3.2 shows a numerical example where the proxy point method with Y_p selected as in Figure 3.1 works well for the Laplace kernel but poorly for a Gaussian kernel. A more effective but still heuristic selection of Y_p for Gaussian kernels is suggested in Ref. [15]. These unsolved problems limit the present application of the proxy point method to the construction of HSS and $\mathcal{H}^2$ matrices with specific kernel functions. To the best of our knowledge, only two specific instances of the proxy point method have been heuristically used in practice: the proxy surface method [10, 12, 13, 14] and a variant [15, 16] of the proxy surface method.



(a) Laplace kernel $K(x, y) = \log(|x - y|)$



(b) Gaussian kernel $K(x, y) = e^{-0.1|x-y|^2}$

Figure 3.2: Relative approximation error of the proxy point method for $K(X_0, Y_0)$ with different approximation ranks and with two kernel functions. Let $\mathcal{X} = [-1, 1]^2$, $\mathcal{Y} = [-5, 5]^2 \backslash [-3, 3]^2$, and $\Gamma = \partial([-3, 3]^2)$ as shown in Figure 3.1. We randomly selected 400 points in $\mathcal{X}$ for X_0 and 6400 points in $\mathcal{Y}$ for Y_0, and uniformly selected 300 points on Γ for Y_p.

In this chapter, we present a general form of the proxy point method for a general kernel function $K(x, y)$ and a pair of compact domains $\mathcal{X}$ and $\mathcal{Y}$ satisfying the conditions that $K(x, y)$ in $\mathcal{X} \times \mathcal{Y}$ is smooth and can be represented by an accurate, low-degree degenerate approximation. In this general problem setting, we provide a rigorous error analysis for the proxy point method. Using this error analysis, we further develop a systematic and adaptive scheme to select proxy points that can guarantee the method to be effective, paving the way for a wider application of the proxy point method in the construction of different types of rank-structured matrices. The rest of the chapter is organized as follows.

- Section 3.1 presents a general form of the proxy point method and describes a fundamental mathematical interpretation of how the method works.
- Section 3.2 provides a rigorous error analysis for the proxy point method under a general setting, proving the effectiveness of the method.
- Section 3.3 gives guidelines for selecting a proper set of proxy points for a given kernel function $K(x, y)$ and a pair of domains $\mathcal{X} \times \mathcal{Y}$, and also proposes a systematic and adaptive proxy point selection scheme that can guarantee the proxy point method is effective with the selected proxy points.
- Section 3.4 demonstrates numerical experiments that illustrate the effectiveness of the proposed proxy point selection scheme in Section 3.3 and the performance of the proxy point method in $\mathcal{H}^2$ matrix construction.

3.1 Mathematical interpretation

Recall that the proxy surface method and its variant only work for specific kernel functions with a pair of domains as exemplified in Figure 3.1. In this chapter, we focus on the proxy point method as presented in Algorithm 4 which has a general form that contains both the proxy surface method and its variant as special cases. It turns out that this general proxy point method can compute a good ID approximation of $K(X_0, Y_0)$ for any kernel function $K(x, y)$ with two compact domains $\mathcal{X}$ and $\mathcal{Y}$ satisfying the conditions that $K(x, y)$ in $\mathcal{X} \times \mathcal{Y}$ is smooth and can be represented by an accurate, low-degree degenerate approximation (i.e., $\mathcal{X} \times \mathcal{Y}$ is admissible for $K(x, y)$). The key is to select a set of proxy points Y_p in $\mathcal{Y}$ properly and adaptively according to $K(x, y)$ and $\mathcal{X} \times \mathcal{Y}$. To justify the effectiveness of the proxy point method under the above general setting, this section first provides a fundamental interpretation of how the proxy point method works.

Algorithm 4 Proxy point method

Input: $K(x, y)$, $\mathcal{X}$, $\mathcal{Y}$, $X_0 \subset \mathcal{X}$, $Y_0 \subset \mathcal{Y}$.
Output: ID approximation of $K(X_0, Y_0)$,

$$K(X_0, Y_0) \approx UK(X_{\text{id}}, Y_0), \quad X_{\text{id}} \subset X_0. \tag{3.2}$$

Step 1: Select a set of proxy points Y_p in $\mathcal{Y}$. Points in Y_p are independent of X_0 and Y_0.
Step 2: Compute U and X_{id} for eq. (3.2) by computing an algebraic ID approximation of $K(X_0, Y_p)$ using SRRQR with a given error threshold (or a given rank),

$$K(X_0, Y_p) \approx UK(X_{\text{id}}, Y_p). \tag{3.3}$$

3.1.1 Algorithm interpretation

In the proxy point method Algorithm 4, the ID approximations eq. (3.2) and eq. (3.3) can be viewed row-by-row as

$$K(X_0, Y_0) \approx UK(X_{\text{id}}, Y_0) \Longleftrightarrow K(x_i, Y_0) \approx u_i^T K(X_{\text{id}}, Y_0), \quad x_i \in X_0, \tag{3.4}$$

$$K(X_0, Y_p) \approx UK(X_{\text{id}}, Y_p) \Longleftrightarrow K(x_i, Y_p) \approx u_i^T K(X_{\text{id}}, Y_p), \quad x_i \in X_0, \tag{3.5}$$

where u_i^T denotes the ith row of U. For each $x_i \in X_0$, the above two row approximations are connected by the function approximation in domain $\mathcal{Y}$,

$$K(x_i, y) \approx u_i^T K(X_{\text{id}}, y), \quad y \in \mathcal{Y}. \tag{3.6}$$

Evaluating this function approximation at Y_0 and Y_p gives the row approximations eq. (3.4) and eq. (3.5), respectively. Furthermore, it always holds that

$$\left\| K(x_i, Y_0) - u_i^T K(X_{\text{id}}, Y_0) \right\|_2 / \sqrt{|Y_0|} \leqslant \max_{y \in \mathcal{Y}} \left| K(x_i, y) - u_i^T K(X_{\text{id}}, y) \right|,$$

with any ID components U and X_{id}.

From this viewpoint, a good ID approximation $UK(X_{\text{id}}, Y_0)$ to $K(X_0, Y_0)$ can be found by seeking U and X_{id} such that each function approximation defined in eq. (3.6) has small error in the whole domain $\mathcal{Y}$. To make the problem tractable, instead of considering the approximation eq. (3.6) at every $y \in \mathcal{Y}$, the proxy point method considers it at a finite set of proxy points $Y_p \subset \mathcal{Y}$. With $\mathcal{X} \times \mathcal{Y}$ being admissible for $K(x, y)$, it turns out that there exists a proper selection of Y_p such that the approximation eq. (3.6) to each $K(x_i, y)$ with any ID components U and X_{id} has maximum absolute error bounded by a small multiple of its root-mean-square error at Y_p, i.e.,

$$\max_{y \in \mathcal{Y}} \left| K(x_i, y) - u_i^T K(X_{\text{id}}, y) \right| \leqslant O(1) \left(\left\| K(x_i, Y_p) - u_i^T K(X_{\text{id}}, Y_p) \right\|_2 / \sqrt{|Y_p|} \right). \tag{3.7}$$

Such a selection of Y_p will be discussed in Section 3.3.

Assuming that we have a set of proxy points Y_p satisfying eq. (3.7), the proxy point method computes U and X_{id} from eq. (3.3) using SRRQR, which has the approximation error $\left\| K(x_i, Y_p) - u_i^T K(X_{\text{id}}, Y_p) \right\|_2$ for each row bounded by a specified error threshold. According to eq. (3.7), the resulting U and X_{id} define a good function approximation eq. (3.6) at points in Y_p and thus in $\mathcal{Y}$. Thus, the error of the ID approximation $UK(X_{\text{id}}, Y_0)$ to $K(X_0, Y_0)$ is controlled by the error of the ID approximation $UK(X_{\text{id}}, Y_p)$ to $K(X_0, Y_p)$ as

$$\left\|K(x_i, Y_0) - u_i^T K(X_{\text{id}}, Y_0)\right\|_2 / \sqrt{|Y_0|} \leqslant O(1) \left(\left\|K(x_i, Y_p) - u_i^T K(X_{\text{id}}, Y_p)\right\|_2 / \sqrt{|Y_p|}\right).$$

As to be shown in Section 3.3, the number of proxy points Y_p needed to satisfy eq. (3.7) can be as small as the degree of a putative degenerate approximation of $K(x, y)$ in $\mathcal{X} \times \mathcal{Y}$ with a given accuracy. As a result, $K(X_0, Y_p)$ is usually a much smaller matrix than $K(X_0, Y_0)$ and the proxy point method in Algorithm 4 can be much faster than the direct ID approximation of $K(X_0, Y_0)$ using SRRQR alone.

3.1.2 Connection with pseudoskeleton approximation

The proxy point method is closely related to the degenerate approximation of $K(x, y)$ in $\mathcal{X} \times \mathcal{Y}$ in a pseudoskeleton form [28],

$$K(x, y) \approx K(x, Y_{pt}) K(X_{pt}, Y_{pt})^{\dagger} K(X_{pt}, y), \quad x \in \mathcal{X},\ y \in \mathcal{Y}, \tag{3.8}$$

where X_{pt} and Y_{pt} are "pivot" points selected in $\mathcal{X}$ and $\mathcal{Y}$, respectively. Pseudoskeleton approximations with different choices of X_{pt} and Y_{pt} are used in many fast matrix-vector multiplication algorithms such as the kernel independent FMM [21, 25], the fast directional multilevel algorithm [19], and the butterfly method [23]. Compression techniques such as adaptive cross approximation [29] and skeletonized interpolation [30] are also in this form. Specifically, it turns out that the proxy point method is equivalent to the combination of a pseudoskeleton approximation of $K(x, y)$ in $\mathcal{X} \times \mathcal{Y}$ and an algebraic recompression. For two sets of points, X_0 in $\mathcal{X}$ and Y_0 in $\mathcal{Y}$, a pseudoskeleton approximation eq. (3.8) defines a low-rank approximation of $K(X_0, Y_0)$ as

$$K(X_0, Y_0) \approx K(X_0, Y_{pt}) K(X_{pt}, Y_{pt})^{\dagger} K(X_{pt}, Y_0). \tag{3.9}$$

Consider a recompression of the above low-rank approximation via computing an ID approximation $UK(X_{\text{id}}, Y_{pt})$ to the factor $K(X_0, Y_{pt})$ using SRRQR. This computation is analogous to the computation of eq. (3.3) in the proxy point method. With this ID approximation, the approximation eq. (3.9) can then be written as

$$\begin{aligned} K(X_0, Y_0) &\approx UK(X_{\text{id}}, Y_{pt}) K(X_{pt}, Y_{pt})^{\dagger} K(X_{pt}, Y_0) \\ &\approx UK(X_{\text{id}}, Y_0), \end{aligned}$$

where the second approximation is obtained by substituting X_{id} and Y_0 into eq. (3.8). This final ID approximation $UK(X_{\text{id}}, Y_0)$ is exactly the result eq. (3.2) computed by the proxy point method if Y_p is selected as Y_{pt}.

The main difference between the proxy point method and the above recompressed pseudoskeleton approximation is that there is no need for X_{pt} and $K(X_{pt}, Y_{pt})^{\dagger}$ in the proxy point method. The matrix $K(X_{pt}, Y_{pt})$ is usually close to singular and calculation of its pseudoinverse can be numerically unstable. In practice, Ref. [21] applies a Tikhonov regularization to compute $K(X_{pt}, Y_{pt})^{\dagger}$ and Ref. [30] applies a backward stable algorithm to compute $K(X_{pt}, Y_{pt})^{\dagger} K(X_{pt}, y)$ for a given point y.

3.1.3 Examples of proxy points

In the proxy point method, the key component is a set of proxy points Y_p that satisfies eq. (3.7). Clearly, the selection of proxy points in $\mathcal{Y}$ should depend on $K(x, y)$ and $\mathcal{X} \times \mathcal{Y}$. Based on the above connection between the proxy point method and the pseudoskeleton approximation, it is natural to use pivot points Y_{pt} from existing pseudoskeleton approximation methods as the corresponding proxy points Y_p. Figure 3.3 shows three examples of proxy points borrowed from existing pseudoskeleton approximation methods. The effectiveness of some of these proxy point selections can be rigorously justified by the error analysis in Section 3.2.

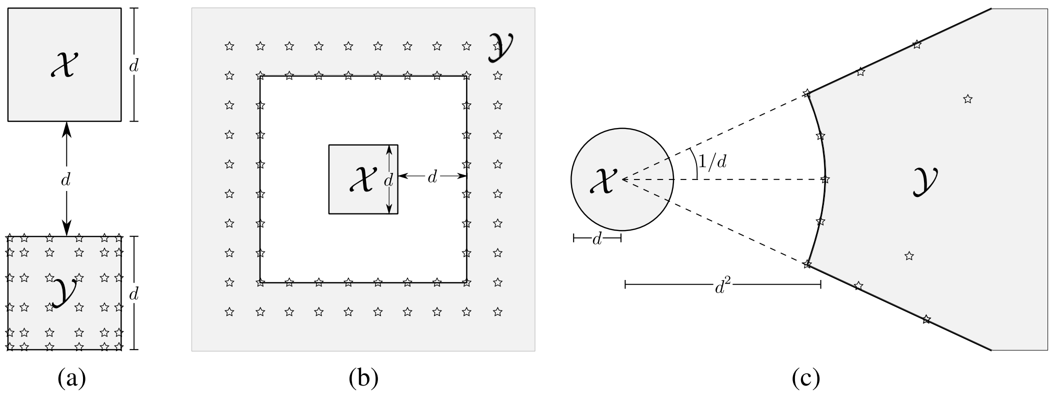


Figure 3.3: Examples of proxy points (marked as stars) borrowed from existing pseudoskeleton approximation methods for different kernel functions and corresponding admissible domain pairs: (a) For smooth kernels that can be well approximated by polynomials in $\mathcal{Y}$, proxy points are selected as a tensor grid of Chebyshev points in $\mathcal{Y}$ based on Ref. [30]. (b) For smooth radial basis functions, proxy points are selected as multiple layers of uniform grid points around the interior boundary of $\mathcal{Y}$ based on Ref. [25]. (c) For oscillatory kernels, proxy points are selected via the pivoted QR decomposition of a kernel matrix defined by points densely selected in $\mathcal{X}$ and $\mathcal{Y}$ based on Ref. [19] (cf. Figure 3 in [31]).

3.2 Error analysis

In this section, we present a rigorous error analysis for the proxy point method, which proves the effectiveness of the method and also provides theoretical guidance for selecting the proxy points as to be discussed in Section 3.3. Recall that we always assume that $\mathcal{X}$ and $\mathcal{Y}$ are compact and $K(x,y)$ is smooth in $\mathcal{X} \times \mathcal{Y}$ in the following discussion.

Using the computed U and X_{id} from Algorithm 4, denote the error of the function approximation eq. (3.6) to each $K(x_i, y)$ as

$$e_i(y) = K(x_i, y) - u_i^T K(X_{\text{id}}, y), \quad x_i \in X_0,\ y \in \mathcal{Y}. \tag{3.10}$$

With this notation, the ith row of the error matrix $K(X_0, Y_0) - UK(X_{\text{id}}, Y_0)$ for the ID approximation eq. (3.2) is exactly $e_i(Y_0)$. Similarly, the ith row of the error matrix $K(X_0, Y_p) - UK(X_{\text{id}}, Y_p)$ for the ID approximation eq. (3.3) is $e_i(Y_p)$.

For an arbitrary set of points Y_0 in $\mathcal{Y}$, the best upper bound for $\|e_i(Y_0)\|_2$ is

$$\|e_i(Y_0)\|_2 \leqslant \sqrt{|Y_0|} \max_{y \in \mathcal{Y}} |e_i(y)| = \sqrt{|Y_0|} \|e_i(y)\|_\infty, \tag{3.11}$$

where equality holds when $|e_i(y)|$ reaches the same maximum in $\mathcal{Y}$ for all points in Y_0. On the other hand, $\|e_i(Y_p)\|_2$ is bounded by the error threshold specified for the ID approximation eq. (3.3) of $K(X_0, Y_p)$. Thus, the following error analysis for the proxy point method seeks an upper bound for $\|e_i(y)\|_\infty$ in terms of $\|e_i(Y_p)\|_2$, i.e., an inequality in the form of eq. (3.7).

For analysis purposes, consider a generic r-term ε-expansion of $K(x, y)$ in $\mathcal{X} \times \mathcal{Y}$,

$$K(x, y) = \sum_{j=1}^{r} \psi_j(x)\phi_j(y) + R_r(x, y), \quad x \in \mathcal{X},\ y \in \mathcal{Y}, \tag{3.12}$$

where the remainder $|R_r(x, y)|$ is bounded by ε. Assume that functions in $\{\phi_j(y)\}_{j=1}^r$ are linearly independent. Such an expansion eq. (3.12) shows that, for any $x \in \mathcal{X}$, the function $K(x, y)$ in the single variable $y \in \mathcal{Y}$ is close to the r-dimensional function space spanned by $\{\phi_j(y)\}_{j=1}^r$ with distance less than ε, i.e.,

$$\text{dist}\left(K(x, y), \text{span}(\{\phi_j(y)\}_{j=1}^r)\right) = \inf_{f \in \text{span}(\{\phi_j(y)\}_{j=1}^r)} \|K(x, y) - f(y)\|_\infty \leqslant \varepsilon.$$

Being a linear combination of $K(x_i, y)$ and $\{K(x_j, y)\}_{x_j \in X_{\text{id}}}$, each error function $e_i(y)$ is also close to $\text{span}(\{\phi_j(y)\}_{j=1}^r)$ with small distance,

$$\begin{aligned} \text{dist}\left(e_i(y), \text{span}(\{\phi_j(y)\}_{j=1}^r)\right) &\leqslant (1 + \|u_i\|_1) \max_{x \in X_0} \text{dist}\left(K(x, y), \text{span}(\{\phi_j(y)\}_{j=1}^r)\right) \\ &\leqslant (1 + C_{\text{qr}}|Y_p|)\varepsilon. \end{aligned} \tag{3.13}$$

The first inequality above is based on the triangular inequality. The second inequality is based on the facts that entries of U computed in eq. (3.3) are bounded by the prespecified parameter C_{qr} for SRRQR and that $|X_{\text{id}}|$ computed in eq. (3.3) is no greater than $|Y_p|$.

To estimate $\|e_i(y)\|_\infty$ in terms of $\|e_i(Y_p)\|_2$, the idea is to first approximate $e_i(y)$ in the function space span($\{\phi_j(y)\}_{j=1}^r$) based on its values at points in Y_p and then estimate $\|e_i(y)\|_\infty$ in this finite-dimensional space. To begin with, consider finding an approximation of $e_i(y)$ in span($\{\phi_j(y)\}_{j=1}^r$) as

$$e_i(y) \approx c_1\phi_1(y) + \ldots + c_r\phi_r(y) = c^T\Phi(y), \tag{3.14}$$

where $c = (c_1, c_2, \ldots, c_r)^T$ and $\Phi(y) = (\phi_1(y), \phi_2(y), \ldots, \phi_r(y))^T$. The coefficient vector c is selected to minimize the function approximation error at Y_p, i.e.,

$$c = \arg_{v\in\mathbb{R}^r} \min \|e_i(Y_p) - v^T\Phi(Y_p)\|_2,$$

where $\Phi(Y_p) \in \mathbb{R}^{r\times|Y_p|}$ denotes the matrix of column vectors $\Phi(y_j)$ for all y_j in Y_p. For the uniqueness of c and thus the uniqueness of the approximant $c^T\Phi(y)$, a necessary condition that Y_p needs to satisfy is

$$\text{rank}(\Phi(Y_p)) = r, \tag{3.15}$$

meaning that row vectors of $\Phi(Y_p)$ are linearly independent. Under this condition, c is solved as $c^T = e_i(Y_p)\Phi(Y_p)^\dagger$ and $e_i(y)$ is approximated as

$$e_i(y) \approx e_i(Y_p)\Phi(Y_p)^\dagger\Phi(y).$$

Let $S_{Y_p}(y) = (s_1(y), s_2(y), \ldots, s_{|Y_p|}(y))^T$ denote the vector $\Phi(Y_p)^\dagger\Phi(y)$ of dimension $|Y_p|$. The above approximation process of $e_i(y)$ can be generalized as a linear operator $\mathcal{L} : C^\infty(\mathcal{Y}) \to \text{span}(\{\phi_j(y)\}_{j=1}^r)$ in the function space $C^\infty(\mathcal{Y})$ that contains all the smooth functions in $\mathcal{Y}$,

$$\mathcal{L}f(y) = \sum_{y_j \in Y_p} f(y_j)s_j(y) = f(Y_p)S_{Y_p}(y), \quad f \in C^\infty(\mathcal{Y}). \tag{3.16}$$

This operator defines a generalized interpolation process such that $\mathcal{L}f(y)$ is the unique function in span$(\{\phi_j(y)\}_{j=1}^r)$ whose values at points in Y_p are equal to the projection of $f(Y_p)$ onto the vector space span$(\{\phi_j(Y_p)\}_{j=1}^r)$, i.e., $\mathcal{L}f(Y_p) = f(Y_p)\Phi(Y_p)^\dagger\Phi(Y_p)$. Also, it holds that $\mathcal{L}f(y) = f(y)$ for any $f(y)$ in span$(\{\phi_j(y)\}_{j=1}^r)$ based on eq. (3.15).

We then estimate $\|e_i(y)\|_\infty$ based on its approximation $\|\mathcal{L}e_i(y)\|_\infty$,

$$\begin{aligned}
\|e_i(y)\|_\infty &\leqslant \|e_i(y) - \mathcal{L}e_i(y)\|_\infty + \|\mathcal{L}e_i(y)\|_\infty \\
&\leqslant \min_{f\in \text{span}(\{\phi_j(y)\}_{j=1}^r)} (\|e_i(y) - f(y)\|_\infty + \|f(y) - \mathcal{L}e_i(y)\|_\infty) + \|\mathcal{L}e_i(y)\|_\infty \\
&= \min_{f\in \text{span}(\{\phi_j(y)\}_{j=1}^r)} (\|e_i(y) - f(y)\|_\infty + \|\mathcal{L}f(y) - \mathcal{L}e_i(y)\|_\infty) + \|\mathcal{L}e_i(y)\|_\infty \\
&\leqslant \min_{f\in \text{span}(\{\phi_j(y)\}_{j=1}^r)} (\|e_i(y) - f(y)\|_\infty + \|\mathcal{L}\|_\infty\|f(y) - e_i(y)\|_\infty) + \|\mathcal{L}e_i(y)\|_\infty \\
&= (1 + \|\mathcal{L}\|_\infty)\text{dist}(e_i(y), \text{span}(\{\phi_j(y)\}_{j=1}^r)) + \|\mathcal{L}e_i(y)\|_\infty,
\end{aligned} \tag{3.17}$$

where the operator norm $\|\mathcal{L}\|_\infty$ is defined as

$$\|\mathcal{L}\|_\infty = \max_{f\in C^\infty(\mathcal{Y}), f\neq 0} \frac{\|\mathcal{L}f\|_\infty}{\|f\|_\infty} = \max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_1. \tag{3.18}$$

The second representation of $\|\mathcal{L}\|_\infty$ above is based on eq. (3.16) and Hölder's inequality

$$|\mathcal{L}f(y)| \leqslant \|f(Y_p)\|_\infty \|S_{Y_p}(y)\|_1 \leqslant \|f\|_\infty \|S_{Y_p}(y)\|_1, \quad y \in \mathcal{Y},$$

where equality can hold true for some $f \in C^\infty(\mathcal{Y})$ and thus the second equality in eq. (3.18) holds true. Also, $\mathcal{L}e_i(y) = e_i(Y_p)S_{Y_p}(y)$ can be bounded as

$$\|\mathcal{L}e_i(y)\|_\infty \leqslant \|e_i(Y_p)\|_2 \max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2. \tag{3.19}$$

Lastly, substituting eqs. (3.13), (3.18) and (3.19) into eq. (3.17), $\|e_i(y)\|_\infty$ is bounded as

$$\|e_i(y)\|_\infty \leqslant (1 + \max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_1)(1 + C_{\text{qr}}|Y_p|)\varepsilon + \|e_i(Y_p)\|_2 \max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2. \tag{3.20}$$

We note that the above estimation of $\|e_i(y)\|_\infty$ in terms of $\|e_i(Y_p)\|_2$ works for any r-term ε-expansion eq. (3.12) of $K(x, y)$ and for any Y_p satisfying the condition eq. (3.15). Noting that the upper bound in eq. (3.20) does not rely on $\{\psi_j(x)\}_{j=1}^r$ in the expansion eq. (3.12), eq. (3.20) can thus be further sharpened by fixing $\{\phi_j(x)\}_{j=1}^r$ and varying $\{\psi_j(x)\}_{j=1}^r$ to reduce ε that appears in the upper bound. Specifically, the minimal accuracy ε_* of an expansion eq. (3.12) using a fixed set of functions $\{\phi_j(y)\}_{j=1}^r$ can be defined as

$$\varepsilon_* = \sup_{x\in\mathcal{X}} \inf_{f\in\text{span}(\{\phi_j(y)\}_{j=1}^r)} \|K(x, y) - f(y)\|_\infty, \tag{3.21}$$

which describes the maximum distance between all the functions in $\{K(x, y)\}_{x\in\mathcal{X}}$ and the function space span$(\{\phi_j(y)\}_{j=1}^r)$.

Combining the upper bound eq. (3.20), the minimal accuracy ε_* in eq. (3.21), and the inequality $\|e_i(Y_0)\|_2/\sqrt{|Y_0|} \leqslant \|e_i(y)\|_\infty$ from eq. (3.11), the error bound for the proxy point method can be summarized as follows.

Theorem 1 (Error bound for the proxy point method)**.** *Consider two compact domains $\mathcal{X}$ and $\mathcal{Y}$ and a kernel function $K(x, y)$ being smooth in $\mathcal{X} \times \mathcal{Y}$. Given a set of linearly independent functions $\{\phi_j(y)\}_{j=1}^r$ in $\mathcal{Y}$, the minimal accuracy ε_* of all the possible degenerate approximations of $K(x, y)$ in $\mathcal{X} \times \mathcal{Y}$ using $\{\phi_j(y)\}_{j=1}^r$ is defined in eq.* (3.21)*. If the set of proxy points Y_p satisfies the condition rank$(\Phi(Y_p)) = r$, the ID approximation of $K(X_0, Y_0)$ calculated by the proxy point method with Y_p has error in the ith row $e_i(Y_0)$*

bounded as

$$\frac{\|e_i(Y_0)\|_2}{\sqrt{|Y_0|}} \leqslant \left(1 + \max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_1\right)(1 + C_{qr}|Y_p|)\,\varepsilon_* + \|e_i(Y_p)\|_2 \max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2. \quad (3.22)$$

Theorem 1 works for any set of linearly independent functions $\{\phi_j(y)\}_{j=1}^r$ which will be referred to as a set of *basis functions*. For the proxy point method, the additional assumption that $\mathcal{X} \times \mathcal{Y}$ is admissible for $K(x, y)$, i.e., there exists an accurate, low-degree degenerate approximation of $K(x, y)$ in $\mathcal{X} \times \mathcal{Y}$, guarantees that there exists a small set of basis functions $\{\phi_j(y)\}_{j=1}^r$ such that the minimal accuracy ε_* is negligible compared to $\|e_i(Y_p)\|_2$. With such $\{\phi_j(y)\}_{j=1}^r$, the first term of the upper bound eq. (3.22) is negligible if $\max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_1$ is of scale $O(1)$. Theorem 1 then shows that

$$\|e_i(Y_0)\|_2/\sqrt{|Y_0|} \leqslant O(1)\|e_i(Y_p)\|_2/\sqrt{|Y_p|},$$

if $\max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2$ and $|Y_p|$ are also of scale $O(1)$, which proves the effectiveness of the proxy point method. The remaining problem becomes how to select a small set of proxy points Y_p so that $\max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2$ and $\max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_1$ are of scale $O(1)$.

3.3 Proxy point selection

3.3.1 Guidelines for selecting proxy points

Recall that the ultimate goal of a proper selection of the proxy points Y_p is to guarantee that $\|e_i(y)\|_\infty$ is bounded by a small multiple of $\|e_i(Y_p)\|_2/\sqrt{|Y_p|}$. Assume that $\|e_i(Y_p)\|_2 \leqslant \varepsilon_{\text{id}}$ from the error threshold ε_{id} specified for the algebraic ID approximation of $K(X_0, Y_p)$. Based on Theorem 1, the guidelines for selecting proxy points are established as follows.

To simplify our discussion, the upper bound eq. (3.22) is slightly loosened as

$$\|e_i(y)\|_\infty \lesssim \left(\varepsilon_*|Y_p|^{\frac{3}{2}} + \|e_i(Y_p)\|_2\right) \max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2, \quad (3.23)$$

using $\|S_{Y_p}(y)\|_1 \leqslant \sqrt{|Y_p|}\|S_{Y_p}(y)\|_2$. This upper bound relies on two important components: basis functions $\{\phi_j(y)\}_{j=1}^r$ and proxy points $Y_p \subset \mathcal{Y}$. Fixing a set of basis functions $\{\phi_j(y)\}_{j=1}^r$, eq. (3.23) gives the following guidelines for selecting proxy points:

1. Y_p should satisfy rank$(\Phi(Y_p)) = r$ as required by Theorem 1.

2. Y_p should have $O(1)r$ points for the efficiency of the proxy point method.

3. Y_p should make $\max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2$ of scale $O(1)$.

To make $\|e_i(y)\|_\infty$ bounded by $O(1)\varepsilon_{\text{id}}$ via eq. (3.23), the basis functions $\{\phi_j(y)\}_{j=1}^r$ used in the above guidelines need to satisfy

$$\varepsilon_*|Y_p|^{\frac{3}{2}} \sim \varepsilon_* r^{\frac{3}{2}} \leqslant O(1)\varepsilon_{\text{id}}. \tag{3.24}$$

The number of basis functions r should also be small so that only a small number of proxy points is selected following the guidelines. In other words, we need to use a small set of basis functions $\{\phi_j(y)\}_{j=1}^r$ (small r) whose span is close to all the functions in $\{K(x,y)\}_{x\in\mathcal{X}}$ (small ε_*). As to be discussed next, such a set of basis functions can be selected heuristically based on analytic properties of the kernel function. The basis functions can also be selected in a numerical way, which is the approach used by the proposed proxy point selection scheme in this section.

3.3.2 Selection of basis functions

General expansion techniques such as Taylor expansion, interpolation, and Fourier series are commonly used to construct degenerate approximations of $K(x,y)$ in $\mathcal{X}\times\mathcal{Y}$, which correspond to approximating functions in $\{K(x,y)\}_{x\in\mathcal{X}}$ using specific basis functions, e.g., polynomials and trigonometric polynomials. If we know that $K(x,y)$ in the single variable $y \in \mathcal{Y}$ can be well approximated using a small number of polynomials or trigonometric polynomials, then these basis functions can be used in the guidelines for selecting proxy points. The number of such basis functions r can be selected by trial-and-error or by analytic estimation of the minimal accuracy ε_* in terms of r.

Another idea that does not require a priori knowledge of a degenerate approximation of $K(x, y)$ is to use a finite subset of $\{K(x, y)\}_{x\in\mathcal{X}}$ as the basis functions. Denote such a subset as $\{K(x_j, y)\}_{x_j\in X_p}$ associated with a set of points X_p in $\mathcal{X}$. Based on the assumption that $\mathcal{X}$ and $\mathcal{Y}$ are compact and $K(x, y)$ is smooth in $\mathcal{X} \times \mathcal{Y}$, it can be proved that there always exists such a finite subset that can approximate all the functions in $\{K(x, y)\}_{x\in\mathcal{X}}$ with a given accuracy. The basic idea is that we can select a finite set of points $X_p \subset \mathcal{X}$ such that, for any $x \in \mathcal{X}$, there exists some $x_* \in X_p$ whose distance to x is below an arbitrarily small threshold. Then, $K(x_*, y)$ can approximate $K(x, y)$ well in $\mathcal{Y}$ and thus so can a linear combination of $\{K(x_j, y)\}_{x_j\in X_p}$. We skip the detailed proof here. In practice, the basis functions $\{K(x_j, y)\}_{x_j\in X_p}$ with the minimal accuracy ε_* approximately below a given threshold ε_p can be numerically selected as follows.

First sample domain $\mathcal{Y}$ to obtain a set of uniformly distributed points Y_1 with high point density. Similarly, sample domain $\mathcal{X}$ to obtain a set of points X_1 which is larger than the expected size of X_p (through trial-and-error as explained below). Compute an ID approximation of $K(X_1, Y_1)$ using SRRQR and define X_p as the subset of X_1 that corresponds to the row subset of this ID approximation, i.e.,

$$K(X_1, Y_1) \approx U_1 K(X_p, Y_1). \tag{3.25}$$

The error threshold for this ID approximation is set to $\varepsilon_p\sqrt{|Y_1|}$. This selection of X_p satisfies the condition that $K(x, Y_1)$ for any $x \in X_1$ can be approximated by a linear combination of $\{K(x_j, Y_1)\}_{x_j\in X_p}$ with root-mean-square error bounded by ε_p. Since $K(x, y)$ is smooth in $\mathcal{X} \times \mathcal{Y}$ and X_1 and Y_1 have high point densities in $\mathcal{X}$ and $\mathcal{Y}$, respectively, we can expect that functions in $\{K(x, y)\}_{x\in\mathcal{X}}$ can be approximated by linear combinations of $\{K(x_j, y)\}_{x_j\in X_p}$ with error $O(\varepsilon_p)$. The above selection process can start with a relatively small set of points X_1. If the ID approximation eq. (3.25) has full rank, i.e., $X_p = X_1$, we can double the size of X_1 and repeat the selection process.

To make ε_* satisfy the condition eq. (3.24), we can conservatively set ε_p a few orders of magnitude smaller than ε_{id} while taking arithmetic rounding errors into account. This selection of basis functions is summarized by Step 1 in Algorithm 5.

3.3.3 Selection of proxy points

Assume that we have a set of basis functions $\{\phi_j(y)\}_{j=1}^r$ selected according to the above discussion. Following the guidelines for selecting proxy points, we seek exactly r proxy points such that rank$(\Phi(Y_p)) = r$ and $\max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2$ is of scale $O(1)$. If $|Y_p| = r$ and rank$(\Phi(Y_p)) = r$, it holds that $S_{Y_p}(y) = \Phi(Y_p)^{-1}\Phi(y)$ where the pseudoinverse becomes the exact inverse. Then the operator $\mathcal{L}$ defined in eq. (3.16) is exactly the interpolation operator in span$(\{\phi_j(y)\}_{j=1}^r)$ where Y_p is the set of interpolation nodes and entries of $S_{Y_p}(y)$ are the corresponding Lagrangian functions.

In numerical approximation theory, the norm of $\mathcal{L}$, $\|\mathcal{L}\|_\infty = \max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_1$, is called the *Lebesgue constant* and is used to characterize the quality of interpolation nodes Y_p. For polynomials and trigonometric polynomials in a regular box domain, selections of interpolation nodes leading to small Lebesgue constant have been well studied [32]. Since $\|S_{Y_p}(y)\|_2 \leqslant \|S_{Y_p}(y)\|_1$, these interpolation nodes can be directly used as proxy points. For example, if $\{\phi_j(y)\}_{j=1}^r$ are polynomials in a 2D box $\mathcal{Y}$ of degree up to $k-1$ for each variable ($r = k^2$), we can select Y_p as a $k \times k$ tensor grid of Chebyshev nodes in $\mathcal{Y}$ as illustrated in Figure 3.3a, which has $\max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2$ that is well bounded, i.e.,

$$\max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2 \leqslant \max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_1 \approx \left(\frac{2}{\pi}\log(k+1)\right)^2.$$

Furthermore, we can also select Y_p as a $k \times k$ tensor grid of Gauss-Lobatto nodes which gives an even smaller scaling factor, $\max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2 = 1$ (see Ref. [33]).

Here, we consider the general case with basis functions of the form $\{K(x_j, y)\}_{x_j \in X_p}$ and with irregular domain $\mathcal{Y}$, where analytic choices of interpolation nodes with small Lebesgue constant are not available. In this case, we introduce a numerical method to select r proxy points Y_p in $\mathcal{Y}$ such that $\max_{y \in \mathcal{Y}} \|S_{Y_p}(y)\|_2$ is of scale $O(1)$.

First, we sample domain $\mathcal{Y}$ to obtain a set of uniformly distributed points Y_2 with high point density. Then, we find an r-point subset Y_p of Y_2 that bounds $\|S_{Y_p}(y)\|_2 = \|\Phi(Y_p)^{-1}\Phi(y)\|_2$ at any point $y \in Y_2$. To do this, compute a numerically exact SRRQR decomposition of $\Phi(Y_2) \in \mathbb{R}^{r \times |Y_2|}$,

$$\Phi(Y_2)P = Q(R_1 \; R_2), \tag{3.26}$$

where P is a permutation matrix, Q is an orthogonal matrix, and R_1 is an $r \times r$ upper-triangular matrix. Let Y_p be the subset of points in Y_2 that corresponds to the columns of R_1 in the decomposition, i.e., $\Phi(Y_p) = QR_1$. It is the key feature of SRRQR that, with eq. (3.26), the matrix

$$\Phi(Y_p)^{-1}\Phi(Y_2) = (QR_1)^{-1}Q(R_1 \; R_2)P^T = (I_r \; R_1^{-1}R_2)P^T$$

has all its entries bounded by the prespecified parameter $C_{\text{qr}} \geqslant 1$. Thus, $\|S_{Y_p}(y)\| \leqslant C_{\text{qr}}\sqrt{r}$ for any $y \in Y_2$. Since Y_2 is uniformly distributed and has high point density in $\mathcal{Y}$, this selection of Y_p satisfies the condition that $\max_{y \in \mathcal{Y}} \|S_{Y_p}(y)\|_2 \lesssim C_{\text{qr}}\sqrt{r}$.

This selection of proxy points is summarized by Step 2 in Algorithm 5. Also, we note that this selection approach is closely related to the pseudoskeleton approximation method used in Ref. [19] and the numerical construction of a special set of interpolation nodes called *Fekete points* [34].

3.3.4 Summary of the selection scheme

Algorithm 5 summarizes the overall systematic numerical scheme above to select the basis functions and the proxy points. A heuristic "densification" step is included in Algorithm 5 in order to improve the quality of the set of selected proxy points. The motivation for this additional step is explained below.

```
**Algorithm 5** Proxy point selection scheme
Input: $K(x,y)$, $\mathcal{X}$, $\mathcal{Y}$, $\varepsilon_p$.
Output: Proxy points $Y_p$.
  Step 1: Selection of basis functions $\{\phi_j(y)\}_{j=1}^r$.
    a. Sample domains $\mathcal{X}$ and $\mathcal{Y}$ to obtain two sets of uniformly distributed points $X_1$ and $Y_1$ with high point density, respectively.
    b. Compute an ID approximation eq. (3.25) of $K(X_1, Y_1)$ using SRRQR with error threshold $\varepsilon_p\sqrt{|Y_1|}$.
    c. Set $\{\phi_j(y)\}_{j=1}^r$ as $\{K(x_j, y)\}_{x_j \in X_p}$ with $X_p$ defined by eq. (3.25).
  Step 2: Selection of proxy points $Y_p$.
    a. Sample domain $\mathcal{Y}$ to obtain a set of uniformly distributed points $Y_2$ with high point density.
    b. Compute an exact SRRQR decomposition eq. (3.26) of $\Phi(Y_2)$.
    c. Set $Y_p$ as the subset of points in $Y_2$ that corresponds to the columns of $R_1$ in eq. (3.26).
  Step 3: Heuristic densification of $Y_p$. For each point $y_j \in Y_p$, calculate the distance $d_j$ between $y_j$ and its nearest neighbor in $Y_p$. Randomly select one (or more) extra point in the ball centered at $y_j$ with radius $d_j/3$ and add it to $Y_p$.
```

In Algorithm 5, Step 1 obtains a set of basis functions $\{K(x_j, y)\}_{x_j \in X_p}$ that can approximate functions in $\{K(x,y)\}_{x\in\mathcal{X}}$ with error $O(\varepsilon_p)$. Step 2 then selects $r = |X_p|$ number of proxy points Y_p that satisfies $\max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2 \lesssim C_{\text{qr}}\sqrt{r}$. By eq. (3.23), the proxy points Y_p obtained by Step 1 and Step 2 satisfy the error bound

$$\|e_i(y)\|_\infty \lesssim \left(\varepsilon_p r^{\frac{3}{2}} + \|e_i(Y_p)\|_2\right) C_{qr}\sqrt{r}, \tag{3.27}$$

which justifies the effectiveness of the selected proxy points if $\varepsilon_p r^{\frac{3}{2}}$ is negligible or of similar scale as the error threshold ε_{id} specified for $\|e_i(Y_p)\|_2$.

However, note that ε_p is the root-mean-square error threshold for the algebraic ID approximation eq. (3.25). Due to arithmetic rounding errors, ε_p cannot be of smaller scale than machine precision, $\varepsilon_{\text{machine}}$. Thus, the error bound eq. (3.27) suggests that the approximation error of the proxy point method with Y_p selected by Step 1 and Step 2 in Algorithm 5 may stagnate around $\varepsilon_{\text{machine}} r^{\frac{3}{2}}$ if the specified error threshold ε_{id} is close to machine precision. Such error stagnation is indeed observed in some of our preliminary tests, where the smallest relative approximation error of the proxy point method can only reach around $10^{-13} \sim 10^{-10}$ with double precision (10^{-16}) arithmetic. To tackle this possible error stagnation for extremely small error threshold ε_{id}, a heuristic densification of Y_p by Step 3 in Algorithm 5 is added where extra proxy points are added. This densification step turns out to be experimentally effective as illustrated by the numerical results in the next section.

In terms of computation cost, Algorithm 5 is expensive due to the large number of sample points in Y_1 and Y_2. Reusing the proxy points selected by Algorithm 5 is critical for practical application of Algorithm 5. Luckily, in most rank-structured matrix applications, the kernel function is translationally invariant, i.e.,

$$K(x, y) = k(x - y), \quad \text{for some univariate function } k(\cdot).$$

As a result, the proxy points selected by Algorithm 5 can be reused by simple translations for different admissible pairs of domains in the construction of rank-structured matrices (see the numerical tests in Section 3.4.5).

Lastly, in practice and for the numerical experiments in the next section, the sample points X_1, Y_1, and Y_2 in Algorithm 5 are randomly and uniformly sampled from $\mathcal{X}$ and $\mathcal{Y}$ with their numbers of points heuristically decided for efficiency and simplicity. However, to rigorously justify the arguments about X_1, Y_1, and Y_2 in Sections 3.3.2 and 3.3.3, the point densities of these sets should depend on the magnitude of the variation of $K(x, y)$ in

$\mathcal{X} \times \mathcal{Y}$, about which we skip more detailed discussion. Possible approaches for reducing the sizes of these sample point sets are the "weakly admissible meshes" used in Ref. [34] and the tensor grid of Chebyshev nodes used in Ref. [30]. Another heuristic but usually effective approach, similar to the idea used in Ref. [19], is to initially apply Algorithm 5 with small X_1, Y_1, and Y_2 and recursively enlarge the three sets of points if needed.

3.4 Numerical experiments

In this section, we provide numerical tests to illustrate the effectiveness of the proxy point selection scheme (Algorithm 5) and the performance of the proxy point method (Algorithm 4) for general kernel functions and corresponding admissible domain pairs. The parameter C_{qr} for SRRQR used in the proxy point method and in the proxy point selection scheme is set to 2. In all the tests of Algorithm 5, X_1 contains 1500 points sampled in $\mathcal{X}$, Y_1 contains 10000 points sampled in $\mathcal{Y}$, Y_2 is the same set of points as Y_1, and the parameter ε_p is set to 10^{-14}. All these sets of sample points, X_1 and Y_1, are randomly and uniformly sampled in their corresponding domains.

3.4.1 Basic tests

Consider the following four different problem settings:

1. $K(x,y) = 1/\sqrt{1+|x-y|^2}$, $\mathcal{X} = [-1,1]^2$, $\mathcal{Y} = [3,5] \times [-1,1]$.

2. $K(x,y) = e^{-|x-y|^2}$, $\mathcal{X} = [-1,1]^2$, $\mathcal{Y} = [-7,7]^2 \backslash [-3,3]^2$.

3. $K(x,y) = e^{2\pi i|x-y|}$, $\mathcal{X} = B_2(0,2)$, $\mathcal{Y} = \{y \in \mathbb{R}^2 : \theta(y,l) \leqslant 1/2,\ 4 \leqslant |y| \leqslant 16\}$ with $l = (1,0)$. (Note: $B_d(0,a)$ denotes the d-dimensional ball centered at the origin with radius a, and $\theta(y,l)$ denotes the angle between y and l.)

4. $K(x,y) = e^{2\pi i|x-y|}/|x-y|$, $\mathcal{X} = B_3(0,2)$, $\mathcal{Y} = \{y \in \mathbb{R}^3 : \theta(y,l) \leqslant 1/2,\ 4 \leqslant |y| \leqslant 8\}$ with $l = (1,0,0)$.

For each problem setting above, the domains $\mathcal{X}$ and $\mathcal{Y}$ and the corresponding proxy points Y_p selected by Algorithm 5 are plotted in Figures 3.4 to 3.7.

Two sets of points X_0 and Y_0 are randomly and uniformly selected in $\mathcal{X}$ and $\mathcal{Y}$, respectively, with an average 100 points per unit of area in 2D or 50 points per unit of volume in 3D. For different approximation ranks, the relative approximation error of the proxy point method with the selected Y_p for $K(X_0, Y_0)$ is also plotted in the figure for each of the problem settings. The relative error of the intermediate ID approximation of $K(X_0, Y_p)$ in the proxy point method is plotted as well.

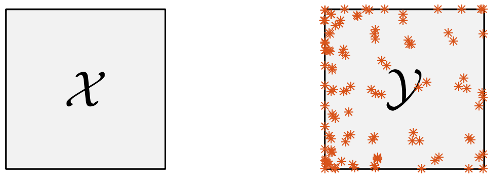
As can be observed from the results, the proxy point method with proxy points selected by Algorithm 5 has approximation error close to those of SVD and the ID approximation using SRRQR. Also, the intermediate ID approximation of $K(X_0, Y_p)$ has similar relative error as the ID approximation of $K(X_0, Y_0)$. This shows that the accuracy of the final ID approximation in the proxy point method can be controlled by controlling the accuracy of the ID approximation of $K(X_0, Y_p)$.

3.4.2 Error bound for $\|e_i(y)\|_\infty$

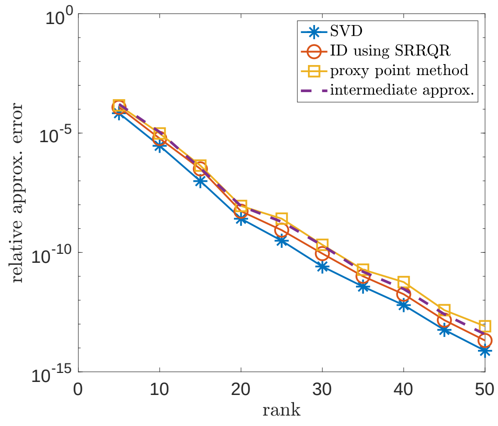
The proxy point selection scheme in Algorithm 5 and the effectiveness of the proxy point method are both based on Theorem 1 and the derived error bound eq. (3.23) for $\|e_i(y)\|_\infty$. In this subsection, we numerically study the tightness of this error bound.

Consider $K(x, y) = e^{-|x-y|^2}$, $\mathcal{X} = [-1, 1]^2$, and $\mathcal{Y} = [-7, 7]^2 \backslash [-3, 3]^2$. We randomly and uniformly select 400 points in $\mathcal{X}$ for X_0. We fix the error threshold $\varepsilon_{\text{id}} = 10^{-6}$ for the ID approximation of $K(X_0, Y_p)$ with any Y_p. In order to test the proxy point method with a given number of proxy points, we use a simple modification of Algorithm 5. Specifically, given an integer r, we select a set of r basis functions $\{K(x_j, y)\}_{x_j \in X_p}$ with X_p computed by a rank-r ID approximation of $K(X_1, Y_1)$ in Step 1 of Algorithm 5. Exactly r proxy points Y_p can be then selected by Step 2 of Algorithm 5 without densification.

Using this selection scheme, we vary r and select the corresponding r proxy points

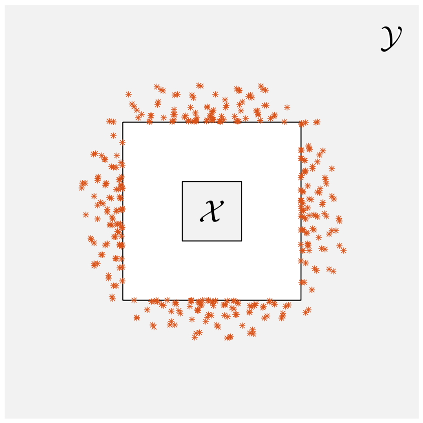


(a) proxy points

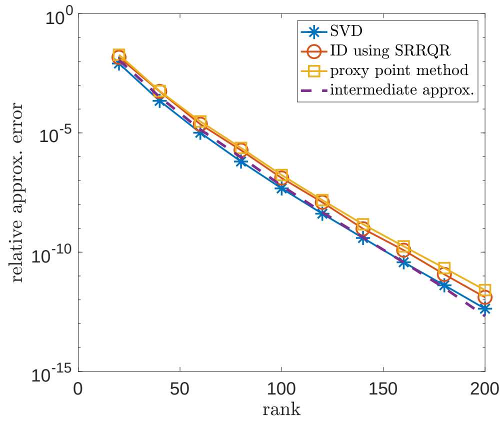


(b) relative approximation error

Figure 3.4: Basic test for problem setting 1: $K(x,y) = 1/\sqrt{1+|x-y|^2}$, $|X_0| = 400$, $|Y_0| = 400$, and $|Y_p| = 118$. In this and the following three figures, "intermediate approx." refers to the relative error of the ID approximation of $K(X_0,Y_p)$ in the proxy point method.



(a) proxy points

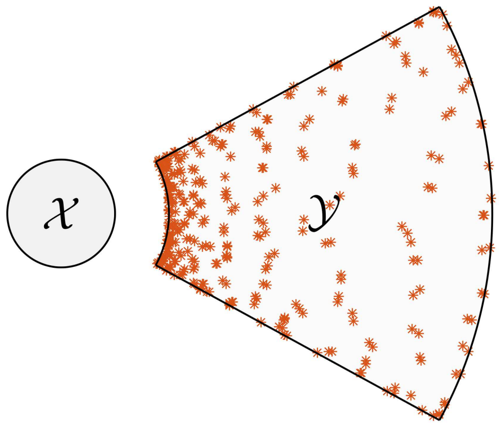


(b) relative approximation error

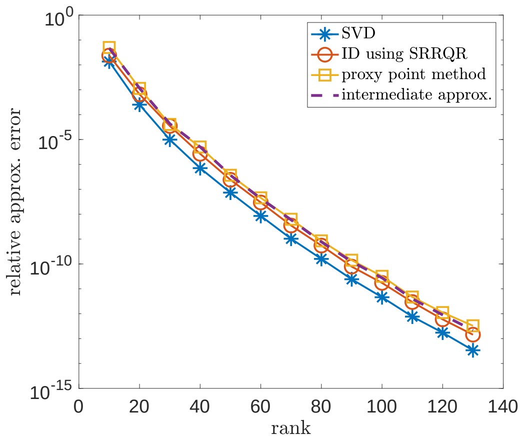
Figure 3.5: Basic test for problem setting 2: $K(x,y) = e^{-|x-y|^2}$, $|X_0| = 400$, $|Y_0| = 16000$, and $|Y_p| = 384$.

Y_p. The error function $e_i(y)$ for each $x_i \in X_0$ is then defined by the ID approximation of $K(X_0,Y_p)$ with error threshold ε_{id}. For different Y_p, Figure 3.8 plots $\max_{x_i\in X_0} \|e_i(y)\|_\infty$ and its upper bound derived from eq. (3.23), i.e.,

$$\max_{x_i\in X_0} \|e_i(y)\|_\infty \lesssim (r^{\frac{3}{2}}\varepsilon_* + \varepsilon_{\text{id}}) \max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2, \tag{3.28}$$

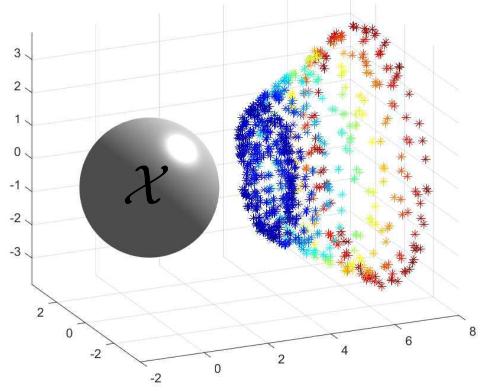


(a) proxy points

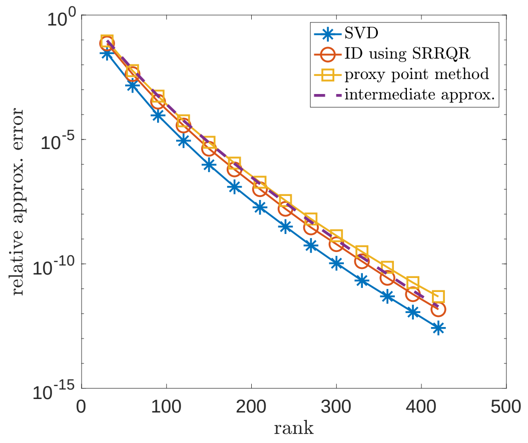


(b) relative approximation error

Figure 3.6: Basic test for problem setting 3: $K(x, y) = e^{2\pi i|x-y|}$, $|X_0| = 1300$, $|Y_0| = 6000$, and $|Y_p| = 330$.



(a) proxy points



(b) relative approximation error

Figure 3.7: Basic test for problem setting 4: $K(x, y) = e^{2\pi i|x-y|}/|x - y|$, $|X_0| = 1700$, $|Y_0| = 22500$, and $|Y_p| = 870$. Domain $\mathcal{Y}$ is not plotted. The selected proxy points are colored in (a) based on their x-axis coordinates for better visualization. Most of the proxy points are near the boundary of $\mathcal{Y}$ excluding the outer hemispherical surface, i.e., $|y| = 8$.

where ε_* and $\max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2$ are numerically estimated for each set of proxy points Y_p based on the associated set of basis functions $\{K(x_j, y)\}_{x_j\in X_p}$. Although having a large gap, the upper bound eq. (3.28) captures the changing trend of $\max_{x_i\in X_0} \|e_i(y)\|_\infty$. Further numerical tests show that the large gap between the upper bound eq. (3.28) and the actual value $\max_{x_i\in X_0} \|e_i(y)\|_\infty$ is mainly due to the loose estimate of $\|e_i(y) - \mathcal{L}e_i(y)\|_\infty$ in eq. (3.17), i.e.,

$$\|e_i(y) - \mathcal{L}e_i(y)\|_\infty \leqslant (1 + \max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2)(1 + C_{\mathrm{qr}}|Y_p|)\varepsilon_* \lesssim r^{\frac{3}{2}}\varepsilon_* \max_{y\in\mathcal{Y}} \|S_{Y_p}(y)\|_2,$$

which is used to derive the first term in eq. (3.28).

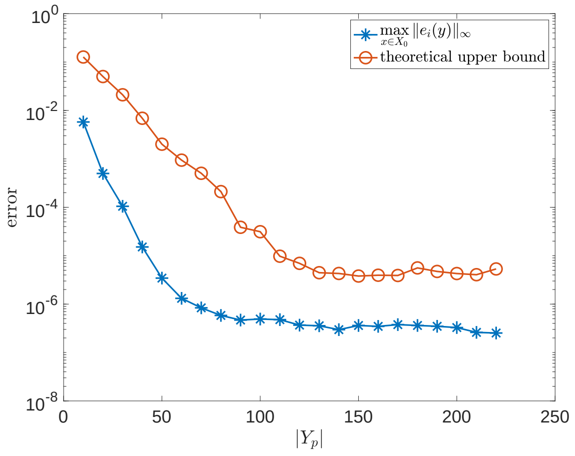


Figure 3.8: Values of $\max_{x_i\in X_0} \|e_i(y)\|_\infty$ and its upper bound eq. (3.28) for different numbers of proxy points selected by a simple modification of Algorithm 5.

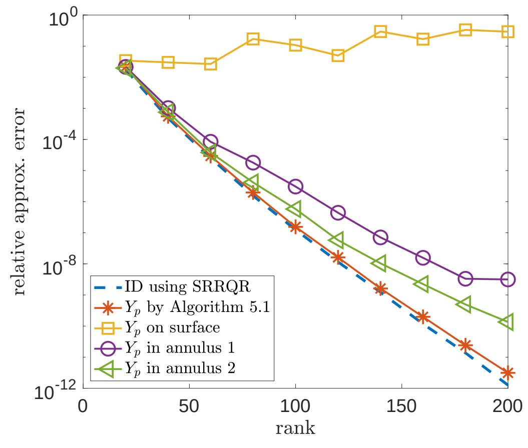
3.4.3 Comparison of different selections of proxy points

As illustrated in the beginning of the chapter, improper selections of proxy points Y_p can lead to much larger approximation errors in the proxy point method when compared to SVD. In this subsection, we compare the proposed proxy point selection scheme in Algorithm 5 with existing heuristic selection schemes.

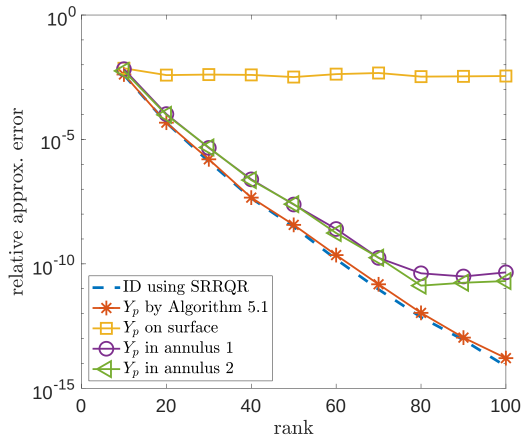
Consider again $K(x, y) = e^{-|x-y|^2}$, $\mathcal{X} = [-1, 1]^2$, and $\mathcal{Y} = [-7, 7]^2\backslash[-3, 3]^2$. Algorithm 5 obtains 384 proxy points with densification, as illustrated previously in Figure 3.5. We test three other heuristic choices of 384 proxy points: (1) uniform selection on the surface $\partial[-3, 3]^2$, (2) uniform and random selection in the annulus $[-3.2, 3.2]^2\backslash[-3, 3]^2$, (3) uniform and random selection in another wider annulus $[-3.5, 3.5]^2\backslash[-3, 3]^2$. The heuristic selection of proxy points in an annulus was suggested in Ref. [15] for Gaussian kernels.

We randomly and uniformly select 400 points in $\mathcal{X}$ for X_0 and 16000 points in $\mathcal{Y}$ for Y_0. The relative approximation errors of the proxy point method with the above four sets of proxy points for $K(X_0, Y_0)$ are plotted in Figure 3.9a. We apply the same test to another Gaussian kernel $K(x, y) = e^{-0.1|x-y|^2}$ with the same domains $\mathcal{X}$ and $\mathcal{Y}$, where 194 proxy points are selected by Algorithm 5. The corresponding relative approximation errors are plotted in Figure 3.9b.

As can be observed, Algorithm 5 outperforms the three heuristic selection schemes. With these heuristic schemes, the relative approximation error can stop decreasing or even increase when the approximation rank becomes large. One part of the reason for the increasing approximation error is the numerical instability of computing the ID approximation of $K(X_0, Y_p)$ for a large approximation rank. With an improper selection of proxy points Y_p (especially the selection on the surface), $K(X_0, Y_p)$ can have numerical rank smaller than the actual rank needed for an ID approximation of $K(X_0, Y_0)$ to obtain a given accuracy. As a result, the ID approximation of $K(X_0, Y_p)$ computed by SRRQR with a given approximation rank larger than the numerical rank of $K(X_0, Y_p)$ becomes numerically unstable.



(a) $K(x, y) = e^{-|x-y|^2}$



(b) $K(x, y) = e^{-0.1|x-y|^2}$

Figure 3.9: Relative approximation error of the proxy point method with different selections of proxy points for two kernel functions. In the legend, "annulus 1" refers to $[-3.2, 3.2]^2 \backslash [-3, 3]^2$ and "annulus 2" refers to $[-3.5, 3.5]^2 \backslash [-3, 3]^2$.

3.4.4 Comparison with algebraic compression methods

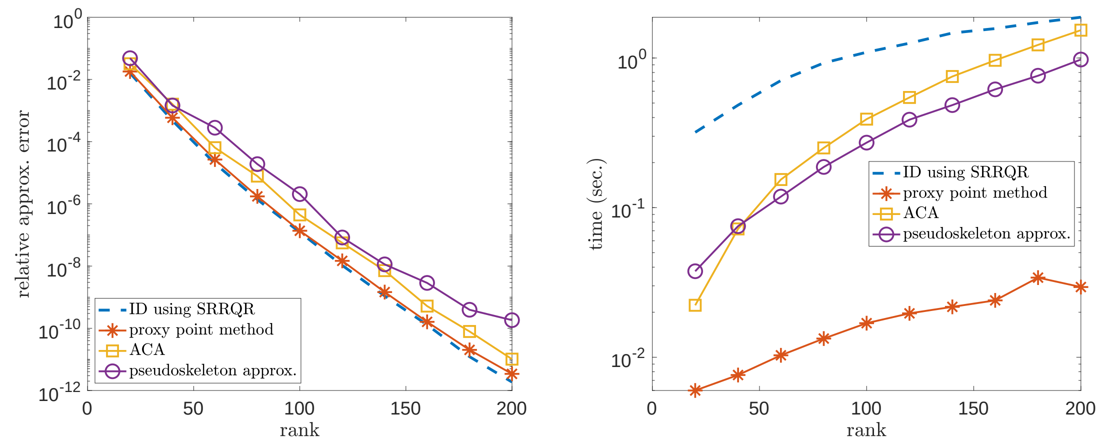
In this subsection, we compare the proxy point method with three algebraic compression methods: (1) ID using SRRQR, (2) adaptive cross approximation (ACA) with partial pivoting [29], and (3) a pseudoskeleton approximation method based on random column sampling and the pivoted QR decomposition that is similarly presented in Ref. [35].

Consider the same test settings as in the previous subsection: $K(x,y) = e^{-|x-y|^2}$, $\mathcal{X} = [-1,1]^2$, $\mathcal{Y} = [-7,7]^2 \backslash [-3,3]^2$, 384 proxy points selected by Algorithm 5, 400 points in $\mathcal{X}$ for X_0, and 16000 points in $\mathcal{Y}$ for Y_0. For different approximation ranks, Figure 3.10 plots the relative approximation errors and running times of the four different compression methods. As can be observed, the proxy point method has slightly better approximation accuracy and much less computational cost compared to ACA and the pseudoskeleton approximation method. However, it is worth noting that the accuracy difference between these tested methods is also related to the actual distribution of points Y_0 in the domain $\mathcal{Y}$. Meanwhile, all these methods and tests are implemented in Matlab without code optimization. These facts could also affect the test results. More tests are needed for a comprehensive comparison of these methods but these are outside the scope of this chapter.

We note that the running time of the proxy point method in Figure 3.10 does not include the running time of the proxy point selection by Algorithm 5, which is 5.9 seconds. As discussed in Section 3.3.4, the proxy point selection by Algorithm 5 should be viewed as a precomputation step and the selected proxy points are used repeatedly when applying the proxy point method in rank-structured matrix construction (see the numerical test in Section 3.4.5). Thus, it is reasonable to ignore the running time of the proxy point selection when comparing the proxy point method with other algebraic compression methods.

3.4.5 Application to $\mathcal{H}^2$ matrix construction

We now consider using the proxy point method to efficiently construct $\mathcal{H}^2$ matrix representations with the strong admissibility condition for kernel matrices $K(X,X)$ defined by



(a) relative approximation error (b) running time

Figure 3.10: Relative approximation error and running time of the proxy point method and three different algebraic compression methods. The running time of the proxy point method in (b) does not include the running time of the proxy point selection by Algorithm 5, which is 5.9 seconds.

a non-oscillatory symmetric kernel function $K(x, y)$ and randomly generated points X. Readers can refer to Section 2.6 for details of the $\mathcal{H}^2$ matrix construction with the associated blocks to be compressed into ID form. Also, readers can refer to Section 5.3 for a comparative study of an $\mathcal{H}^2$ matrix library currently under development with collaborators based on the proxy point method and another two commonly used FMM libraries.

In d-dimensional space ($d = 2$ or $d = 3$), we uniformly and randomly select N points in a square or cubical box with edge length $L = N^{\frac{1}{d}}$ for X. This box, enclosing all the points, is recursively partitioned into smaller boxes. Specifically, a box is uniformly subdivided into 2^d smaller boxes if it contains more than 300 points. With uniformly distributed points, we partition all the boxes at every level until the finest level. At the kth level of the recursive partitioning, the original box is partitioned into 2^{dk} boxes with edge length $L/2^k$. Such a recursive partitioning has $O(\log N)$ levels in total.

$\mathcal{H}^2$ matrix construction at the kth level needs to compute ID approximations of 2^{dk} number of kernel matrix blocks. These blocks share the same form $K(X_0, Y_0)$ where, for some box $\mathcal{B}$ at the kth level, X_0 is a set of points lying in $\mathcal{B}$ and Y_0 is a set of points lying in the far field of $\mathcal{B}$. When $K(x, y)$ is translationally invariant, we only need to select proxy points Y_p using Algorithm 5 for $K(x, y)$ with domains

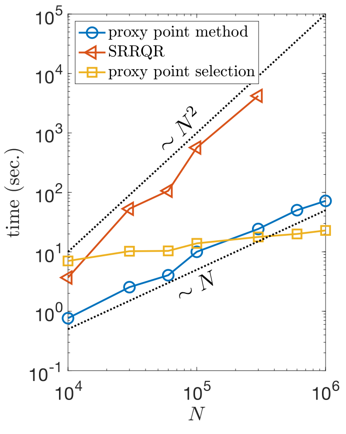
$$\mathcal{X} = \left[-\frac{L}{2^{k+1}}, \frac{L}{2^{k+1}}\right]^d \quad \text{and} \quad \mathcal{Y} = \left[-(L - \frac{L}{2^{k+1}}), L - \frac{L}{2^{k+1}}\right]^d \setminus \left[-\frac{3L}{2^{k+1}}, \frac{3L}{2^{k+1}}\right]^d.$$

We can then apply the proxy point method to each $K(X_0, Y_0)$ at the kth level with Y_p properly shifted according to the position of $\mathcal{B}$ relative to $\mathcal{X}$.

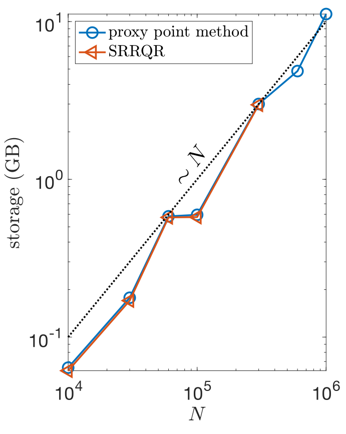
We test two different problem settings: $K(x, y) = 1/\sqrt{1 + |x - y|^2}$ in 2D and $K(x, y) = (1 + 0.01|x - y|)e^{-0.01|x-y|}$ in 3D. The proxy point method is used to compute all the ID approximations in the $\mathcal{H}^2$ matrix construction, where all the intermediate ID approximations of $K(X_0, Y_p)$ in the method are computed with relative error threshold $\tau = 10^{-6}$. Here, an ID approximation UA_J to a matrix A meets a relative error threshold τ if the 2-norm of each row of $A - UA_J$ is bounded by τ times the maximum of the 2-norm of each row of A. For comparison, the $\mathcal{H}^2$ matrix construction with all the ID approximations computed using SRRQR with relative error threshold $\tau = 10^{-6}$ is also tested. All the tests are implemented in Matlab.

Figures 3.11 and 3.12 plot the construction time, storage cost, and relative error of $\mathcal{H}^2$ matrix constructions for the two problem settings, respectively. For a constructed $\mathcal{H}^2$ matrix, its relative error is measured as the relative error of 100 randomly chosen entries of the product of the $\mathcal{H}^2$ matrix with a random vector compared with the entries of the exact product of the original kernel matrix with the vector. The reported relative error results are averaged over 20 independent tests.

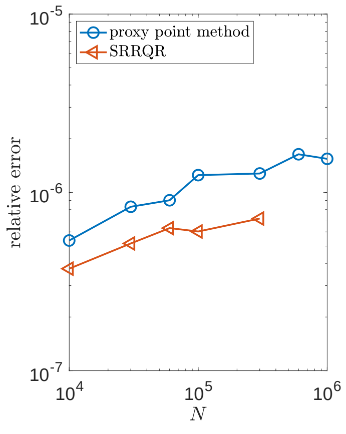
The runtime of selecting proxy points is significant but its asymptotic complexity is only $O(\log N)$ since Algorithm 5 is applied only once at each level of $\mathcal{H}^2$ matrix construction. The proxy point method leads to nearly linear $\mathcal{H}^2$ matrix construction, which can also be justified theoretically if we assume that the number of proxy points Y_p selected at each level is $O(1)$. Also, the storage cost and relative error of $\mathcal{H}^2$ matrices constructed by the proxy point method are close to those by SRRQR, indicating that the proxy point method is as



(a) construction time

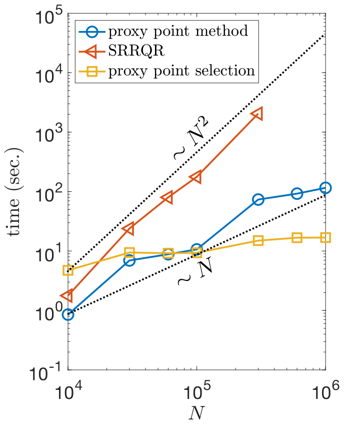


(b) storage cost

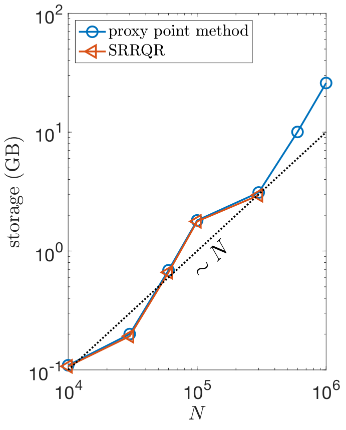


(c) relative error

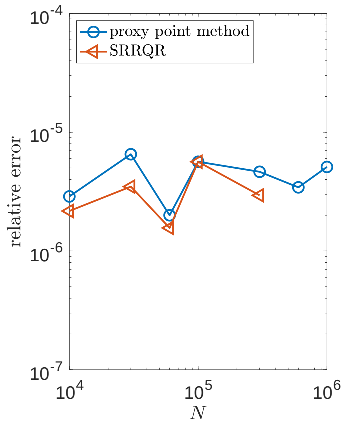
Figure 3.11: Results of $\mathcal{H}^2$ matrix construction for $K(x,y) = 1/\sqrt{1+|x-y|^2}$ in 2D: (a) construction time of $\mathcal{H}^2$ matrices, (b) storage cost of the constructed $\mathcal{H}^2$ matrices, (c) relative error of the constructed $\mathcal{H}^2$ matrices. "Proxy point method" and "SRRQR" refer to the ID approximation methods used in the $\mathcal{H}^2$ matrix construction. In (a), "proxy point selection" refers to the total runtime of selecting the proxy points by Algorithm 5 for $\mathcal{H}^2$ matrix construction. Reference lines for linear and quadratic scaling with N are plotted.



(a) construction time



(b) storage cost



(c) relative error

Figure 3.12: Results of $\mathcal{H}^2$ matrix construction for $K(x,y) = (1+0.01|x-y|)e^{-0.01|x-y|}$ in 3D: (a) construction time of $\mathcal{H}^2$ matrices, (b) storage cost of the constructed $\mathcal{H}^2$ matrices, (c) relative error of the constructed $\mathcal{H}^2$ matrices.

effective as SRRQR in terms of the approximation rank and accuracy for ID approximations in $\mathcal{H}^2$ matrix construction.

CHAPTER 4
FAST COULOMB MATRIX CONSTRUCTION VIA COMPRESSING THE INTERACTIONS BETWEEN CONTINUOUS CHARGE DISTRIBUTIONS

In this chapter, we extend the applicability of the proxy point method and $\mathcal{H}^2$ matrix representations to matrices defined by interactions between continuous charge distributions for fast Coulomb matrix construction in quantum chemistry. Constructing the Coulomb matrix is one of the main steps in many quantum chemical methods. The Coulomb matrix can be defined as

$$J_{ab} = \sum_{c,d} (\phi_a \phi_b | \phi_c \phi_d) D_{cd}, \tag{4.1}$$

where D is a density matrix and $(\phi_a \phi_b | \phi_c \phi_d)$ denotes an entry of a four-dimensional electron repulsion integral (ERI) tensor. Each entry of the ERI tensor is defined as

$$(\phi_a \phi_b | \phi_c \phi_d) = \int_{\mathbb{R}^3} \int_{\mathbb{R}^3} \phi_a(r_1) \phi_b(r_1) \frac{1}{|r_1 - r_2|} \phi_c(r_2) \phi_d(r_2) \mathrm{d}r_1 \mathrm{d}r_2, \tag{4.2}$$

where ϕ_a, etc., are known basis functions. In quantum chemical methods where high accuracy is desired, the standard basis functions are Gaussian-type functions (GTFs)

$$\phi_a(r) = (x - x_a)^l (y - y_a)^m (z - z_a)^n e^{-\alpha |r - r_a|^2},$$

where $r_a = (x_a, y_a, z_a)$ is the center of the function, α is an exponent, and $(l+m+n)$ is the total angular momentum. (In practice, the basis functions are a known linear combination of GTFs that have the same center, and are called *contracted* GTFs. This fact does not change the development of this chapter, and it will be ignored until Section 4.5 on numerical experiments.)

In self-consistent field iterations, the Coulomb matrix is constructed repeatedly for different density matrices while the ERI tensor is fixed. The computational challenge in constructing the Coulomb matrix is the fact that the ERIs are expensive to compute and, for typical numbers of basis functions, the distinct, non-negligible ERIs are too numerous to store in memory. The ERI tensor is central to many quantum chemical methods and a variety of techniques have been developed to approximate the ERI tensor to reduce computation and/or storage costs.

From eqs. (4.1) and (4.2), constructing the Coulomb matrix calculates the Coulomb potential for a system of *continuous charge distributions*. Here, $\phi_a\phi_b$ and $\phi_c\phi_d$ are distributions; the latter multiplied by the corresponding charge weight D_{cd} is a charge distribution. From this viewpoint, the ERI tensor $(\phi_a\phi_b|\phi_c\phi_d)$ can be regarded as a matrix by folding together its first two dimensions and folding together its last two dimensions so that (a, b) denotes a matrix row index and (c, d) denotes a matrix column index. We will refer to this matrix as the *ERI matrix* whose entries correspond to the pairwise interactions between distributions. At the same time, the density matrix can be regarded as a vector. Thus, the tensor contraction eq. (4.1) can be viewed as a matrix-vector multiplication.

The discrete case, i.e., the Coulomb potential for a system of *point* charges where the ERI matrix is replaced by a kernel matrix defined by $K(x, y) = 1/|x-y|$, can be efficiently calculated by FMM and $\mathcal{H}^2$ matrix technique. For the case of continuous distributions, the continuous fast multipole method (CFMM) and related methods have been developed for constructing the Coulomb matrix [36, 37, 38, 39, 40, 41]. These methods use the multipole expansion technique from FMM to compress the interactions between "well-separated" distributions into low-rank form and thus to accelerate the evaluation of these interactions. The remaining interactions are evaluated directly.

Similar to FMM, CFMM is also algebraically equivalent to multiplying the ERI matrix in $\mathcal{H}^2$ format. However, CFMM is not as efficient as one would hope. For two distributions to be well-separated, they cannot overlap (to be described precisely in Section 4.2). As a result, CFMM essentially uses a much stronger admissibility condition than that commonly used in FMM and $\mathcal{H}^2$ matrices for point charges, and thus defines far more inadmissible blocks that have to be evaluated directly. For typical problems, a large number of distributions overlap, and thus the number of interactions that must be evaluated directly is large [39]. These direct computations dominate the computation cost of CFMM.

In this chapter, we propose to construct an explicit $\mathcal{H}^2$ matrix representation of the ERI matrix using an admissibility condition that is much weaker than the one used by CFMM. This $\mathcal{H}^2$ matrix representation allows us to compress far more interactions in the ERI matrix than could be compressed by CFMM, resulting in far fewer interactions that must be computed directly. We then simply use the fast matrix-vector multiplication algorithm for $\mathcal{H}^2$ matrices to construct the Coulomb matrix.

The main challenge of this new $\mathcal{H}^2$-based approach is the expensive computation required to represent the ERI matrix in this specific $\mathcal{H}^2$ format. The multipole expansion technique in CFMM cannot be applied here since the block to be compressed can correspond to overlapping distributions. To tackle this challenge, we propose a new variant of the proxy point method which, just like the proxy point method for kernel matrices, can avoid needing to explicitly form a block of the ERI matrix before compressing it. Experimentally, the computation cost of $\mathcal{H}^2$ matrix construction with this new variant turns out to be nearly linear in the number of distributions. Combining with the linear-scaling $\mathcal{H}^2$ matrix-vector multiplication algorithm, this $\mathcal{H}^2$-based approach to constructing the Coulomb matrix has linear computation cost. This cost is still directly related to the number of direct interactions in the $\mathcal{H}^2$ matrix representation, but we have effectively reduced this number compared to CFMM.

The rest of the chapter is organized as follows.

- Section 4.1 introduces the previous work for approximating the ERI tensor to reduce computation and/or storage cost.
- Section 4.2 describes the limitation of CFMM and experimentally illustrates that there are more numerically low-rank blocks than those compressed by CFMM.

- Section 4.3 describes the $\mathcal{H}^2$ matrix representation of the ERI matrix with the associated blocks compressed into ID form.
- Section 4.4 describes the variant of the proxy point method which efficiently compresses the associated ERI matrix blocks in $\mathcal{H}^2$ matrix construction.
- Section 4.5 describes the numerical experiments for the proposed $\mathcal{H}^2$-based approach to constructing the Coulomb matrix.
- Section 4.6 concludes this extended application of the proxy point method and $\mathcal{H}^2$ matrix technique and discusses future work in quantum chemical computations.

4.1 Previous work

Besides CFMM, there have been significant efforts in the past to develop and use compressed representations of the ERI tensor. Density fitting (e.g., [42, 43, 44]) and its variants [45, 46, 47, 48, 49] represent the 4-index ERI tensor as the contraction of two 3-index tensors or as the contraction of two 3-index tensors and a 2-index tensor. Pseudospectral method [46] represents the ERI tensor as the contraction of a 3-index tensor and two 2-index tensor. Other decompositions of the 4-index ERI tensor, called tensor hypercontraction, have also been recently developed [50].

Block low-rank matrix representations have been used elsewhere in quantum chemistry as well. Lewis, Calvin, and Valeev [51] use a 1-level matrix representation called "clustered low-rank" for the 2-index and 3-index tensors in density fitting. Lu and Ying [52] use interpolative decompositions to produce approximations of the ERI tensor in tensor hypercontraction form.

4.2 Limitations of CFMM

In FMM and CFMM, space is partitioned into boxes and the potential at a point far from a box due to the point charges (FMM) or charge distributions (CFMM) centered in the box is expressed in terms of a multipole expansion. In FMM, if two boxes are *not adjacent*, then the multipole expansion could be used to compactly describe the pairwise interactions between the point charges across the two boxes. In CFMM, it is more complicated to determine whether or not a multipole expansion could be used to approximate the interactions between charge distributions.

To explain the issue with charge distributions, consider the distribution $\phi_a\phi_b$ which is a product of two GTFs. By the Gaussian product rule, $\phi_a\phi_b$ itself is a GTF with center along the line joining the centers of ϕ_a and ϕ_b. For a distribution ϕ, in general, define its *extent* λ with *precision* τ as the radius of the smallest ball centered at the center of the GTF such that $|\phi(r)|$ is less than τ outside the ball [38]. Whether two distributions overlap depends on whether these balls overlap.

Now consider two sets of distributions, $\Phi = \{\varphi_i\}$ being the distributions centered in a given box, and $\Theta = \{\theta_j\}$ being the distributions centered in a non-adjacent box. Define V_Φ to be the *numerical support* of Φ, that is, the convex hull of the balls corresponding to the distributions in Φ, and define V_Θ similarly. In this notation, $(\Phi|\Theta)$ is a block of the ERI matrix and each of its entries is an ERI,

$$(\varphi_i|\theta_j) = \int_{V_\Phi}\int_{V_\Theta} \varphi_i(r_1)\frac{1}{|r_1 - r_2|}\theta_j(r_2)\mathrm{d}r_1\mathrm{d}r_2, \quad \varphi_i \in \Phi,\ \theta_j \in \Theta.$$

If V_Φ and V_Θ do not overlap, then we can approximate $1/|r_1 - r_2|$ by a multipole expansion,

$$(\varphi_i|\theta_j) \approx \int_{V_\Phi}\int_{V_\Theta} \varphi_i(r_1)\left(\sum_{l=0}^{s}\sum_{m=-l}^{l} |r_2|^l Y_l^{-m}(r_2)\frac{Y_l^m(r_1)}{|r_1|^{l+1}}\right)\theta_j(r_2)\mathrm{d}r_1\mathrm{d}r_2,$$

where $Y_l^m(r)$ is the spherical harmonic function of degree l and order m. Here, the multipole expansion is of degree s and is centered at the origin which is assumed to be the center of V_Θ. The above expansion gives an approximation in degenerate form,

$$(\varphi_i|\theta_j) \approx \sum_{l=0}^{s}\sum_{m=-l}^{l}\left(\int_{V_\Phi}\frac{Y_l^m(r_1)}{|r_1|^{l+1}}\varphi_i(r_1)\mathrm{d}r_1\right)\left(\int_{V_\Theta}|r_2|^l Y_l^{-m}(r_2)\theta_j(r_2)\mathrm{d}r_2\right),$$

which is equivalent to a rank-$(s+1)^2$ approximation of $(\Phi|\Theta)$.

If V_Φ and V_Θ do overlap, then the above approximation is not possible, since the multipole expansion of $1/|r_1 - r_2|$ diverges when r_1 and r_2 are equal. In this situation, computing the interactions between distributions in these two boxes cannot be accelerated by CFMM and these interactions must be computed directly. The distinguishing feature of CFMM compared to FMM is the need to identify sets of distributions whose numerical supports do not overlap.

An important observation is that even if V_Φ and V_Θ do overlap, the ERI matrix block

$(\Phi|\Theta)$ may still be numerically low-rank if Φ and Θ are distributions centered in non-adjacent boxes. We simply do not have the analytical apparatus to find these low-rank approximations. In this chapter, a new technique is proposed to find such approximations.

To illustrate the observation that $(\Phi|\Theta)$ may be numerically low-rank although there is no corresponding known degenerate expansion, consider two non-adjacent cubical boxes of edge length $L = 5$ centered at $(0, 0, 0)$ and $(2L, 0, 0)$. For each box, select 600 GTF distributions of the form $(p/\pi)^{3/2} e^{-p|r-r_a|^2}$ with the same exponent p and different centers r_a randomly distributed in the box. These GTFs represent very simple "spherical" distributions.

As before, denote the two sets of GTFs as $\Phi = \{\varphi_i\}$ and $\Theta = \{\theta_j\}$. Denote the center of each distribution φ_i as x_i and the center of each distribution θ_j as y_j. Each entry of the ERI matrix block $(\Phi|\Theta)$ can be calculated analytically as

$$(\varphi_i|\theta_j) = \frac{1}{|x_i - y_j|} \mathrm{erf}\left(\sqrt{\frac{p}{2}}|x_i - y_j|\right). \tag{4.3}$$

Figure 4.1 plots the first 300 singular values of $(\Phi|\Theta)$ for four different cases, corresponding to different values of the exponent p in the GTFs, and thus GTFs with different extents. The extent $\lambda = \sqrt{\frac{1}{p}\left(\frac{3}{2}\ln\frac{p}{\pi} + \ln\frac{1}{\tau}\right)}$ for each value of p is also shown in each subfigure, assuming the extent precision $\tau = 10^{-10}$.

For comparison, we also plot in each subfigure the singular values of the matrix which we denote as $K(X, Y)$, consisting of the entries $K(x_i, y_j)$ for all pairs of centers x_i and y_j, with $K(x, y) = 1/|x - y|$. This is the matrix that describes the Coulomb interactions if we had point charges (instead of distributions) at the location of each center. Since the two boxes under consideration are non-adjacent, the singular values of $K(X, Y)$ decay rapidly and $K(X, Y)$ is numerically low-rank. FMM considers these two sets of point charges to be well separated.

When $p = 10$, the extent λ of the distributions is small, and $(\Phi|\Theta)$ and $K(X, Y)$ have

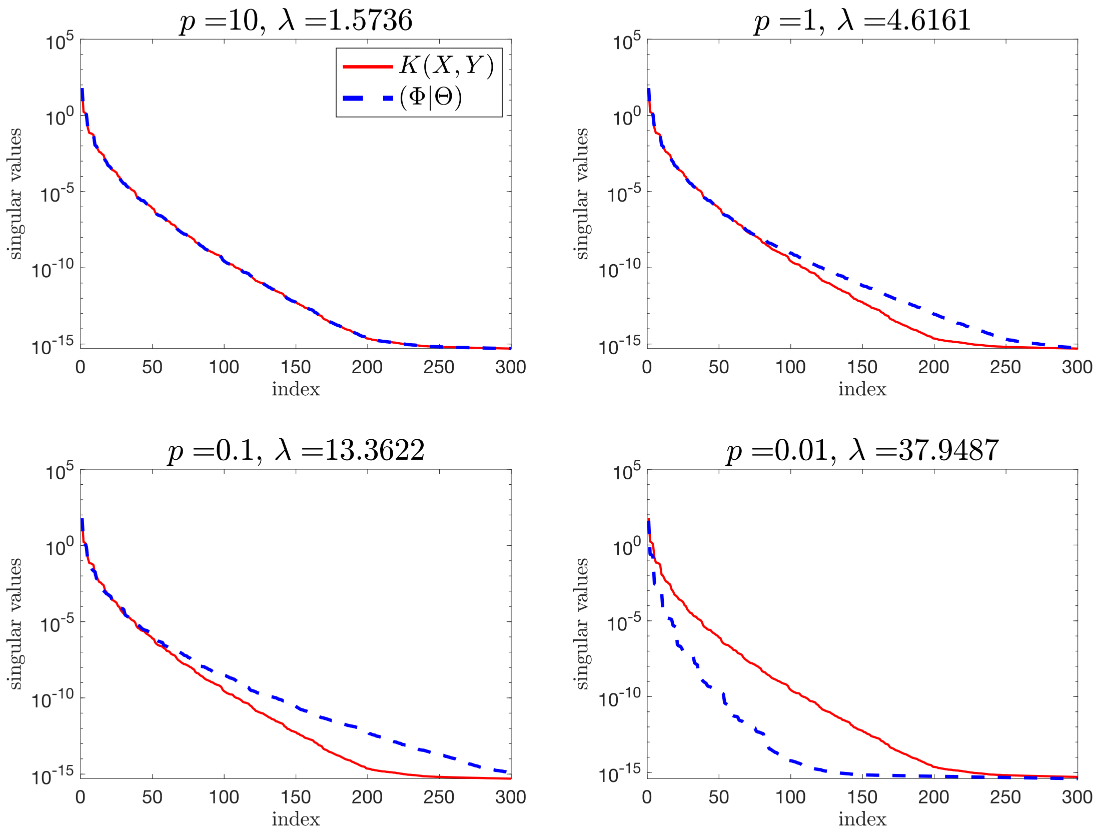


Figure 4.1: First 300 singular values of $K(X, Y)$ and $(\Phi|\Theta)$ for GTFs with different exponents p. For each p, the corresponding value of the extent λ is also shown. Only the ERI block $(\Phi|\Theta)$ with $p = 10$ can be compressed using multipole expansions in CFMM, although $(\Phi|\Theta)$ in other cases is also numerically low-rank.

very similar singular values. When $p = 1$ and $p = 0.1$, the extent is larger and the distributions from the two boxes can overlap. CFMM would consider the interactions between these two boxes to be near-range in these cases, i.e., interactions based on multipole expansions cannot be used. However, Figure 4.1 shows that the singular value decay of $(\Phi|\Theta)$ and $K(X, Y)$ is similar for the first 8 or more decades of singular values. Thus $(\Phi|\Theta)$ in these cases are also numerically low-rank.

When $p = 0.01$, the distributions are very diffusive and the singular value decay of $(\Phi|\Theta)$ is even faster than that of $K(X, Y)$. This odd result turns out to be quite natural from the viewpoint of kernel functions. With a sufficiently small p, the formula eq. (4.3), regarded as a kernel function between x_i and y_j, can be flatter than $1/|x_i - y_j|$ for x_i and y_j

in the two non-adjacent boxes. Heuristically, this flatness usually indicates that eq. (4.3) can be well approximated by a degenerate approximation with smaller degrees than $1/|x_i - y_j|$, leading to the faster singular value decay of $(\Phi|\Theta)$ than that of $K(X, Y)$.

Based on these observations, and unlike CFMM, we will only use the centers of distributions, rather than both centers and extents, to decide whether an interaction can be compressed by a low-rank approximation. A challenge is how to efficiently find such low-rank approximations.

4.3 $\mathcal{H}^2$ matrix representation of the ERI matrix

In this section, we establish notation for representing and constructing the ERI matrix in $\mathcal{H}^2$ format based on Chapter 2.

Hierarchical partitioning and ERI matrix blocks Constructing an $\mathcal{H}^2$ matrix representation of the ERI matrix starts with a hierarchical partitioning of the set of distributions, or basis function products $\{\phi_a\phi_b\}$, for the molecular system and chosen basis set. Like in the case of points, the space enclosing all the distributions is partitioned recursively and adaptively into cubic boxes until the number of distributions centered in each finest box is less than a prescribed small constant. This hierarchical partitioning can be represented by an octree whose nodes correspond to the boxes.

Let $\mathcal{T}$ denote the partition tree. For brevity, we always assume $\mathcal{T}$ to be a perfect octree in this chapter. The following discussion can be easily extended to the non-perfect partition tree case based on Section 2.4. Let I denote the set of all distributions. Let I_i denote the set of distributions with centers in box i and corresponding to node i in the tree. Using this notation, $(I_i|I_j)$ denotes the block in the ERI matrix corresponding to the Coulomb interactions between distributions with centers in boxes i and j. The entire ERI matrix can be denoted as $(I|I)$.

Admissibility condition According to the last section, we choose to use an admissibility condition that is similar to the strong admissibility condition used for points in Section 2.1 to decide whether a block in the ERI matrix is numerically low-rank. Specifically, the *far field* $\mathcal{Y}$ of a box $\mathcal{X}$ is defined as the complement of the union of $\mathcal{X}$ and all its adjacent same-sized boxes. For two sets of distributions I_0 and J_0 with centers in $\mathcal{X}$ and $\mathcal{Y}$, respectively, the ERI matrix block $(I_0|J_0)$ is expected to be numerically low-rank (to be justified experimentally in Section 4.5) and thus is to be compressed into low-rank form in the $\mathcal{H}^2$ matrix representation.

4.3.1 $\mathcal{H}^2$ matrix construction

At each level l, let level(l) denote the set of nodes at level l and $\cup_{i\in \text{level}(l)} I_i = I$. For each node $i \in$ level(l), according to the admissibility condition, boxes in level(l) are split into two subsets,

$$\mathcal{F}_i = \{k \in \text{level}(l) \mid \text{box } k \text{ is in the far field of box } i\}$$
$$\mathcal{N}_i = \text{level}(l) \setminus \mathcal{F}_i.$$

Accordingly, $(I_i|I_j)$ with $j \in \mathcal{F}_i$ is an admissible block and $(I_i|I_j)$ with $j \in \mathcal{N}_i$ is an inadmissible block. Let $J_i = \cup_{k\in\mathcal{F}_i} I_k$ which contains all the distributions in the far field of box i. To construct an $\mathcal{H}^2$ matrix representation of $(I|I)$, $(I_i|J_i)$ and $(J_i|I_i)$ are to be compressed into low-rank form. Since $(I_i|J_i) = (J_i|I_i)^T$, we only need to compress $(I_i|J_i)$ and set the uniform row basis matrix to be the same as the computed uniform column basis matrix for each node i.

Consider a rank-r_0 approximation of $(I_i|J_i)$ in interpolative decomposition (ID) form,

$$(I_i|J_i) \approx U_i(I_i^{\text{id}}|J_i), \tag{4.4}$$

where U_i has r_0 columns, I_i^{id} is a subset of I_i, and $(I_i^{\text{id}}|J_i)$ contains r_0 rows of $(I_i|J_i)$.

For each non-leaf node i with children $\{i_1, \dots, i_8\}$, the nested ID approximation of $(I_i|J_i)$ starts with splitting and approximating the block as

$$(I_i|J_i) = \begin{pmatrix} (I_{i_1}|J_i) \\ \vdots \\ (I_{i_8}|J_i) \end{pmatrix} \approx \begin{pmatrix} U_{i_1}(I_{i_1}^{\text{id}}|J_i) \\ \vdots \\ U_{i_8}(I_{i_8}^{\text{id}}|J_i) \end{pmatrix} = \begin{pmatrix} U_{i_1} & & \\ & \ddots & \\ & & U_{i_8} \end{pmatrix} \begin{pmatrix} (I_{i_1}^{\text{id}}|J_i) \\ \vdots \\ (I_{i_8}^{\text{id}}|J_i) \end{pmatrix}.$$

The last matrix above is then compressed into low-rank form

$$\left(\cup_{i_a} I_{i_a}^{\text{id}}|J_i\right) \approx R_i(I_i^{\text{id}}|J_i) \tag{4.5}$$

with $I_i^{\text{id}} \subset \cup_{i_a} I_{i_a}^{\text{id}} \subset I_i$, which computes the transfer matrix R_i and approximates $(I_i|J_i)$ in ID form as

$$(I_i|J_i) \approx \begin{pmatrix} U_{i_1} & & \\ & \ddots & \\ & & U_{i_8} \end{pmatrix} R_i(I_i^{\text{id}}|J_i) = U_i(I_i^{\text{id}}|J_i).$$

In fact, the components R_i and I_i^{id} for a good ID approximation eq. (4.5) of $(\hat{I}_i|J_i)$ can be computed more efficiently than by using SRRQR directly. Define

$$\hat{I}_i = \cup_{i_a \in \{\text{children of } i\}} I_{i_a}^{\text{id}} \qquad \text{and} \qquad \hat{J}_i = \cup_{k \in \mathcal{F}_i} \cup_{k_a \in \{\text{children of } k\}} I_{k_a}^{\text{id}}. \tag{4.6}$$

The components R_i and I_i^{id} can then be computed from the ID approximation

$$(\hat{I}_i|\hat{J}_i) \approx R_i(I_i^{\text{id}}|\hat{J}_i). \tag{4.7}$$

Since $(\hat{I}_i|\hat{J}_i)$ has fewer columns than $(\hat{I}_i|J_i)$, the direct computation of eq. (4.7) is cheaper than that of eq. (4.5). Readers can refer to Refs. [14, 53] for more details.

One key benefit of using ID approximation to construct an $\mathcal{H}^2$ matrix is that the intermediate matrices $B_{i,j}$ can be efficiently computed (as already discussed in Section 2.6).

For each admissible block $(I_i|I_j)$ with $j \in \mathcal{F}_i$, the columns of the ID in eq. (4.4) that correspond to I_j give the approximation $(I_i|I_j) \approx U_i(I_i^{\text{id}}|I_j)$. Similarly, the ID approximation $(I_j|J_j) \approx U_j(I_j^{\text{id}}|J_j)$ gives $(I_j|I_i^{\text{id}}) \approx U_j(I_j^{\text{id}}|I_i^{\text{id}})$ based on the fact that $I_i^{\text{id}} \subset I_i \subset J_j$. Combining these two approximations leads to

$$(I_i|I_j) \approx U_i(I_i^{\text{id}}|I_j^{\text{id}})U_j^T. \tag{4.8}$$

Thus, the associated intermediate matrix $B_{i,j}$ can be set to $(I_i^{\text{id}}|I_j^{\text{id}})$, which can be directly computed using I_i^{id} and I_j^{id}.

An interesting observation is that the approximation eq. (4.8) has the form of a density fitting (DF) approximation [42, 43, 44] where I_i^{id} and I_j^{id} would correspond to the set of "auxiliary functions" for I_i and I_j, respectively. DF, however, is applied to the entire ERI matrix $(I|I)$. Thus, an $\mathcal{H}^2$ matrix representation of the ERI matrix can also be interpreted as a generalization of DF. This generalization locally and hierarchically applies DF to certain pairs of subsets of basis function products.

4.3.2 Comparison with CFMM

As already mentioned, CFMM is equivalent to multiplying the ERI matrix in a specific $\mathcal{H}^2$ format. The main difference between this equivalent $\mathcal{H}^2$ format and the one we have described in this section is that CFMM has a much stronger admissibility condition. In CFMM, an ERI matrix block is admissible (a.k.a. far-field) if the corresponding two sets of distributions have non-overlapping numerical supports. The block is inadmissible (a.k.a. near-field) otherwise. Additionally, CFMM has the same hierarchical partitioning of I but further splits each I_i into "branches" [40] according to the extents of distributions in I_i. Thus, a leaf-level block $(I_i|I_j)$ that corresponds to two sets of distributions with overlapping numerical supports is further subdivided into smaller blocks, some of which can be defined as admissible blocks. Also, CFMM uses the multipole expansion technique to compress the admissible blocks instead of ID approximations.

4.4 Accelerated compression via proxy points

For an ERI matrix, the construction of an $\mathcal{H}^2$ matrix representation is dominated by the cost of the ID approximation of $(I_i|J_i)$ for leaf nodes i and of $(\hat{I}_i|\hat{J}_i)$ for non-leaf nodes i. These ERI matrix blocks share the same form $(I_*|J_*)$ where, for some node i, I_* is a set of distributions (I_i or $\hat{I}_i$) centered in box i, and J_* is a set of distributions (J_i or $\hat{J}_i$) centered in the far field of box i. In general, the set J_* is much larger than the set I_*. Using purely algebraic methods such as SRRQR to compress $(I_*|J_*)$ leads to quadratic $\mathcal{H}^2$ construction cost, due to needing at least to form and examine every element in $(I_*|J_*)$. At the same time, the multipole expansion technique used in CFMM cannot generally be applied here, since I_* and J_* can have overlapping numerical supports.

This section introduces a variant of the proxy point method in Chapter 3 which is a hybrid analytic-algebraic method and can efficiently calculate an ID approximation of $(I_*|J_*)$ while avoiding the evaluation of all the elements in $(I_*|J_*)$.

4.4.1 Splitting of J_*

Consider two sets of distributions, I_* and J_*, as described above. Let $\mathcal{B}$ denote the box that encloses the centers of distributions in I_* and let $\mathcal{B}^{\text{adj}}$ denote the union of $\mathcal{B}$ and its 26 adjacent, same-sized boxes. All the distributions in J_* have their centers outside $\mathcal{B}^{\text{adj}}$. A 2D example of I_*, J_*, $\mathcal{B}$, and $\mathcal{B}^{\text{adj}}$ is illustrated in Figure 4.2.

We split J_* into two subsets, J_{near} and J_{far}, where J_{near} contains all the distributions in J_* that overlap with $\mathcal{B}^{\text{adj}}$ and $J_{\text{far}} = J_* \setminus J_{\text{near}}$. Figure 4.3 illustrates an example of this splitting. We note that this splitting of J_* is not related to the numerical support of I_*, and distributions in both J_{near} and J_{far} may overlap with distributions in I_*. Since all the distributions in J_* have bounded extents, J_{near} generally has $O(1)$ number of distributions and J_{far} can be of size on the order of the total number of distributions, i.e., $O(|I|)$. To

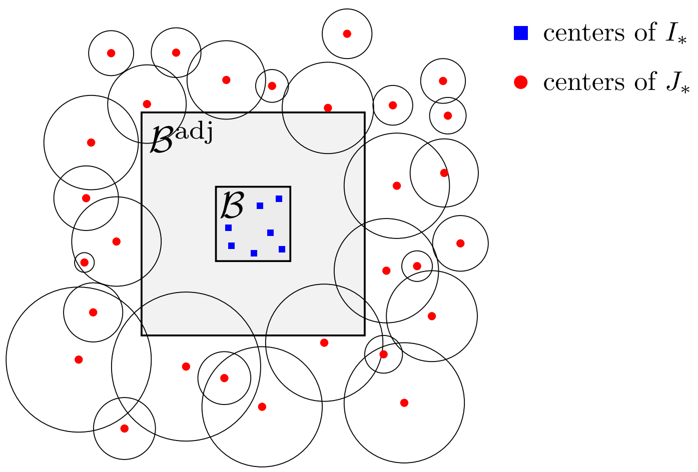


Figure 4.2: 2D illustration of I_*, J_*, $\mathcal{B}$, and $\mathcal{B}^{\text{adj}}$. Each circle around a red point denotes one distribution in J_*. The radius of a circle is the extent of a distribution. Distributions in I_* are not plotted but the balls associated with these distributions generally can spread outside $\mathcal{B}^{\text{adj}}$.

efficiently compute an ID approximation of $(I_*|J_*)$, we will consider two parts: $(I_*|J_{\text{near}})$ and $(I_*|J_{\text{far}})$. The former has a relatively small size; the latter is the critical part that we discuss now.

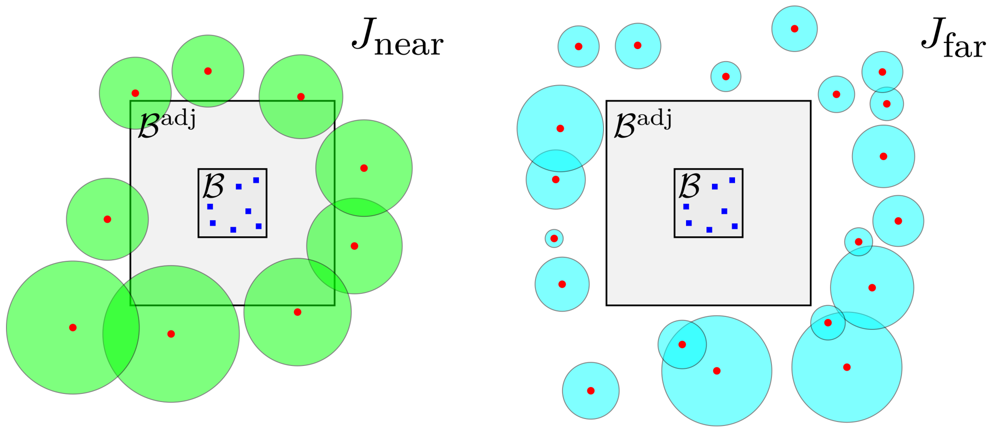


Figure 4.3: 2D illustration of the splitting of J_* (corresponding to Figure 4.2) into J_{near} and J_{far}, where J_{near} contains all the distributions that overlap with $\mathcal{B}^{\text{adj}}$ and $J_{\text{far}} = J_* \backslash J_{\text{near}}$.

For the case of point charges rather than charge distributions, methods already exist for computing low-rank approximation to $(I_*|J_{\text{far}})$ without needing to evaluate the entire matrix itself (note that $J_{\text{far}} = J_*$ for the case of point charges). As discussed in Chapter 3, in the proxy surface and related methods [10, 14, 21, 54], the points in J_{far} are replaced by a smaller set of *proxy points* on the surface $\partial\mathcal{B}^{\text{adj}}$ between the points in I_* and J_{far}. This does not work for charge distributions because the distributions in I_* and J_{far} may overlap. One could redefine J_{far} as the set of distributions that do not overlap with those of I_*, but this would result in a very large set of distributions J_{near} which we are trying to avoid in the first place.

Moreover, the proxy point method proposed in Chapter 3 can not be directly applied to $(I_*|J_{\text{far}})$ since the method is developed and analyzed based on kernel matrices while ERI matrix blocks are defined by specific integral-form interactions between distributions.

For the case of charge distributions, our approach to the low-rank approximation of $(I_*|J_{\text{far}})$ resembles the proxy point method and uses multiple layers of proxy points around $\partial\mathcal{B}^{\text{adj}}$. We begin below with a theoretical motivation for this approach.

4.4.2 Theoretical motivation

If we imagine each distribution φ_i in I_* is a unit charge distribution, then its induced potential $p_i(y)$ in $\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}$ is

$$p_i(y) = \int_{\mathbb{R}^3} \varphi_i(r)\frac{1}{|r-y|}\mathrm{d}r, \quad y \in \mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}. \tag{4.9}$$

For any $\varphi_i \in I_*$ and $\theta_j \in J_{\text{far}}$, the entry $(\varphi_i|\theta_j)$ of $(I_*|J_{\text{far}})$ can be written as

$$(\varphi_i|\theta_j) = \int_{\mathbb{R}^3}\int_{\mathbb{R}^3} \varphi_i(r_1)\frac{1}{|r_1-r_2|}\theta_j(r_2)\mathrm{d}r_1\mathrm{d}r_2 = \int_{\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}} p_i(r_2)\theta_j(r_2)\mathrm{d}r_2, \tag{4.10}$$

where the numerical support of θ_j is entirely within $\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}$ by the definition of J_{far}.

For analysis purposes, let $U(I_*^{\text{id}}|J_{\text{far}})$ be an ID approximation of $(I_*|J_{\text{far}})$. Each entry of $(I_*|J_{\text{far}})$ is then approximated as

$$(\varphi_i|\theta_j) \approx u_i^T(I_*^{\text{id}}|\theta_j), \quad \varphi_i \in I_*,\ \theta_j \in J_{\text{far}},$$

where u_i^T denotes the ith row of U. Substituting eq. (4.10) into the above equation gives

$$\int_{\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}} p_i(r_2)\theta_j(r_2)\mathrm{d}r_2 \approx \int_{\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}} \big(u_i^T P^{\text{id}}(r_2)\big)\theta_j(r_2)\mathrm{d}r_2, \tag{4.11}$$

where $P^{\text{id}}(y)$ denotes the vector of potentials $p_j(y)$ for all $\varphi_j \in I_*^{\text{id}}$. This rewriting shows that the ID approximation actually approximates each potential $p_i(y)$ in the domain $\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}$ by $u_i^T P^{\text{id}}(y)$ which is a linear combination of the potentials due to the distributions in I_*^{id}. Define the error of each approximation as

$$e_i(y) = p_i(y) - u_i^T P^{\text{id}}(y), \quad y \in \mathbb{R}^3\backslash\mathcal{B}^{\text{adj}},\ \varphi_i \in I_*. \tag{4.12}$$

Using Hölder's inequality, the elementwise error of the ID approximation in eq. (4.11) can be bounded as

$$\big|(\varphi_i|\theta_j) - u_i^T(I_*^{\text{id}}|\theta_j)\big| \leqslant \max_{y\in\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}} |e_i(y)| \int_{\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}} |\theta_j(r_2)|\mathrm{d}r_2. \tag{4.13}$$

From this analysis, a good ID approximation $U(I_*^{\text{id}}|J_{\text{far}})$ to $(I_*|J_{\text{far}})$ can be found by seeking U and I_*^{id} such that each defined $e_i(y)$ in eq. (4.13) is small in the domain $\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}$. In other words, we seek a subset of the potentials $\{p_j(y)\}_{\varphi_j\in I_*}$ whose linear combination can well approximate each $p_i(y)$ in the domain $\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}$.

To make the problem tractable, instead of considering the approximation to $p_i(y)$ at every $y \in \mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}$, we consider it at a finite set of proxy points Y_p that lie in $\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}$. An approximation to $p_i(y)$ can be accurate in $\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}$ as long as it is accurate at a small set of properly selected points Y_p. Such a choice of Y_p will be discussed in the next subsection.

Let $P(y)$ denote the vector of potentials $p_i(y)$ for all $\varphi_i \in I_*$. Assuming we have a set of proxy points Y_p, then the approximation to $P(y)$ can be computed by using SRRQR to compute the ID approximation,

$$P(Y_p) = \begin{pmatrix} p_1(Y_p)^T \\ p_2(Y_p)^T \\ \vdots \\ p_{|I_*|}(Y_p)^T \end{pmatrix} \approx U \begin{pmatrix} p_{i_1}(Y_p)^T \\ p_{i_2}(Y_p)^T \\ \vdots \\ p_{i_k}(Y_p)^T \end{pmatrix} = UP^{\text{id}}(Y_p). \tag{4.14}$$

The error in the ith row of this approximation is $p_i(Y_p)^T - u_i^T P^{\text{id}}(Y_p)$ and has its norm bounded by the error threshold specified for SRRQR. Thus, the ID approximation eq. (4.14) defines an approximation $u_i^T P^{\text{id}}(y)$ to each $p_i(y)$ with error $e_i(y)$ bounded at Y_p. Based on the previous analysis, the resulting U and I_*^{id} from eq. (4.14) can then be used for the ID approximation of $(I_*|J_{\text{far}})$. The remaining problem becomes how to select an effective but small set of proxy points Y_p.

4.4.3 Proxy point selection

We define $\mathcal{X}$ to be the smallest cubical domain that encloses the numerical support of I_*. In particular, $\mathcal{X}$ encloses $\mathcal{B}$ and shares the same center, as illustrated in Figure 4.4. Each potential $p_i(y)$ defined in eq. (4.9) can be further written as

$$p_i(y) = \int_{\mathcal{X}} \varphi_i(r) \frac{1}{|r-y|} \mathrm{d}r, \quad y \in \mathbb{R}^3 \backslash \mathcal{B}^{\text{adj}}.$$

From this formula, it can be noted that $p_i(y)$ is a harmonic function outside $\mathcal{X}$. As a linear combination of potentials $p_j(y)$ for all $\varphi_j(y) \in I_*$, $e_i(y)$ defined in eq. (4.12) with any u_i^T and I_*^{id} is also harmonic outside $\mathcal{X}$. By the maximum principle of harmonic functions, $e_i(y)$ satisfies $\max_{y\in\mathbb{R}^3\backslash\mathcal{X}} |e_i(y)| = \max_{y\in\partial\mathcal{X}} |e_i(y)|$ and thus

$$\max_{y\in\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}} |e_i(y)| = \begin{cases} \max_{y\in\mathcal{X}\backslash\mathcal{B}^{\text{adj}}} |e_i(y)| & \text{if } \mathcal{X} \supset \mathcal{B}^{\text{adj}} \\ \max_{y\in\partial\mathcal{B}^{\text{adj}}} |e_i(y)| & \text{if } \mathcal{X} \subset \mathcal{B}^{\text{adj}} \end{cases}. \tag{4.15}$$

As a result, it is sufficient to make $e_i(y)$ small in $\mathcal{X}\backslash\mathcal{B}^{\text{adj}}$ (or on $\partial\mathcal{B}^{\text{adj}}$) in order to make $e_i(y)$ small in $\mathbb{R}^3\backslash\mathcal{B}^{\text{adj}}$. This indicates that we only need to select the proxy points Y_p in $\mathcal{X}\backslash\mathcal{B}^{\text{adj}}$ (or on $\partial\mathcal{B}^{\text{adj}}$) for the ID approximation eq. (4.14).

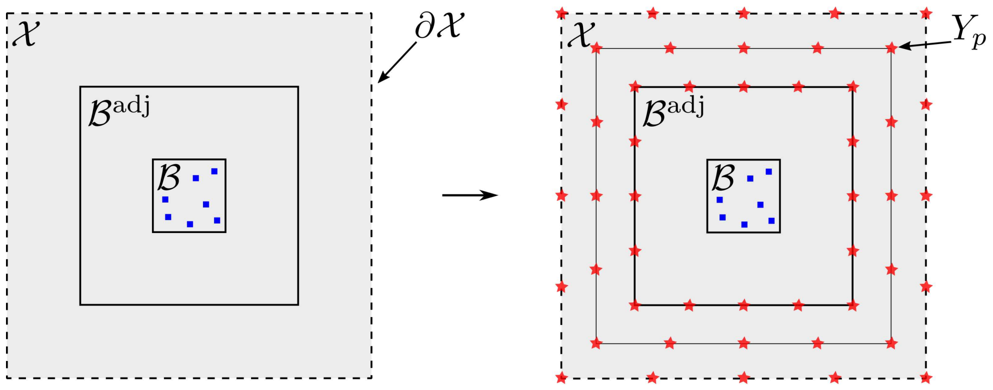


Figure 4.4: 2D illustration of the selection of proxy points Y_p in $\mathcal{X}\backslash\mathcal{B}^{\text{adj}}$.

For the case of point charges, $\mathcal{X}$ is within $\mathcal{B}^{\text{adj}}$ and the proxy points are selected on $\partial\mathcal{B}^{\text{adj}}$. The calculation of U and I_*^{id} for the ID approximation of $(I_*|J_{\text{far}})$ via eq. (4.14) is exactly the proxy surface method [10, 14]. In this case, Ref. [55] shows that the number of proxy points needed only depends on the ratio of the radius of $\mathcal{B}$ to that of $\mathcal{B}^{\text{adj}}$ and is not related to the absolute size of $\partial\mathcal{B}^{\text{adj}}$.

For GTF distributions in I_* (which have exponentially decaying tails), we continue to expect that only a constant number of proxy points is needed on $\partial\mathcal{X}$ (or on $\partial\mathcal{B}^{\text{adj}}$ when $\mathcal{X} \subset \mathcal{B}^{\text{adj}}$). With this idea, the proxy points are chosen heuristically as follows. If $\mathcal{X} \subset \mathcal{B}^{\text{adj}}$, we select a fixed number of points uniformly distributed on $\partial\mathcal{B}^{\text{adj}}$. Otherwise, we select multiple layers of evenly spaced cubic surfaces between and including $\partial\mathcal{B}^{\text{adj}}$ and $\partial\mathcal{X}$, with a fixed number of proxy points distributed uniformly on each cubic surface. The number of surfaces is proportional to the ratio of the distance between $\partial\mathcal{X}$ and $\partial\mathcal{B}^{\text{adj}}$ to the edge length of $\mathcal{B}$. Figure 4.4 gives a 2D example of the selected proxy points. Such a selection gives $O(1)$ number of proxy points and thus the approximated matrix $P(Y_p)$ in eq. (4.14) is also of $O(1)$ size.

To be consistent with ERI notation $(\cdot|\cdot)$, denote $P(Y_p)$ from eq. (4.14) as $(I_*|Y_p)$ where

$y_j \in Y_p$ stands for a point charge at y_j and thus

$$(\varphi_i|y_j) = (\varphi_i|\delta_{y_j}) = \int_{\mathbb{R}^3} \varphi_i(r)\frac{1}{|r-y_j|}\mathrm{d}r = p_i(y_j), \quad \varphi_i \in I_*,\ y_j \in Y_p.$$

It is important to note that each entry of $(I_*|Y_p)$ above is not an ERI—it is a nuclear attraction integral and is much cheaper to evaluate than an ERI [56].

4.4.4 Algorithm for computing the ID of $(I_*|J_*)$

From the above discussion, to construct the ID approximation

$$(I_*|J_*) \approx U(I_*^{\text{id}}|J_*), \tag{4.16}$$

it is sufficient to compute the components U and I_*^{id} such that

$$(I_*|J_{\text{near}}) \approx U(I_*^{\text{id}}|J_{\text{near}}) \qquad \text{and} \qquad (I_*|Y_p) \approx U(I_*^{\text{id}}|Y_p).$$

Using the idea of the randomized ID approximation method [57], the components U and I_*^{id} are computed as follows. First, generate two random matrices Ω_1 and Ω_2 of dimension $|J_{\text{near}}| \times |I_*|$ and $|Y_p| \times |I_*|$, respectively, whose entries follow the standard normal distribution. Multiply $(I_*|J_{\text{near}})$ with Ω_1 and $(I_*|Y_p)$ with Ω_2,

$$A_1 = (I_*|J_{\text{near}})\Omega_1 \quad \text{and} \quad A_1 = (I_*|Y_p)\Omega_2.$$

Then, normalize each column of A_1 and A_2 to have unit norm and denote the normalized matrices as $\tilde{A}_1$ and $\tilde{A}_2$. Lastly, compute U and I_*^{id} from the ID approximation,

$$[\tilde{A}_1,\ \tilde{A}_2] \approx U[\tilde{A}_1,\ \tilde{A}_2]_{I_*^{\text{id}},:}, \tag{4.17}$$

using SRRQR, where $[\tilde{A}_1,\ \tilde{A}_2]_{I_*^{\text{id}},:}$ denotes the subset of rows in $[\tilde{A}_1,\ \tilde{A}_2]$ computed by this

ID and $I_*^{\text{id}} \subset I_*$ is associated with the indices of this subset.

The reason for the normalization step is that $(I_*|J_{\text{near}})$ and $(I_*|Y_p)$ can have different number of columns and also their entries can be of different magnitudes. As a result, A_1 and A_2 can have their entries of different magnitudes. If directly computing an ID approximation of $[A_1,\ A_2]$, the obtained U and I_*^{id} could be biased and define a better ID approximation to the one of A_1 and A_2 that has larger entries.

This accelerated ID approximation of $(I_*|J_*)$ is summarized in Algorithm 6. Noting that $(I_*|J_{\text{near}})$ and $(I_*|Y_p)$ only have $O(1)$ number of columns, Algorithm 6 can be much faster than the purely algebraic ID approximation using SRRQR alone. More importantly, applying this compression method in $\mathcal{H}^2$ matrix construction can reduce the construction cost to nearly linear in the number of distributions.

Algorithm 6 Efficient ID approximation of $(I_*|J_*)$

```
**Input:** $I_*$, $J_*$, $\mathcal{B}^{\text{adj}}$, $\mathcal{X}$.
**Output:** $U$ and $I_*^{\text{id}}$ for an ID approximation $U(I_*^{\text{id}}|J_*)$ to $(I_*|J_*)$.
  • Split $J_*$ into $J_{\text{near}}$ and $J_{\text{far}}$.
  • Select proxy points $Y_p$ in $\mathcal{X}\backslash\mathcal{B}^{\text{adj}}$ (or on $\partial\mathcal{B}^{\text{adj}}$ when $\mathcal{X} \subset \mathcal{B}^{\text{adj}}$).
  • Generate random matrices $\Omega_1 \in \mathbb{R}^{|J_{\text{near}}|\times|I_*|}$ and $\Omega_1 \in \mathbb{R}^{|Y_p|\times|I_*|}$.
  • Calculate $A_1 = (I_*|J_{\text{near}})\Omega_1$ and $A_2 = (I_*|Y_p)\Omega_2$.
  • Normalize the columns of $A_1$ and $A_2$ to obtain $\tilde{A}_1$ and $\tilde{A}_2$.
  • Compute $U$ and $I_*^{\text{id}}$ from an ID approximation of $[\tilde{A}_1,\ \tilde{A}_2]$ using SRRQR.
```

We numerically demonstrate Algorithm 6 as follows. Consider a cube $\mathcal{B} = [-\frac{1}{2}L, \frac{1}{2}L]^3$ of edge length $L = 5$ and $\mathcal{B}^{\text{adj}} = [-\frac{3}{2}L, \frac{3}{2}L]^3$. Select 600 and 20000 GTF distributions of the form $\{(p/\pi)^{3/2}e^{-p|r-r_a|^2}\}$ with the same exponent p and different centers r_a randomly distributed in $\mathcal{B}$ and $[-\frac{11}{2}L, \frac{11}{2}L]^3\backslash\mathcal{B}^{\text{adj}}$, respectively. Denote the two sets of GTFs as I_* and J_*. To define $\mathcal{X}$, let the extent precision be $\tau = 10^{-10}$. Two exponents $p = 1$ and $p = 0.1$ are tested. Figure 4.5 shows the relative error of the low-rank approximation of $(I_*|J_*)$ calculated by Algorithm 6 for different choices of the rank. For both values of p, Algorithm 6 gives relative errors close to those of SVD and ID using SRRQR. Meanwhile, the intermediate approximation of $[\tilde{A}_1,\ \tilde{A}_2]$ in Algorithm 6 has slightly larger relative errors than the obtained ID approximation of $(I_*|J_*)$. The accuracy of the final approximation can be controlled by controlling the accuracy of the ID approximation eq. (4.17) computed by SRRQR.

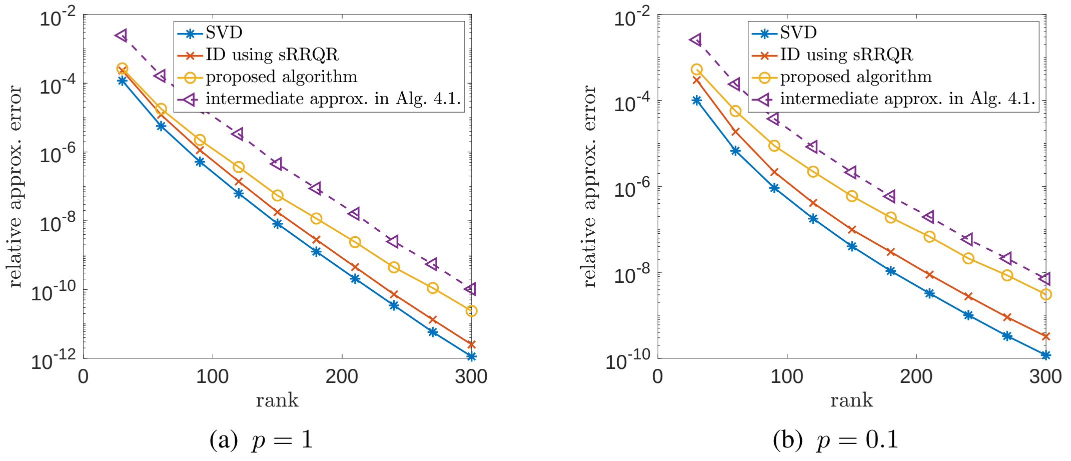


Figure 4.5: Relative error of the low-rank approximations of $(I_*|J_*)$ in the Frobenius norm. Three methods are used: SVD, ID using SRRQR, and Algorithm 6. In addition, the dashed lines show the relative error of the intermediate approximation eq. (4.17) for $[\tilde{A}_1,\ \tilde{A}_2]$. The test problem parameters are: (a) $p = 1$, $\lambda = 4.6$, $|J_{\text{near}}| = 1354$, $|J_{\text{far}}| = 18646$, and 1 layer of proxy points in Y_p with 384 points, (b) $p = 0.1$, $\lambda = 13.4$, $|J_{\text{near}}| = 8409$, $|J_{\text{far}}| = 11591$, and 3 layers of proxy points in Y_p with 1152 points.

4.4.5 Summary of the $\mathcal{H}^2$ method

We refer to the proposed Coulomb matrix construction method (Algorithm 7) as the $\mathcal{H}^2$ *method*. The method consists of two phases: (1) use the new compression technique to construct an $\mathcal{H}^2$ matrix representation of the ERI matrix $(I|I)$, and then (2) use the fast $\mathcal{H}^2$ matrix-vector multiplication algorithm to construct the Coulomb matrix.

Assuming that the ranks of the ID approximations in lines 4 and 10 of Algorithm 7 are bounded by a constant r (to be experimentally justified in Section 4.5), the first phase has $O(|I|r^2)$ computation cost and the second phase has $O(|I|r)$ computation cost. As mentioned in the introduction of the chapter, in self-consistent field iterations, a Coulomb matrix is constructed with different density matrices in each iteration while the ERI matrix is fixed. The relatively expensive cost for constructing the $\mathcal{H}^2$ matrix representation can be amortized over many matrix-vector multiplications.

Algorithm 7 Construct the Coulomb matrix by the $\mathcal{H}^2$ method

```
Input: distribution set $I$, density matrix $D$.
Output: $J = (I|I)D$.

Phase 1: Construct an $\mathcal{H}^2$ matrix representation of $(I|I)$
1: • Hierarchically partition $I$ into subsets $\{I_i\}$ with $L$ levels.
2: for node $i$ at level $L$ (the leaf level) do
3:     • Compute $U_i$ and $I_i^{\text{id}}$ from the ID approximation of $(I_i|J_i)$ in eq. (4.4) using
4:     Algorithm 6.
5: end for
6: for $k = L-1, L-2, \ldots, 3$ do
7:     for node $i$ at level $k$ do
8:         • Construct $\hat{I}_i$ and $\hat{J}_i$ according to eq. (4.6).
9:         • Compute $R_i$ and $I_i^{\text{id}}$ from the ID approximation of $(\hat{I}_i|\hat{J}_i)$ in eq. (4.7) using
10:        Algorithm 6.
11:    end for
12: end for
13: • (optional, see line 15) Construct inadmissible blocks $(I_i|I_j)$ for each inadmissible
    pair of nodes $i$ and $j$ at level $L$, and skeleton blocks $(I_i^{\text{id}}|I_j^{\text{id}})$ for each admissible pair
    of nodes $i$ and $j$ at the same level whose parent nodes are inadmissible.

Phase 2: Construct the Coulomb matrix
14: • Unfold the density matrix $D$ as a vector.
15: • Apply the $\mathcal{H}^2$ matrix-vector multiplication algorithm to construct $J = (I|I)D$. If
    line 13 is not applied, the inadmissible blocks and skeleton blocks are constructed
    when needed in the matrix-vector multiplication.
16: • Fold the vector $J$ as the computed Coulomb matrix.
```

The constructed $\mathcal{H}^2$ matrix representation has $O(|I|r)$ storage cost. The representation stores the following "necessary" components for each node i with a non-empty J_i: (1) I_i^{id}, (2) U_i if i is a leaf node, and (3) R_i if i is a non-leaf node. Further, the representation can either store the following components if line 13 of Algorithm 7 is applied, or compute them when they are needed in the second phase of Algorithm 7: (4) *inadmissible blocks* $(I_i|I_j)$ for each inadmissible pair of nodes i and j at the leaf level, and (5) *skeleton blocks* $(I_i^{\text{id}}|I_j^{\text{id}})$ for each admissible pair of nodes i and j at the same level whose parent nodes are inadmissible. These latter blocks are the intermediate blocks $B_{i,j}$ associated with the low-rank approximations eq. (4.8) to the admissible blocks used in the final $\mathcal{H}^2$ matrix representation.

As will be shown in the numerical tests, the storage cost for the inadmissible blocks and the skeleton blocks is much larger than the storage required for the other components in the $\mathcal{H}^2$ matrix representation. If these blocks are to be stored, they are best stored in dense matrix format, since they do not have enough sparsity to warrant storage in a sparse matrix format. In addition, storage of these blocks should be non-redundant, utilizing the 8-way symmetry present in the ERI tensor.

4.5 Numerical experiments

We test the $\mathcal{H}^2$ method and compare it to CFMM using two sets of molecular systems. The first is a set of linear alkanes of different sizes. The second is a set of truncated protein-ligand systems derived from the 1hsg system in the protein data bank. In this second set, each system consists of a ligand with its protein environment within a certain radius. Different radii give different sized systems. Such truncated systems are used in order to make protein-ligand simulations tractable. See Ref. [58] for more information on these systems.

These two sets of systems span an important determinant of CFMM and $\mathcal{H}^2$ method performance. The alkane systems are long and narrow while the 1hsg protein-ligand systems are globular. One may say that they have 1D and 3D "shapes", respectively. We thus expect a larger proportion of interactions that can be compressed in CFMM and the $\mathcal{H}^2$ method for the alkanes than for the 1hsg systems.

Prescreening

In practice, many rows and columns of the ERI matrix are numerically zero. Specifically, the row and column associated with a product $\phi_a\phi_b$ can be neglected if $|(\phi_a\phi_b|\phi_c\phi_d)| \leqslant \delta$ for any $\phi_c\phi_d$. A threshold of $\delta = 10^{-10}$ is used in our tests. Such numerically zero rows and columns can be identified efficiently as follows. From the Schwarz inequality

$|(\phi_a\phi_b|\phi_c\phi_d)| \leqslant \sqrt{(\phi_a\phi_b|\phi_a\phi_b)(\phi_c\phi_d|\phi_c\phi_d)}$, a product $\phi_a\phi_b$ and its corresponding row and column can be neglected if

$$\sqrt{(\phi_a\phi_b|\phi_a\phi_b)} \leqslant \frac{\delta}{\max_{c,d}\sqrt{(\phi_c\phi_d|\phi_c\phi_d)}}, \tag{4.18}$$

which only requires evaluating $(\phi_a\phi_b|\phi_a\phi_b)$ for each pair of basis functions. This process is called *prescreening* of basis function products [59]. Prescreening effectively reduces the dimension of the ERI matrix. We refer to this reduced dimension as the "effective dimension."

Basis set and contracted basis functions

The cc-pVDZ basis set is used for both sets of molecular systems. Like almost all Gaussian basis sets, the basis functions in this basis set are *contracted* GTFs (known linear combinations of GTFs), as mentioned in the introduction of the chapter. The product of two contracted GTF basis functions can be written as (neglecting contraction coefficients)

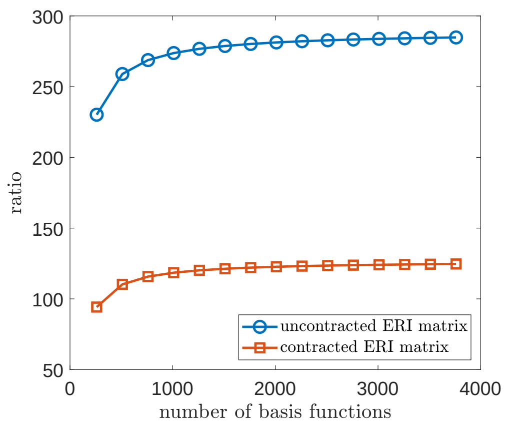
$$\phi_a\phi_b = \sum_{\chi_e\in[\phi_a]}\sum_{\chi_f\in[\phi_b]}\chi_e\chi_f,$$

where $[\phi_a]$ denotes the set of "primitive" GTFs that make up ϕ_a. Each ERI matrix entry $(\phi_a\phi_b|\phi_c\phi_d)$ can thus be written as the sum of ERIs with primitive GTFs as

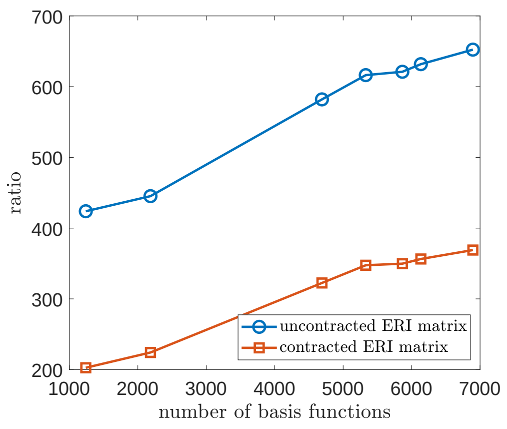
$$(\phi_a\phi_b|\phi_c\phi_d) = \sum_{\chi_e\in[\phi_a]}\sum_{\chi_f\in[\phi_b]}\sum_{\chi_g\in[\phi_c]}\sum_{\chi_h\in[\phi_d]}(\chi_e\chi_f|\chi_g\chi_h). \tag{4.19}$$

CFMM and the $\mathcal{H}^2$ method can be applied to either the original contracted ERI matrix $(\phi_a\phi_b|\phi_c\phi_d)$ or the uncontracted ERI matrix $(\chi_e\chi_f|\chi_g\chi_h)$. Compared to the contracted ERI matrix, the uncontracted ERI matrix has larger dimensions, i.e., more products in $\{\chi_e\chi_f\}$. However, there are also more products in $\{\chi_e\chi_f\}$ that can be prescreened. A more important advantage of using the uncontracted ERI matrix is that each $\chi_e\chi_f$ is a primitive GTF and thus its numerical support can be more precisely described by a ball than contracted GTFs, which improves the identification of well-separated interactions in CFMM and the identification of J_{near}, J_{far}, and $\mathcal{X}$ for Algorithm 6 in $\mathcal{H}^2$ matrix construction.

Figure 4.6 plots the ratio of the effective ERI matrix dimension to the number of basis functions for molecular systems of different sizes. The x-axis in this and other figures is the size of the molecular system in terms of the number of contracted basis functions $\{\phi_a\}$ (roughly 10 basis functions per atom). The figure shows that for our choice of $\delta = 10^{-10}$, the uncontracted ERI matrix is only about twice the dimension of the corresponding contracted ERI matrix. For increasing molecular system size, the effective ERI matrix dimension is expected to be asymptotically linear in the number of basis functions [56, 60]. This can be observed for the tested alkane molecules and is expected to be observed for larger 1hsg molecules.



(a) 1D alkane molecules



(b) 3D 1hsg molecules

Figure 4.6: Ratio of the effective ERI matrix dimension to the number of contracted basis functions for two types of molecules of different sizes. This ratio is plotted against the size of the molecular systems in terms of the number of contracted basis functions. Results for both uncontracted and contracted ERI matrices are shown.

In the following numerical tests, we apply CFMM and the $\mathcal{H}^2$ method to uncontracted and prescreened ERI matrices, i.e., the set of distributions I in Algorithm 7 contains the primitive basis function products obtained by prescreening and uncontraction. In practice, especially for basis sets with highly-contracted basis functions, it may be advantageous to work with contracted rather than uncontracted ERI matrices, which we intend to investigate in future work.

Method settings

In both the $\mathcal{H}^2$ method and CFMM, the extent precision is set to $\tau = 10^{-10}$. The hierarchical partitioning of the set of distributions is stopped when each finest box has less than 300 distributions or has edge length less than 1 Bohr.

For the selection of proxy points Y_p described in Section 4.4.3, when $\mathcal{X}$ is within $\mathcal{B}^{\text{adj}}$, only one cubical surface $\partial\mathcal{B}^{\text{adj}}$ is selected. Otherwise, we select cubical surfaces evenly spaced between and including $\partial\mathcal{B}^{\text{adj}}$ and $\partial\mathcal{X}$. The total number of these cubical surfaces is 3, 5, 7,..., when the ratio of the distance between $\partial\mathcal{X}$ and $\partial\mathcal{B}^{\text{adj}}$ to the edge length of $\mathcal{B}$ (when rounded up) equals 1, 2, 3,..., respectively. Figure 4.4 gives an example of three selected cubical surfaces when the ratio equals one. The number of proxy points selected on each cubical surface is 384, i.e., 8×8 uniform grid points on each face of the cubical surface.

4.5.1 Total number of direct interactions and rank of the low-rank approximations

In CFMM, the computation of the interactions that cannot be accelerated by multipole expansions dominates the total computation time. Similarly, in the $\mathcal{H}^2$ method, the computation of the interactions associated with inadmissible blocks dominates the computation time. In both cases, these interactions are evaluated directly. In this section, we compare the two methods in terms of the total number of these interactions. For convenience, we also refer to the interactions between two sets of distributions that are directly evaluated in CFMM as entries of an inadmissible block, and the interactions between two sets of distributions accelerated using multipole expansions in CFMM as entries of an admissible block. We follow Ref. [40] in defining admissible and inadmissible blocks in CFMM.

Figure 4.7 plots the total number of entries in the inadmissible and admissible blocks in the two methods. The main experimental result of this paper is that CFMM has approximately 5 times more inadmissible block entries (direct interactions) than the $\mathcal{H}^2$ method for the alkane molecules, and approximately 18 times more for the 1hsg molecules. Thus, the evaluation, multiplication, and storage of inadmissible blocks in CFMM are expected to be 5 and 18 times more expensive than in the $\mathcal{H}^2$ method for the two types of molecules, respectively. The result shows that the $\mathcal{H}^2$ method has even more advantage over CFMM on globular molecules like 1hsg. The number of admissible block entries is large, but these interactions are computed very efficiently (they are not computed explicitly in either method).

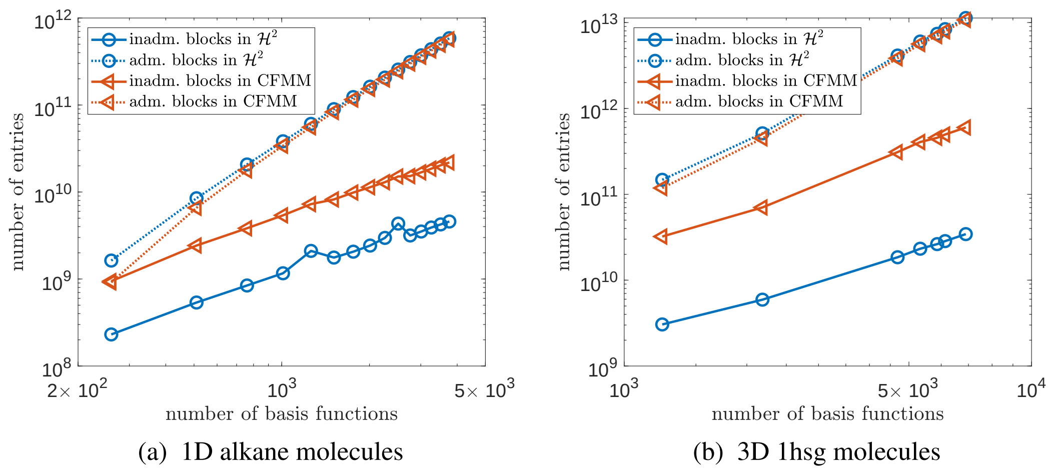


Figure 4.7: Total number of entries in the admissible and inadmissible blocks defined in CFMM and in the $\mathcal{H}^2$ method for two types of molecules of different sizes. Redundant interactions due to 8-way symmetry in the ERI tensor are not counted.

The maximum ranks of the low-rank approximations of all the admissible blocks in each constructed $\mathcal{H}^2$ matrix representation are shown in Figure 4.8. Here, the results are shown for two values of the relative error threshold, $\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-7}$, which is required for SRRQR in Algorithm 6. This threshold affects the approximation rank and storage required for the admissible blocks in the $\mathcal{H}^2$ matrix representation. The figure shows that the maximum rank is bounded for problems of different sizes. This justifies the observation in Section 4.2 that we can simply use the centers of distributions to decide whether an interaction can be compressed by a low-rank approximation. With bounded maximum rank, the $\mathcal{H}^2$ method (both phases in Algorithm 7) has computation cost and storage cost that are linear in the effective dimension of the ERI matrix, as explained in Section 4.4.5. The numerical results below in Sections 4.5.2 and 4.5.3 also confirm this linear scaling property.

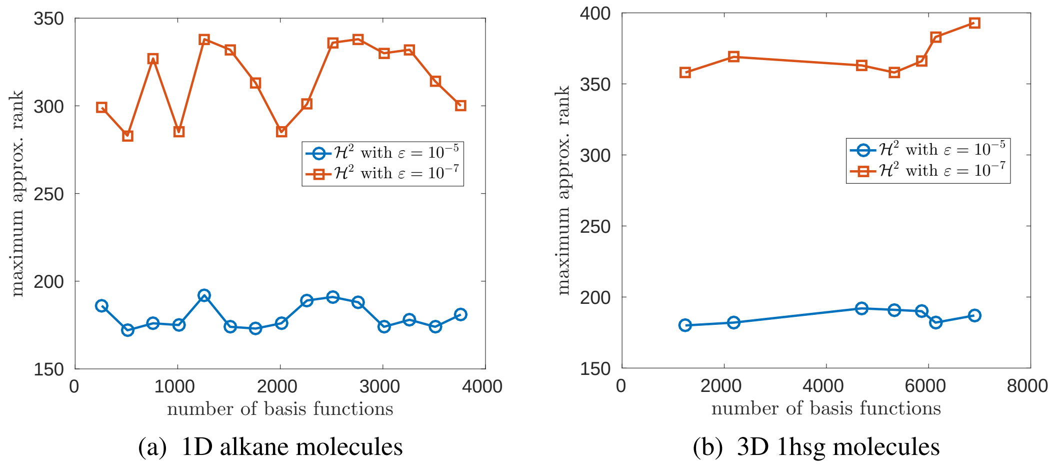
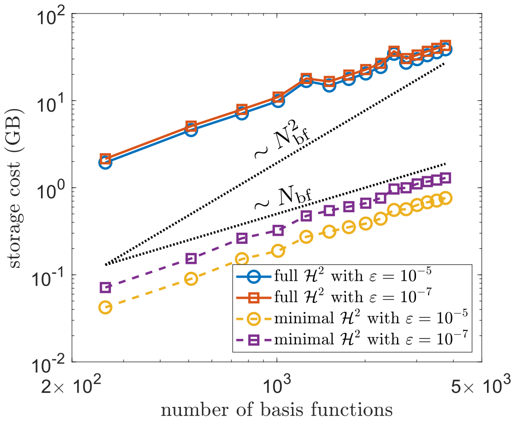


Figure 4.8: Maximum rank of the low-rank approximations of all the admissible blocks in the constructed $\mathcal{H}^2$ matrix representation for two types of molecules of different sizes.

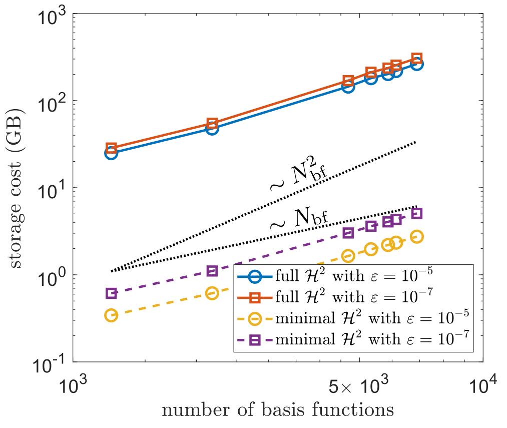
4.5.2 $\mathcal{H}^2$ matrix construction

The first phase of Algorithm 7 is the construction of the $\mathcal{H}^2$ matrix representation of an ERI matrix. In this subsection, the aim is to demonstrate this construction and show how the $\mathcal{H}^2$ matrix storage and construction execution time vary with increasing problem size. Again, we use two values of the relative error threshold ε for the ID approximations.

The storage cost for the $\mathcal{H}^2$ matrix representations is shown in Section 4.5.2. Results are shown for both the case when line 13 in Algorithm 7 is applied and the inadmissible blocks and skeleton blocks are stored (full $\mathcal{H}^2$), and the case when line 13 is not applied and these blocks are not stored (minimal $\mathcal{H}^2$). The high cost of storing the inadmissible and skeleton blocks is evident (although storage for the skeleton blocks is much less than the storage for the inadmissible blocks). The results show that the storage cost is almost linear in the number of basis functions for alkane molecules in either storage mode. The slightly superlinear cost for the 1hsg molecules is due to the slightly superlinear growth of the effective dimension of the ERI matrix with the number of basis functions, as shown earlier in Figure 4.6.



(a) 1D alkane molecules

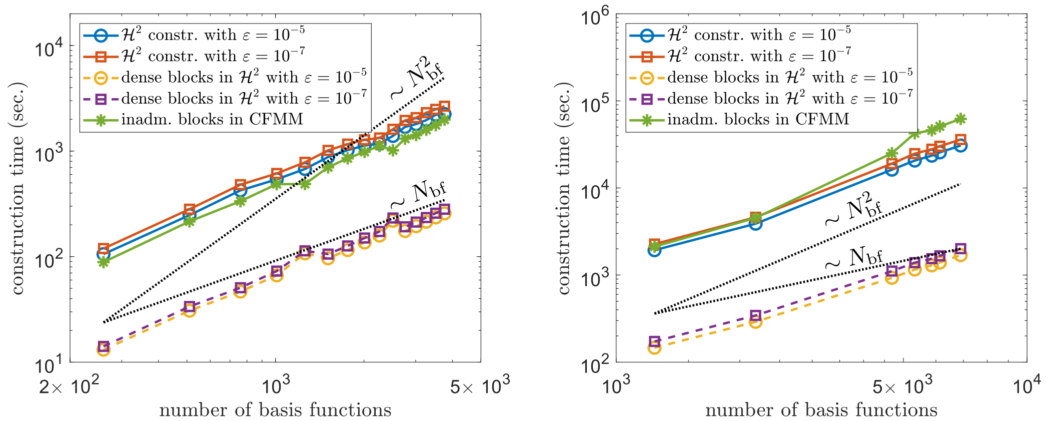


(b) 3D 1hsg molecules

Figure 4.9: Storage cost for $\mathcal{H}^2$ matrix representations of ERI matrices for two types of molecules of different sizes. "Full $\mathcal{H}^2$" refers to storing both the necessary components and the inadmissible and skeleton blocks according to Section 4.4.5. "Minimal $\mathcal{H}^2$" refers to storing only the necessary components. Reference lines for linear and quadratic scaling with the number of basis functions N_{bf} are also shown.

The timings for constructing the $\mathcal{H}^2$ matrix representations are shown in Section 4.5.2. These timings should only be regarded as an indication of relative trends, as our codes are implemented in Matlab. (ERIs and nuclear attraction integrals were computed analytically using recurrence relations implemented in the Simint package [61] using the C programming language.) For alkane molecules, the construction time is linear in the number of basis functions. For 1hsg molecules, the construction time is slightly superlinear, again because of the slightly superlinear growth of the effective dimension of the ERI matrix with the number of basis functions.

The timings for evaluating the inadmissible blocks in CFMM are also shown. As expected, these timings are much larger than for the $\mathcal{H}^2$ method, since there are far more entries in these blocks for CFMM as shown earlier in Figure 4.7. Meanwhile, $\mathcal{H}^2$ matrix



(a) 1D alkane molecules (b) 3D 1hsg molecules

Figure 4.10: Timings for constructing $\mathcal{H}^2$ matrix representations of ERI matrices for two types of molecules of different sizes. "$\mathcal{H}^2$ constr." refers to the timings for constructing the $\mathcal{H}^2$ matrix without evaluating the inadmissible and skeleton blocks, i.e., the first phase of Algorithm 7 without line 13. "Dense blocks in $\mathcal{H}^2$" refers to the timings for evaluating the inadmissible and skeleton blocks, i.e., line 13 of Algorithm 7. "Inadm. blocks in CFMM" refers to the timings for evaluating the inadmissible blocks in CFMM.

construction has similar execution time as evaluating the inadmissible blocks in CFMM. Since the cost for constructing the $\mathcal{H}^2$ matrix representations can be amortized by many matrix-vector multiplications (whose cost is to be shown next), the $\mathcal{H}^2$ method has better overall performance compared to CFMM.

4.5.3 Coulomb matrix construction

In the second phase of Algorithm 7, the Coulomb matrix for a given density matrix is constructed based on the $\mathcal{H}^2$ matrix representation of the ERI matrix constructed in the first phase. This second phase simply involves the fast $\mathcal{H}^2$ matrix-vector multiplication algorithm. The aim of this subsection is to demonstrate how the execution time of this $\mathcal{H}^2$ matrix-vector multiplication algorithm in different settings (storing the inadmissible and skeleton blocks or computing them dynamically) varies with increasing problem size. In comparison to CFMM, the improvement in execution time is directly related to the number of entries in the inadmissible blocks, as shown earlier in Figure 4.7. In this subsection, we also show the accuracy of the computed Coulomb matrix and demonstrate that this accuracy can be controlled by the SRRQR threshold, ε. For each molecule, we test Coulomb matrix construction with two types of density matrices: (a) randomly generated symmetric matrices whose entries follow the standard normal distribution, and (b) a density matrix obtained after 10 self-consistent field (SCF) iterations of the Hartree-Fock method.

Section 4.5.3 plots the relative errors in the constructed Coulomb matrices, where the "exact" Coulomb matrices are calculated directly. As before, we test two values of the relative error threshold ε used for the ID approximations. The results show that the relative error in the Coulomb matrices is consistent across the different types of molecules and molecule sizes. More specifically, the relative error is close to the value of the threshold ε for random density matrices, and is one order of magnitude smaller than the value of the threshold ε for density matrices generated by SCF iterations.

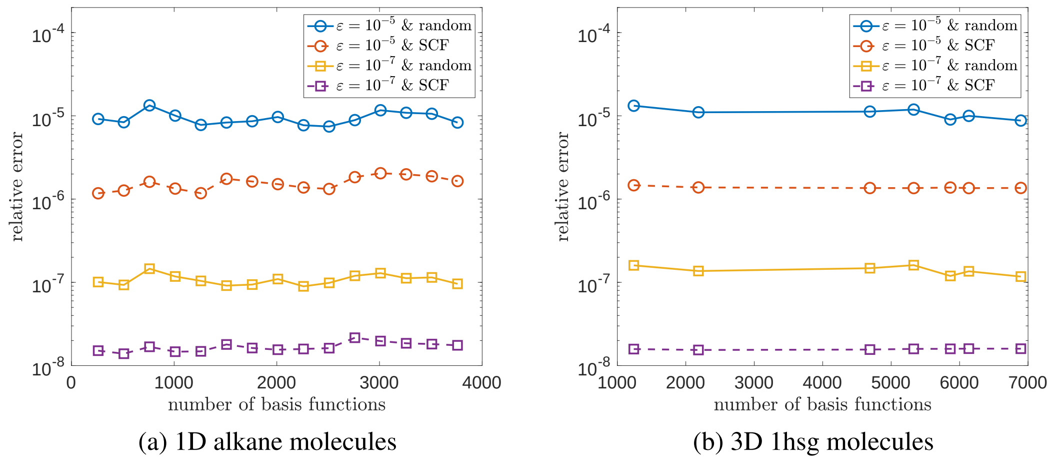
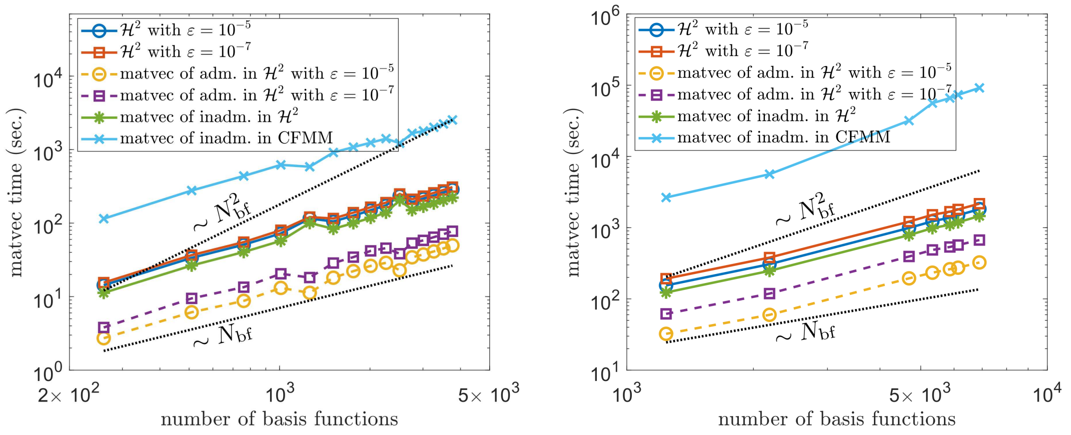


Figure 4.11: Relative error (in the Frobenius norm) of the Coulomb matrix constructed by the $\mathcal{H}^2$ method for two types of molecules of different sizes. For random density matrices, the results are the average of 5 independent tests.

Figure 4.12 plots the timings for the $\mathcal{H}^2$ matrix-vector multiplication used to construct the Coulomb matrix for the two types of molecules of different sizes. Here, the ERIs in the inadmissible and skeleton blocks are dynamically calculated when needed during the matrix-vector multiplication. Just like for $\mathcal{H}^2$ matrix construction, the matrix-vector multiplication for a matrix in $\mathcal{H}^2$ format is almost linear in the effective dimension of the ERI matrix (which is slightly superlinear in the number of basis function in the case of 1hsg).

The figure also shows the timings broken down into the portion for multiplying by admissible blocks and by inadmissible blocks. It is evident that forming and multiplying by the inadmissible blocks, i.e., computing the direct interactions, is the bottleneck, even after the reduction in the total size of these blocks due to the $\mathcal{H}^2$ method compared to CFMM.



(a) 1D alkane molecules (b) 3D 1hsg molecules

Figure 4.12: Timings for constructing Coulomb matrices by the $\mathcal{H}^2$ method where inadmissible and skeleton blocks are dynamically calculated when needed (the second phase of Algorithm 7 with line 13 not applied). The timing is also broken down into the portion for multiplying by admissible blocks and by inadmissible blocks. For comparison, the timings for the multiplications with inadmissible blocks in CFMM is also shown. The timings are the average of 5 independent tests.

Figure 4.13 again plots the timings for Coulomb matrix construction for molecules of different sizes, but this time we assume that the inadmissible and skeleton blocks have been precomputed and stored. Due to memory limitations, only the alkane molecules are tested. In this case, the multiplication of admissible blocks and that of inadmissible blocks require a similar amount of time. With the $\mathcal{H}^2$ method, multiplying by the inadmissible blocks when these blocks have been precomputed is no longer a clear bottleneck.

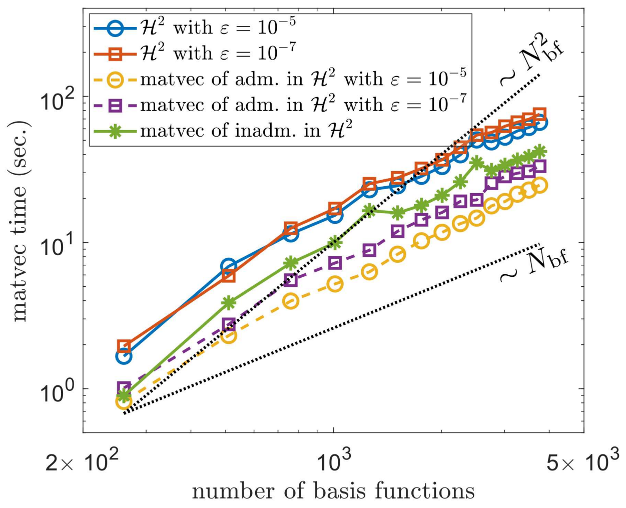


Figure 4.13: Timings for constructing Coulomb matrices for alkane molecules by the $\mathcal{H}^2$ method where inadmissible and skeleton blocks have been precomputed and stored (the second phase of Algorithm 7 with line 13 applied). The timings are also broken down into the portion for multiplying by admissible blocks and by inadmissible blocks. The timings are the average of 5 independent tests.

4.6 Conclusion

In this chapter, a new variant of the proxy point method is proposed to efficiently compress the interactions between continuous charge distributions. Using this method, an $\mathcal{H}^2$ matrix representation of the ERI matrix is constructed, which is then used to construct the Coulomb matrix. This overall approach can also be viewed as extending the capability of $\mathcal{H}^2$ matrices to represent the interactions between continuous charge distributions, at least for charge distributions from Gaussian basis sets.

Our approach to constructing the Coulomb matrix has cost that appears to be nearly linear in the effective ERI matrix dimension. The effective ERI matrix dimension has been argued to be asymptotically linear (rather than quadratic) with the number of basis functions [56]. More importantly, compared to CFMM, far fewer interactions need to be directly computed. The promise of this approach is demonstrated using a common Gaussian basis set on alkane and globular molecules of different sizes. In general, basis sets using compactly-supported or fast-decaying basis functions could be used.

The new proxy point method and the $\mathcal{H}^2$ matrix approach can be extended to accelerate the tensor contractions in density fitting [42, 62, 43, 44] and, in general, quantum chemical methods that already use CFMM. In particular, they could be extended to calculate Coulomb energy gradients [63, 64, 65] and potentials for periodic systems [66, 67].

To further accelerate the proposed variant of the proxy point method and thus to reduce the $\mathcal{H}^2$ matrix construction cost, it is possible to apply heuristic algebraic compression methods, such as sampling-based methods [68, 28], to accelerate the intermediate ID approximation in Algorithm 6. Finally, it is also possible to use an even weaker admissibility condition than that used in this chapter for $\mathcal{H}^2$ matrix representations to try to compress even more interactions in the ERI matrix.

CHAPTER 5
EQUIVALENCE AND COMPARISON BETWEEN FMM AND $\mathcal{H}^2$ MATRICES

The original fast multipole method (FMM) [17, 18] was designed to accelerate the evaluation of Coulombic or gravitational interactions between particles. Essentially, FMM calculates a matrix-vector multiplication by a kernel matrix defined by the 3D Laplace kernel (or its gradient) with linear complexity. Later, variants of FMM such as the black-box FMM [20] and the kernel independent FMM [21, 25] were proposed to adapt FMM to particle interactions defined by kernel functions that are more general than the Laplace kernel. The key component in FMM is the multipole expansion of the Laplace kernel, while in FMM variants the multipole expansion is replaced by other general degenerate expansions such as polynomial interpolations [20] and pseudoskeleton approximations [21, 25].

It is already folklore in the fast algorithm community that FMM is algebraically equivalent to multiplying by a matrix in a specific $\mathcal{H}^2$ matrix format. In this thesis, we formally show this equivalence. Further, we also provide analytic and numerical comparisons between FMM and $\mathcal{H}^2$ matrices. We test two state-of-the-arts FMM libraries and our own $\mathcal{H}^2$ matrix library that is based on the proxy point method. Numerical experiments demonstrate the relative advantages and disadvantages between the two methods.

The rest of this chapter is organized as follows.

- Section 5.1 explains the exact equivalence between FMM and $\mathcal{H}^2$ matrices.
- Section 5.2 discusses the analytic comparisons between FMM and $\mathcal{H}^2$ matrices.
- Section 5.3 discusses the numerical comparisons between FMM and $\mathcal{H}^2$ matrices.

5.1 Exact equivalence between FMM and $\mathcal{H}^2$ matrices

Consider a kernel function $K(x, y)$ and a set of points X in a low-dimensional space. Let q be a vector of weights associated with the points in X. Forming the kernel summation,

$$b_i = \sum_{x_j \in X} K(x_i, x_j) q_j, \quad x_i \in X,$$

is nothing other than computing $b = K(X, X)q$ where $K(X, X)$ is a kernel matrix consisting of $K(x_i, x_j)$ with all pairs of x_i and x_j in X. In the following discussion, the vector q is referred to as a vector of *charges* and $K(X, X)q$ is a vector of the induced *potentials*. For a kernel block $K(X_i, X_j)$, X_j is the set of *source points* and X_i is the set of *target points*. In the case of $K(X, X)$, both the source points and the target points are X.

FMM shares the same hierarchical partitioning of X and $K(X, X)$ as an $\mathcal{H}^2$ matrix representation of $K(X, X)$. An admissible (inadmissible) block in the $\mathcal{H}^2$ matrix corresponds to a set of *far field* (*near field*) interactions in FMM. Also, FMM splits the multiplication $K(X, X)q$ into the multiplications of the same sets of blocks as in the $\mathcal{H}^2$ matrix, i.e., the inadmissible, admissible, and partially admissible blocks denoted by the sets $\mathcal{D}$, $\mathcal{A}$, and $\mathcal{A}_p$, respectively.

Let $\mathcal{T}$ be the partition tree associated with FMM for $K(X, X)$. For each node $i \in \mathcal{T}$, let q_i and b_i denote the subvectors of q and b, respectively, whose entry indices are associated with X_i in X. FMM traverses all the three sets of blocks $\mathcal{D}$, $\mathcal{A}$, and $\mathcal{A}_p$ and accumulates the corresponding multiplications, i.e.,

$$b_i = b_i + K(X_i, X_j) q_j, \quad (i, j) \in \mathcal{D} \cup \mathcal{A} \cup \mathcal{A}_p,$$

with $K(X_i, X_j)q_j$ computed directly or computed indirectly but faster based on the multipole expansion technique.

5.1.1 Multiplication with inadmissible blocks

Multiplying an inadmissible block $K(X_i, X_j)$ with $(i, j) \in \mathcal{D}$ by q_j in FMM is the same as in an $\mathcal{H}^2$ matrix. In FMM, such a block $K(X_i, X_j)$ is referred to as a *source-to-target linear operator* (S2T) since it maps the charges q_j at X_j to their induced potentials $K(X_i, X_j)q_j$ at X_i. This is the first analogy between FMM and $\mathcal{H}^2$ matrices:

- For each pair of nodes $(i, j) \in \mathcal{D}$, $K(X_i, X_j)$ in an $\mathcal{H}^2$ matrix corresponds to the S2T operator mapping from box j to box i.

5.1.2 Multiplication with admissible blocks

Consider an admissible block $K(X_i, X_j)$ with $(i, j) \in \mathcal{A}$. For simplicity, we assume that nodes i and j are at the same level and their children $\{i_1, i_2, \ldots, i_s\}$ and $\{j_1, j_2, \ldots, j_s\}$ are all leaf nodes. Figure 5.1 plots such a pair of (i, j) in a 1D example and illustrates the computation of $K(X_i, X_j)q_j$ by FMM. We note that $K(z, X_j)q_j$ as a function of point z in the space is the potential field induced by charges q_j at X_j. Thus, $K(X_i, X_j)q_j$ can be viewed as the evaluation of the potential field $K(z, X_j)q_j$ at $z = X_i$.

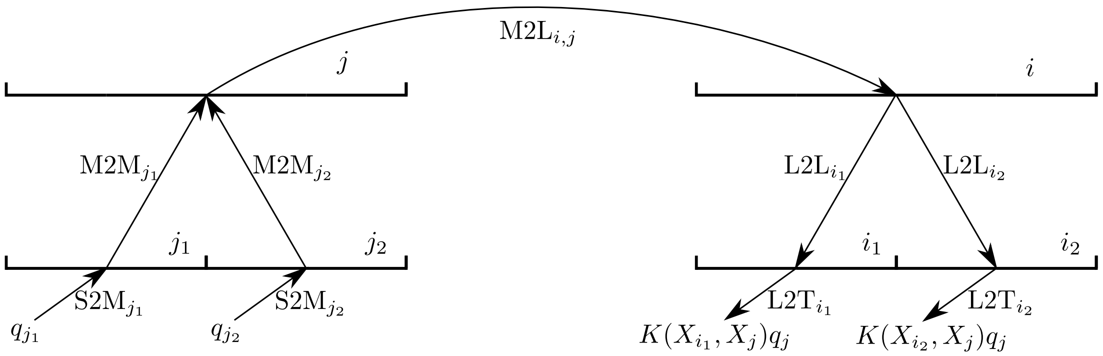


Figure 5.1: Illustration of FMM computing $K(X_i, X_j)q_j$ with $(i, j) \in \mathcal{A}$ in a 1D example where j has its children j_1 and j_2 to be leaves and i has its children i_1 and i_2 to be leaves.

The basic idea of FMM is to approximate the overall potential field $K(z, X_j)q_j$ for z in each child box domain i_a of i using *multipole expansions* and *local expansions* and then evaluate the approximate potential at $z = X_i$ to compute $K(X_i, X_j)q_j$. Given a box domain

$\mathcal{X}$ and its far field $\mathcal{Y}$, a multipole expansion **centered at $\mathcal{X}$** refers to a linear combination of specific basis functions $\{\phi_k(z - z_{\mathcal{X}})\}$ **defined in $\mathcal{Y}$**, i.e.,

$$f(z) = M_1\phi_1(z - z_{\mathcal{X}}) + M_2\phi_2(z - z_{\mathcal{X}}) + \ldots + M_r\phi_r(z - z_{\mathcal{X}}), \quad z \in \mathcal{Y} \tag{5.1}$$

where $z_{\mathcal{X}}$ denotes the center of $\mathcal{X}$, r is the degree of the expansion, $(M_k)_{k=1}^r$ are the coefficients of the expansion (referred to as the moments). Multipole expansions are used in FMM to approximate the potential field in $\mathcal{Y}$ induced by any charges in $\mathcal{X}$ based on an existing degenerate approximation of $K(z, w)$ in $\mathcal{Y} \times \mathcal{X}$ as

$$K(z, w) \approx \lambda_1(w - z_{\mathcal{X}})\phi_1(z - z_{\mathcal{X}}) + \ldots + \lambda_r(w - z_{\mathcal{X}})\phi_r(z - z_{\mathcal{X}}), \quad (z, w) \in \mathcal{Y} \times \mathcal{X} \tag{5.2}$$

where $\{\lambda_k(w - z_{\mathcal{X}})\}$ are known functions. For any n_* charges q_* at $X_* \subset \mathcal{X}$, their induced potential field $K(z, X_*)q_*$ in $\mathcal{Y}$ can be approximated by a multipole expansion as in eq. (5.1) using the degenerate approximation eq. (5.2) as

$$\begin{aligned}
&K(z, X_*)q_* \\
&\approx \sum_{w_i \in X_*} \left(\lambda_1(w_i - z_{\mathcal{X}})\phi_1(z - z_{\mathcal{X}}) + \ldots + \lambda_r(w_i - z_{\mathcal{X}})\phi_r(z - z_{\mathcal{X}})\right) q_i \\
&= \left(\sum_{w_i \in X_*} \lambda_1(w_i - z_{\mathcal{X}})q_i\right) \phi_1(z - z_{\mathcal{X}}) + \ldots + \left(\sum_{w_i \in X_*} \lambda_r(w_i - z_{\mathcal{X}})q_i\right) \phi_r(z - z_{\mathcal{X}}) \\
&= M_1\phi_1(z - z_{\mathcal{X}}) + M_2\phi_2(z - z_{\mathcal{X}}) + \ldots + M_r\phi_r(z - z_{\mathcal{X}})
\end{aligned}$$

where the moments $(M_k)_{k=1}^r$ are computed as

$$\begin{pmatrix} M_1 \\ M_2 \\ \vdots \\ M_r \end{pmatrix} = \begin{pmatrix} \lambda_1(w_1 - z_{\mathcal{X}}) & \ldots & \lambda_1(w_{n_*} - z_{\mathcal{X}}) \\ \lambda_2(w_1 - z_{\mathcal{X}}) & \ldots & \lambda_2(w_{n_*} - z_{\mathcal{X}}) \\ \vdots & & \vdots \\ \lambda_r(w_1 - z_{\mathcal{X}}) & \ldots & \lambda_r(w_{n_*} - z_{\mathcal{X}}) \end{pmatrix} q_*, \quad X_* = \{w_1, \ldots, w_{n_*}\}. \tag{5.3}$$

Similarly, a local expansion **centered at $\boldsymbol{\mathcal{X}}$** refers to a linear combination of another set of basis functions $\{\psi_i(z - z_{\mathcal{X}})\}$ **defined in $\boldsymbol{\mathcal{X}}$**. Opposite to multipole expansions, local expansions are used to approximate the potential field in $\mathcal{X}$ induced by charges in $\mathcal{Y}$ based on an existing degenerate approximation of $K(z, w)$ in $\mathcal{X} \times \mathcal{Y}$. In the original FMM [18] for $K(x, y) = 1/|x - y|$ in 3D, $\{\phi_i(z)\}$ and $\{\psi_i(z)\}$ are selected as multipoles and solid harmonics, respectively. Other basis functions have also been used in FMM variants.

To begin with, FMM first applies a *source-to-multipole linear operator* (S2M) to q_{j_a} for each child j_a of j. This S2M$_{j_a}$ operator is exactly the matrix in eq. (5.3) associated with box j_a and the points in X_{j_a}, mapping q_{j_a} to the moments $m_{j_a} = (M_k^{j_a})_{k=1}^r = \text{S2M}_{j_a}(q_{j_a})$ of a multipole expansion centered at **box $\boldsymbol{j_a}$** that approximates the potential field $K(z, X_{j_a})q_{j_a}$ in **the far field of box $\boldsymbol{j_a}$**, i.e.,

$$K(z, X_{j_a})q_{j_a} \approx \sum_{k=1}^{r} M_k^{j_a} \phi_k(z - z_{j_a}), \quad z \text{ in the far field of box } j_a,$$

where z_{j_a} denotes the center of box j_a.

For each child box j_a of j, FMM then constructs a new multipole expansion centered at **box $\boldsymbol{j}$** with moments $m_{j,j_a} = (M_k^{j,j_a})_{k=1}^r$ to approximate $K(z, X_{j_a})q_{j_a}$ in the **far field of box $\boldsymbol{j}$** via approximating the computed multipole expansion centered at box j_a, i.e.,

$$K(z, X_{j_a})q_{j_a} \approx \sum_{k=1}^{r} M_k^{j_a} \phi_k(z - z_{j_a}) \approx \sum_{k=1}^{r} M_k^{j,j_a} \phi_k(z - z_j), \quad z \text{ in the far field of box } j,$$

where z_j denotes the center of box j. Specifically, FMM applies a *multipole-to-multipole linear operator* (M2M) to the moments m_{j_a} for each child box j_a. This M2M$_{j_a}$ operator maps m_{j_a} to the moments m_{j,j_a} of this new multipole expansion centered at box j. Adding up the moments m_{j,j_a} obtained from all the children j_a,

$$m_j = m_{j,j_1} + \ldots + m_{j,j_s} = \text{M2M}_{j_1}(m_{j_1}) + \ldots + \text{M2M}_{j_s}(m_{j_s}),$$

gives the moments $m_j = (M_k^j)_{k=1}^r$ of a multipole expansion centered at **box $\boldsymbol{j}$** that approximates the potential field $K(z, X_j)q_j$ in **the far field of box $\boldsymbol{j}$**,

$$K(z, X_j)q_j = \sum_{j_a} K(z, X_{j_a})q_{j_a} \approx \sum_{k=1}^{r} M_k^j \phi_k(z - z_j), \quad z \text{ in the far field of box } j. \tag{5.4}$$

FMM next approximates the potential field $K(z, X_j)q_j$ in **box $\boldsymbol{i}$** by a local expansion centered at **box $\boldsymbol{i}$** (note that the points in X_j are in the far field of box i). Recall that box i is in the far field of box j. The potential field $K(z, X_j)q_j$ in box i has been well approximated by the computed multipole expansion with moments m_j centered at box j (see eq. (5.4)). Thus, such a local expansion can be constructed by approximating this computed multipole expansion instead. Specifically, FMM applies a *multipole-to-local linear operator* (M2L) to the moments m_j. This M2L$_{i,j}$ operator exactly maps m_j to the moments $l_i = (L_k^i)_{k=1}^r$ of a local expansion centered at **box $\boldsymbol{i}$** satisfying

$$K(z, X_j)q_j \approx \sum_{k=1}^{r} M_k^j \phi_k(z - z_j) \approx \sum_{k=1}^{r} L_k^i \psi_k(z - z_i), \quad z \text{ in box } i, \tag{5.5}$$

where z_i denotes the center of box i.

For each child box i_a of i, FMM then constructs a new local expansion centered at **box $\boldsymbol{i_a}$** with moments l_{i_a} to approximate $K(z, X_j)q_j$ in **box $\boldsymbol{i_a}$** via approximating the computed local expansion centered at box i, i.e.,

$$K(z, X_j)q_j \approx \sum_{k=1}^{r} L_k^i \psi_k(z - z_i) \approx \sum_{k=1}^{r} L_k^{i_a} \psi_k(z - z_{i_a}), \quad z \text{ in box } i_a. \tag{5.6}$$

Specifically, FMM applies a *local-to-local linear operator* (L2L) to the moments l_i for each child box i_a of i. This L2L$_{i_a}$ operator maps l_i to the moments l_{i_a} of this new local expansion centered at box i_a.

Now, the potential field $K(z, X_j)q_j$ in each child **box $\boldsymbol{i_a}$** of i is approximated by the last local expansion computed above centered at box i_a with moments l_{i_a}. Lastly, for all the points z_* in X_{i_a}, $K(z_*, X_j)q_j$ is approximately computed by plugging z_* into this local expansion, i.e., eq. (5.6). This computation is equivalent to applying a *local-to-target linear operator* (L2T), i.e., the matrix $(\psi_k(z - z_{i_a}))_{z \in X_{i_a}; k \in \{1,\dots,r\}}$, to l_{i_a}, which gives the final result $K(X_{i_a}, X_j)q_j \approx L2T_{i_a}(l_{i_a})$ for each child i_a.

We note that the above S2M, M2M, M2L, L2L, and L2T operators (referred to as *translation operators* in FMM) are all analytically formulated based on the box locations and the basis functions used in the multipole and local expansions. The exact definitions of these translation operators can be found in Ref [18]. S2M and L2T operators also depend on the points in the corresponding boxes and have been explicitly illustrated in the above discussion. Writing all these translation operators in matrix form, the above computation of $K(X_i, X_j)q_j$ by FMM can be represented as,

$$\begin{aligned}
K(X_i, X_j)q_j \approx &\begin{pmatrix} \mathrm{L2T}_{i_1} & & \\ & \ddots & \\ & & \mathrm{L2T}_{i_s} \end{pmatrix} \begin{pmatrix} \mathrm{L2L}_{j_1} \\ \vdots \\ \mathrm{L2L}_{j_s} \end{pmatrix} \mathrm{M2L}_{i,j} \times \\
&\begin{pmatrix} \mathrm{M2M}_{j_1} & \dots & \mathrm{M2M}_{j_s} \end{pmatrix} \begin{pmatrix} \mathrm{S2M}_{j_1} & & \\ & \ddots & \\ & & \mathrm{S2M}_{j_s} \end{pmatrix} \begin{pmatrix} q_{j_1} \\ \vdots \\ q_{j_s} \end{pmatrix} \\
= &\,\mathbf{L2T}_i \times \mathbf{L2L}_i \times \mathrm{M2L}_{i,j} \times \mathbf{M2M}_j \times \mathbf{S2M}_j \times q_j, \qquad (5.7)
\end{aligned}$$

where $\mathbf{L2T}_i$, $\mathbf{L2L}_i$, $\mathbf{M2M}_j$, and $\mathbf{S2M}_j$ denote the corresponding bracketed matrices. For example, applying $\mathbf{S2M}_j$ to q_j gives the concatenated moments $(m_{j_1}, m_{j_2}, \dots, m_{j_s})^T$ in child boxes of j, then applying $\mathbf{M2M}_j$ to $(\mathbf{S2M}_j q_j)$ gives the moments m_j in box j, and etc. In comparison, the computation of $K(X_i, X_j)q_j$ by an $\mathcal{H}^2$ matrix is represented as

$$K(X_i, X_j)q_j \approx \begin{pmatrix} U_{i_1} & & \\ & \ddots & \\ & & U_{i_s} \end{pmatrix} R_i B_{i,j} S_j^T \begin{pmatrix} V_{j_1}^T & & \\ & \ddots & \\ & & V_{j_s}^T \end{pmatrix} \begin{pmatrix} q_{j_1} \\ \vdots \\ q_{j_s} \end{pmatrix}. \tag{5.8}$$

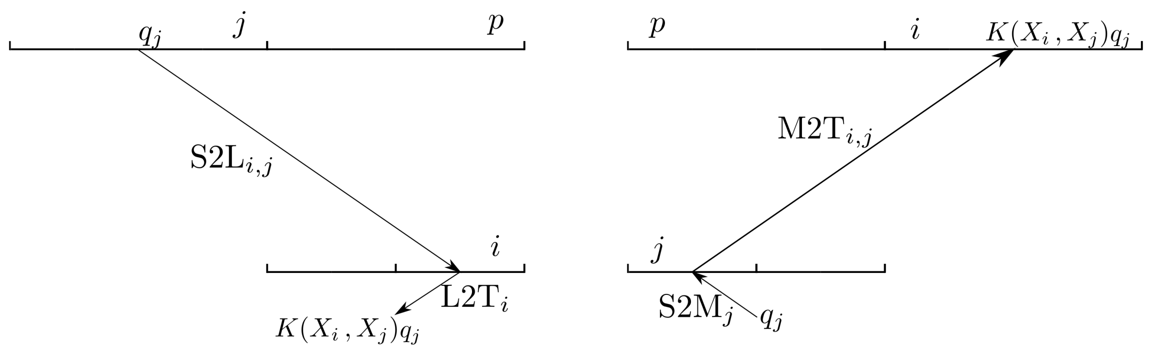
Although the admissible block $K(X_i, X_j)$ is assumed to have the children of i and j to be leaf nodes, the above two representations, eqs. (5.7) and (5.8), of $K(X_i, X_j)q_j$ can be easily extended to general admissible blocks (with more or fewer M2M and L2L operators or transfer matrices R_i and S_j). Comparing the two representations, the analogies between FMM and $\mathcal{H}^2$ matrices related to admissible blocks $K(X_i, X_j)$ with $(i, j) \in \mathcal{A}$ are obtained as follows.

- For each leaf node $i \in \mathcal{T}$, the transpose of the uniform row basis matrix V_i corresponds to the S2M operator for box i. The uniform column basis matrix U_i corresponds to the L2T operator for box i.
- For each non-leaf node $i \in \mathcal{T}$, the transpose of the transfer matrix S_i corresponds to the concatenated M2M operator, $\mathbf{M2M}_i$, for box i. The transfer matrix R_i corresponds to the concatenated L2L operator, $\mathbf{L2L}_i$, for box i.
- For each pair of nodes $(i, j) \in \mathcal{A}$, the intermediate matrix $B_{i,j}$ corresponds to the M2L operator mapping from box j to box i.

5.1.3 Multiplication with partially admissible blocks

Consider a partially admissible block $K(X_i, X_j)$ with $(i, j) \in \mathcal{A}_p$. There are two different cases: $j \in \mathcal{F}_i^2$ and $i \in \mathcal{F}_j^2$. Figure 5.2 illustrates the computation of $K(X_i, X_j)q_j$ by FMM in a 1D example for the two cases.

In the case of $j \in \mathcal{F}_i^2$, recall that box j is in the far field of box i but box i is not in the far-field of box j. The multipole expansion centered at box j computed by the S2M



(a) the case with $j \in \mathcal{F}_i^2$ (b) the case with $i \in \mathcal{F}_j^2$

Figure 5.2: Illustration of FMM computing $K(X_i, X_j)q_j$ with $(i, j) \in \mathcal{A}_p$ in a 1D example. The computation varies in two situations (a) $j \in \mathcal{F}_i^2$ and (b) $i \in \mathcal{F}_i^2$.

operator above cannot accurately approximate the potential field $K(z, X_j)q_j$ in box i, since this approximation is only accurate in the far field of box j. Instead, FMM directly applies a *source-to-local linear operator* (S2L) to charges q_j at X_j that maps q_j to the moments l_i of a local expansion centered at box i. This local expansion approximates the potential field $K(z, X_j)q_j$ in box i. The remaining steps are the same as for admissible blocks, including L2L and L2T operators applied to box i.

In an $\mathcal{H}^2$ matrix, $K(X_i, X_j)$ is approximated by $U_i B_{i,j}$ and $K(X_i, X_j)q_j$ is computed in two steps: $B_{i,j}q_j$ and $U_i(B_{i,j}q_j)$. Since U_i corresponds to the composition of the L2L and L2T operators applied to box i, it can thus be noted that

- For each pair of nodes $(i, j) \in \mathcal{A}_p$ with $j \in \mathcal{F}_i^2$, the intermediate matrix $B_{i,j}$ corresponds to the S2L operator mapping from box j to box i.

In the case of $i \in \mathcal{F}_j^2$, recall that box j is not in the far-field of box i. The local expansion centered at box i computed by applying M2L$_{i,j}$ to box j cannot accurately approximate $K(z, X_j)q_j$ in box i, since such a local expansion is accurate only when box j is in the far field of box i. Instead, FMM directly evaluates the approximate potential at all the points $z_* \in X_i$ by plugging z_* into the multipole expansion centered at box j which is computed by S2M and M2M operators, i.e., the expansion in the middle of eq. (5.5). This evaluation corresponds to the application of a *multipole-to-target linear operator* (M2T) to the moments m_j at box j which maps m_j to the final approximate potential at X_i.

In an $\mathcal{H}^2$ matrix, $K(X_i, X_j)$ is approximated by $B_{i,j} V_j^T$ and $K(X_i, X_j) q_j$ is computed in two steps: $V_j^T q_j$ and $B_{i,j}(V_j^T q_j)$. Since V_j^T corresponds to the composition of the S2M and M2M operators applied to box j, it can thus be noted that

- For each pair of nodes $(i, j) \in \mathcal{A}_p$ with $i \in \mathcal{F}_j^2$, the intermediate matrix $B_{i,j}$ corresponds to the M2T operator mapping from box j to box i.

To summarize, with all these listed analogies, the exact equivalence between FMM and $\mathcal{H}^2$ matrices is established. Replacing all the $\mathcal{H}^2$ matrix components in the $\mathcal{H}^2$ matrix-vector multiplication algorithm (Algorithm 2 illustrated in Chapter 2) by the corresponding translation operators in FMM exactly gives FMM for computing $K(X, X)q$.

In FMM, the combination of the translation operators and the multipole and local expansions essentially construct two degenerate approximations of $K(x, y)$ in each box i and its far field: one by the multipole expansion for source points in box i and one by the local expansion for target points in box i. For each leaf box, the two degenerate approximations of $K(x, y)$ are explicitly known as shown in eq. (5.2). For each non-leaf box, the two degenerate approximations are recursively constructed based on the associated M2M and L2L operators. From the viewpoint of $\mathcal{H}^2$ matrices, these degenerate approximations correspond to an analytic method used by FMM to compress the numerically low-rank blocks $K(X_i, \cup_{k \in \mathcal{F}_i} X_k)$ and $K(\cup_{k \in \mathcal{F}_i} X_k, X_i)$ for each node i and to construct all the $\mathcal{H}^2$ matrix components (the translation operators) analytically. Similarly, many variants of FMM, such as the black-box FMM [20] and the kernel independent FMM [21, 25], can also be regarded as an $\mathcal{H}^2$ matrix, simply with different analytic methods used for low-rank approximation.

5.2 Analytic comparison between FMM and $\mathcal{H}^2$ matrices

We now consider the analytic comparison between FMM and the $\mathcal{H}^2$ matrices with the associated low-rank approximations to be constructed algebraically (referred to as algebraic

$\mathcal{H}^2$ matrices). As illustrated in the last section, the main difference of FMM from algebraic $\mathcal{H}^2$ matrices is that FMM compresses blocks $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ into low-rank form based on specific degenerate approximations in box i and its far field. Thus, FMM shares all the pros and cons of general analytic low-rank approximation methods.

Specifically, FMM is far more efficient in terms of constructing these low-rank approximations than algebraic $\mathcal{H}^2$ matrices if given the same approximation rank. In addition, FMM does not have to store the obtained approximation factors, i.e., the translation operators or the $\mathcal{H}^2$ matrix components, and only needs to dynamically compute them when needed. Thus, FMM requires much less storage than algebraic $\mathcal{H}^2$ matrices but, in sacrifice, requires more computation when applying the translation operators.

On the other hand, the degrees of the degenerate approximations used in FMM, which are the degrees of the applied multipole and local expansions, have to be manually selected in order to meet a given accuracy threshold. In general, choosing the degree to control the approximation accuracy of FMM requires trial-and-error. Furthermore, consider a box $\mathcal{X}$ and its far field $\mathcal{Y}$. For any $X_0 \subset \mathcal{X}$ and $Y_0 \subset \mathcal{Y}$, compressing $K(X_0, Y_0)$ based on a degenerate approximation of $K(x, y)$ in $\mathcal{X} \times \mathcal{Y}$ always has approximation rank larger than the numerical rank of $K(X_0, Y_0)$. In practice, this approximation rank can be prominently larger, especially when X_0 or Y_0 lie in a far smaller subdomain of $\mathcal{X}$ or $\mathcal{Y}$, e.g., a lower-dimensional manifold in $\mathcal{X}$. In comparison, algebraic methods generally can better capture the numerical rank of $K(X_0, Y_0)$. As a result, with the same accuracy, algebraic $\mathcal{H}^2$ matrices can have smaller-rank approximations of all the blocks $K(X_i, \cup_{k\in\mathcal{F}_i} X_k)$ and $K(\cup_{k\in\mathcal{F}_i} X_k, X_i)$ compared to FMM, leading to faster matrix-vector multiplications for algebraic $\mathcal{H}^2$ matrices (note that the $\mathcal{H}^2$ matrix-vector multiplication scales linearly in the approximation rank).

5.3 Numerical comparison between FMM and $\mathcal{H}^2$ matrices

We now demonstrate numerical experiments to substantiate the above analytic comparisons between the two methods, i.e., less construction and storage cost but slower matrix-vector multiplications in FMM compared to algebraic $\mathcal{H}^2$ matrices.

We test three libraries: FMM3D[1] library is used for the typical FMM, PVFMM[2] library [69] is used for the kernel independent FMM (KIFMM) [21], and H2Pack library is used for algebraic $\mathcal{H}^2$ matrices. H2Pack is an $\mathcal{H}^2$ matrix library currently under development by Hua Huang and the author, which applies the proxy point method proposed in Chapter 3 to efficiently construct $\mathcal{H}^2$ matrices. It is worth noting that FMM3D works for three important kernel functions, PVFMM works for kernel functions from potential theory, and H2Pack can work for any general kernel functions. Kernel functions in all the three libraries should be defined in a low-dimensional space, e.g., 2D and 3D.

We test kernel matrices $K(X,X)$ defined by the 3D Laplace kernel $K(x,y) = 1/|x-y|$ and different-sized point sets X randomly generated by two types of distributions: random distribution on the unit sphere in 3D (which is a 2D manifold) and random distribution in the unit ball in 3D. The number of points in X ranges from 10^5 to 1.6×10^6. In the hierarchical partitioning of X, a box is further partitioned into smaller boxes if it contains more than 400 points. For all the libraries, three relative multiplication accuracies around 10^{-5}, 10^{-8}, and 10^{-11} are tested.

All the numerical tests are performed using an Intel Xeon Skylake node on the Stampede2 supercomputer at Texas Advanced Computing Center. Such a node has two sockets and 192 GB DDR4 memory, and each socket has an Intel Xeon Platinum 8160 processor with 24 cores and 2 hyperthreads per core. All the tests are run using one thread per core on all 48 cores. The three libraries are all compiled with Intel C/C++ compiler with optimization flags "-xHost -O3". Intel MKL 17.0.3 is used to perform optimized matrix-vector

[1]FMM3D (https://fmm3d.readthedocs.io/en/latest/index.html) is an improved version of the FMMLIB3D software (https://github.com/zgimbutas/fmmlib3d).

[2]http://pvfmm.org

multiplications, matrix-matrix multiplications, and FFT that appear in these libraries.

Tables 5.1 to 5.3 list the main test results characterized as follows.

- **precomputation cost**, runtime of specific precomputations in H2Pack and PVFMM that can be reused for different sets of points but vary for different accuracy requirements and for different kernel functions. (FMM3D does not have precomputations.) In H2Pack, the precomputation involves computing proxy points for the application of the proxy point method (see Chapter 3) in $\mathcal{H}^2$ matrix construction. In PVFMM, the precomputation involves computing fixed translation operators in KIFMM and storing them into a file.

- **setup cost**, runtime of all the computations other than precomputations before the matrix-vector multiplications, e.g., hierarchical partitioning of X in all the libraries and the $\mathcal{H}^2$ matrix construction in H2Pack.

- **peak memory**, the peak memory usage recorded by the operating system which can be used for a consistent comparison among the three libraries.

- **storage cost**, the storage cost of the translation operators in PVFMM and that of the $\mathcal{H}^2$ matrix components in H2Pack. (FMM3D does not report its storage cost.)

- **runtime and relative error of the multiplication**. These results are averaged over 5 multiplications with 5 random vectors for each tested point set X. For each multiplication, its relative error is measured as the relative error of 100 randomly chosen entries of the computed product compared with the entries of the exact product.

- **rank and/or degree**. The "degree" in PVFMM and FMM3D is a parameter characterizing the number of expansion terms used. In PVFMM, the degree being k corresponds to a rank-$6k^2$ analytic approximation of each block to be compressed in the equivalent $\mathcal{H}^2$ matrix format. In FMM3D, the degree being k corresponds to the approximation rank being $(k+1)^2$. In H2Pack, the maximum and average ranks of all the low-rank approximations in each constructed $\mathcal{H}^2$ matrix are listed.

From the results, the cost for $\mathcal{H}^2$ matrix construction ("setup cost") in H2Pack increases with the number of points and the relative multiplication accuracy. This cost can be prominently more expensive than the setup costs in PVFMM and FMM3D when the number of points becomes large. For example, the setup takes 5.1, 0.8, and 0.9 seconds in H2Pack, PVFMM, and FMM3D, respectively, for 1.6×10^6 points in the unit ball with relative accuracy around 10^{-11}. Meanwhile, the setup cost of H2Pack is relatively much cheaper when points are on the unit sphere than in the unit ball. This is mainly due to the smaller approximation ranks for all the blocks compressed in the $\mathcal{H}^2$ matrix construction.

The maximum and average approximation ranks in H2Pack are all much smaller than those in PVFMM and FMM3D. The approximation ranks in H2Pack also vary with different point distributions, while PVFMM and FMM3D have fixed approximation ranks for both types of point distributions. As a result of smaller approximation ranks, H2Pack is the fastest library in matrix-vector multiplications among the three and this efficiency advantage becomes even greater when dealing with points on the unit sphere, i.e., around 5 times faster than PVFMM and 25 times faster than FMM3D.

The storage cost of H2Pack is proportional to the number of points and the approximations ranks in the constructed $\mathcal{H}^2$ matrices. Meanwhile, the storage cost of PVFMM changes very mildly under different problem settings. Thus, H2Pack has much smaller storage cost for small problems compared with PVFMM but ultimately can have larger storage cost when the number of points or the relative accuracy increases. For example, H2Pack begins to have more storage cost for 8×10^5 points in the unit ball with relative accuracy 10^{-11}. FMM3D does not report its storage cost but FMM theoretically should have very small storage cost, since all the translation operators in FMM are implemented only using a small constant number of analytic functions taking corresponding box locations and points as inputs.

It is worth noting that the peak memory recorded by the operating system depends on the actual implementations of these libraries and can only be used as a rough reference for comparing the three different methods. As can be noted, H2Pack has its peak memory increasing much faster than PVFMM and eventually has larger peak memory than PVFMM when dealing with large number of points and high relative accuracy, e.g., 8×10^5 points in the unit ball with relative accuracy 10^{-11}. Meanwhile, FMM3D also has increasing peak memory with more points but has the smallest peak memory among the three libraries when dealing with a large number of points.

Compared to FMM3D, both H2Pack and PVFMM have relatively expensive precomputations. However, since these precomputations can be reused when the kernel function and the relative accuracy are fixed, the precomputation costs do not make a big difference in the performance comparisons between the three libraries.

To summarize, both the analytical and numerical comparisons above justify that FMM has less cost for setup and storage but slower matrix-vector multiplications than algebraic $\mathcal{H}^2$ matrices. Thus, FMM is more suitable for problems where only a few multiplications are required, e.g., particle simulations. Meanwhile, algebraic $\mathcal{H}^2$ matrices are more suitable for problems where many multiplications are required, e.g., numerical solution of integral equations, so that the expensive $\mathcal{H}^2$ construction cost can be amortized by many relatively cheap multiplications.

Table 5.1: Numerical results of the three libraries with relative accuracy around 10^{-5}. "Precomp" refers to the precomputations in H2Pack and PVFMM. "Mem" refers to the peak memory usage recorded by the operating system. "Storage" refers to the storage cost of translation operators in PVFMM and that of $\mathcal{H}^2$ matrix components in H2Pack.

H2Pack							
#pts $\times 10^5$	precomp(s)	setup(s)	matvec(s)	mem(MB)	storage(MB)	relerr	max/avg rank
sphere 1	2.235	0.073	0.005	519	15	9.19E-06	27/15
sphere 2	2.687	0.084	0.009	436	29	1.07E-05	27/15
sphere 4	3.452	0.125	0.018	569	56	1.23E-05	27/14
sphere 8	4.753	0.220	0.037	839	109	1.33E-05	27/14
sphere 16	6.426	0.429	0.077	1345	217	1.37E-05	27/14
ball 1	2.104	0.102	0.011	669	41	5.20E-06	68/38
ball 2	2.409	0.153	0.018	823	88	8.01E-06	69/35
ball 4	2.129	0.180	0.035	836	156	8.77E-06	71/37
ball 8	2.674	0.293	0.095	1181	305	6.90E-06	72/38
ball 16	3.109	0.750	0.149	2418	669	1.00E-05	70/35

PVFMM								
#pts $\times 10^5$	precomp(s)	setup(s)	matvec(s)	mem(MB)	storage(MB)	relerr	degree	rank
sphere 1	1.161	0.069	0.020	1164	1138	7.51E-06	5	150
sphere 2	1.117	0.087	0.027	1388	1186	8.85E-06	5	150
sphere 4	1.114	0.134	0.056	1799	1271	6.41E-06	5	150
sphere 8	1.163	0.328	0.132	2753	1461	9.91E-06	5	150
sphere 16	1.115	0.686	0.230	4528	1845	9.70E-06	5	150
ball 1	1.113	0.042	0.035	1128	1134	1.99E-05	5	150
ball 2	1.113	0.083	0.030	1355	1186	1.49E-05	5	150
ball 4	1.114	0.113	0.075	1751	1252	1.85E-05	5	150
ball 8	1.113	0.221	0.267	2547	1394	2.98E-05	5	150
ball 16	1.115	0.691	0.219	4518	1832	1.54E-05	5	150

FMM3D						
#pts $\times 10^5$	setup(s)	matvec(s)	mem(MB)	relerr	degree	rank
sphere 1	0.041	0.120	238	8.81E-06	15	256
sphere 2	0.085	0.163	414	8.88E-06	15	256
sphere 4	0.183	0.329	747	9.41E-06	15	256
sphere 8	0.441	0.626	1397	8.61E-06	15	256
sphere 16	1.025	1.259	2784	9.56E-06	15	256
ball 1	0.042	0.167	302	6.55E-06	15	256
ball 2	0.081	0.168	353	6.85E-06	15	256
ball 4	0.170	0.192	554	6.77E-06	15	256
ball 8	0.443	1.266	1830	6.84E-06	15	256
ball 16	0.955	1.261	2025	6.75E-06	15	256

Table 5.2: Numerical results of the three libraries with relative accuracy around 10^{-8}.

H2Pack

#pts $\times 10^5$	precomp(s)	setup(s)	matvec(s)	mem(MB)	storage(MB)	relerr	max/avg rank
sphere 1	2.120	0.101	0.006	778	40	2.29E-08	67/34
sphere 2	2.659	0.138	0.012	717	79	3.03E-08	69/32
sphere 4	3.491	0.194	0.023	908	151	3.22E-08	69/32
sphere 8	4.108	0.333	0.046	1320	303	3.67E-08	69/32
sphere 16	5.271	0.599	0.101	2046	584	3.50E-08	68/32
ball 1	2.108	0.194	0.016	857	123	1.61E-08	186/88
ball 2	2.692	0.330	0.042	1294	288	2.05E-08	191/72
ball 4	2.107	0.444	0.069	1652	498	2.39E-08	195/86
ball 8	2.860	0.668	0.144	2246	906	1.97E-08	195/85
ball 16	3.234	1.573	0.347	4642	2188	2.28E-08	194/75

PVFMM

#pts $\times 10^5$	precomp(s)	setup(s)	matvec(s)	mem(MB)	storage(MB)	relerr	degree	rank
sphere 1	2.981	0.048	0.030	1397	1211	2.35E-08	8	384
sphere 2	2.967	0.089	0.052	1597	1266	2.46E-08	8	384
sphere 4	2.966	0.138	0.122	2112	1358	1.75E-08	8	384
sphere 8	2.966	0.471	0.201	3288	1574	2.57E-08	8	384
sphere 16	2.967	0.658	0.420	4927	1994	2.70E-08	8	384
ball 1	2.970	0.043	0.041	1233	1205	3.57E-08	8	384
ball 2	2.957	0.087	0.058	1530	1264	2.48E-08	8	384
ball 4	2.955	0.115	0.118	1948	1331	3.55E-08	8	384
ball 8	2.959	0.336	0.330	2800	1477	4.07E-08	8	384
ball 16	2.954	0.883	0.454	5052	1971	3.97E-08	8	384

FMM3D

#pts $\times 10^5$	setup(s)	matvec(s)	mem(MB)	relerr	degree	rank
sphere 1	0.041	0.156	298	1.10E-08	21	484
sphere 2	0.084	0.295	454	1.21E-08	21	484
sphere 4	0.184	0.535	860	1.14E-08	21	484
sphere 8	0.418	1.099	1615	1.19E-08	21	484
sphere 16	1.027	2.138	3235	1.28E-08	21	484
ball 1	0.042	0.172	366	1.13E-08	21	484
ball 2	0.081	0.210	359	1.10E-08	21	484
ball 4	0.171	0.863	632	1.11E-08	21	484
ball 8	0.452	1.037	2113	1.11E-08	21	484
ball 16	0.926	1.322	2330	1.18E-08	21	484

Table 5.3: Numerical results of the three libraries with relative accuracy around 10^{-11}.

H2Pack

#pts $\times 10^5$	precomp(s)	setup(s)	matvec(s)	mem(MB)	storage(MB)	relerr	max/avg rank
sphere 1	2.046	0.183	0.010	1222	131	1.56E-12	184/80
sphere 2	3.228	0.250	0.021	1160	264	2.30E-12	186/75
sphere 4	4.124	0.356	0.040	1616	500	2.41E-12	186/75
sphere 8	3.644	0.596	0.079	2398	965	2.58E-12	187/74
sphere 16	4.967	1.093	0.164	4035	1969	2.59E-12	312/75
ball 1	2.168	0.731	0.037	1676	398	2.28E-12	476/185
ball 2	2.119	1.114	0.083	2270	813	5.50E-12	491/109
ball 4	2.848	1.954	0.180	3590	1632	7.75E-12	490/170
ball 8	3.360	3.044	0.330	5426	2770	1.25E-11	499/168
ball 16	2.657	5.140	0.682	9442	5642	1.27E-11	507/109

PVFMM

#pts $\times 10^5$	precomp(s)	setup(s)	matvec(s)	mem(MB)	storage(MB)	relerr	degree	rank
sphere 1	9.700	0.055	0.054	1953	1445	1.29E-11	12	864
sphere 2	9.559	0.104	0.126	2196	1517	1.47E-11	12	864
sphere 4	9.558	0.159	0.234	2555	1624	9.75E-12	12	864
sphere 8	9.562	0.600	0.491	3535	1893	1.44E-11	12	864
sphere 16	9.575	1.014	0.890	5496	2392	1.69E-11	12	864
ball 1	9.547	0.056	0.060	1527	1434	2.76E-11	12	864
ball 2	9.578	0.158	0.151	2086	1510	2.22E-11	12	864
ball 4	9.652	0.180	0.181	2514	1578	2.73E-11	12	864
ball 8	9.595	0.410	0.430	3552	1880	4.31E-11	12	864
ball 16	9.607	0.784	1.123	5544	2351	2.17E-11	12	864

FMM3D

#pts $\times 10^5$	setup(s)	matvec(s)	mem(MB)	relerr	degree	rank
sphere 1	0.034	0.272	278	9.90E-12	29	900
sphere 2	0.078	0.472	553	1.08E-11	29	900
sphere 4	0.167	0.899	907	1.09E-11	29	900
sphere 8	0.375	1.698	1780	1.12E-11	29	900
sphere 16	0.917	3.541	3366	1.11E-11	29	900
ball 1	0.037	0.238	208	9.55E-12	29	900
ball 2	0.098	0.522	678	1.07E-11	29	900
ball 4	0.163	0.654	728	1.10E-11	29	900
ball 8	0.346	2.117	994	1.08E-11	29	900
ball 16	0.947	3.106	4502	1.14E-11	29	900

CHAPTER 6
PRESERVING POSITIVE DEFINITENESS IN HSS MATRIX APPROXIMATIONS

Other than matrix-vector multiplications, solving linear systems defined by dense matrices with block low-rank structure is also common in practice, e.g., numerical solution of integral equations and Gaussian processes. For kernel matrices defined by non-oscillatory kernels and low-dimensional points, hierarchically semi-separable (HSS) matrix format has been used to construct fast direct solvers [10, 12, 13, 14]. Although derived by different approaches, HSS format is equivalent to $\mathcal{H}^2$ format with the weak admissibility condition. Specifically, an HSS matrix compresses all the off-diagonal blocks into low-rank form, leading to fast decomposition and solve algorithms. To construct a direct solver, an accurate HSS representation (rather than an approximate one) is needed but such a representation can be expensive to construct and decompose due to the large approximation ranks needed for the off-diagonal blocks. Instead, to fully exploit block low-rank structures for solving linear systems, it is natural to use highly accurate $\mathcal{H}^2$ approximations (with the strong admissibility condition) to accelerate matrix-vector multiplications or to use less accurate HSS approximations as preconditioners in Krylov subspace methods.

Given a symmetric positive definite (SPD) matrix A, it is desirable to compute an approximate rank-structured representation that is also positive definite. However, positive definiteness is not guaranteed as rank-structured representations all focus on compressing matrix blocks into low-rank form. Preserving positive definiteness is essential for rank-structured approximations to be used efficiently in various algorithms and applications, e.g., using the $\mathcal{H}^2$ matrix representation to accelerate matrix-vector products in the conjugate gradient algorithm. The goal of this chapter is to propose two methods for constructing HSS approximations to an SPD matrix that preserve positive definiteness.

For constructing an HSS approximation using projection to compress the off-diagonal blocks into low-rank form (e.g., using a truncated QR decomposition) [5, 26, 70], we give a new recursive description of the process that says the HSS approximation A_{hss} can be constructed recursively from the leaf level to the root level of the partition tree as

$$A = A^{(0)} \Rightarrow A^{(1)} \Rightarrow A^{(2)} \Rightarrow \ldots \Rightarrow A^{(L)} = A_{\text{hss}},$$

where $A^{(k)}$ is an approximant constructed from $A^{(k-1)}$.

Based on this new description, two positive-definite-preserving methods are designed by making sure that every stage of the construction, $A^{(k-1)} \Rightarrow A^{(k)}$, maintains positive definiteness. Both methods are efficient in that they can be implemented in parallel at each level and their computational complexities are of the same order as existing HSS construction methods with purely algebraic compression approaches [26, 70]. Furthermore, the two methods are also flexible in that they can be further combined with possible accelerated compression techniques, such as the proxy point method introduced in Chapter 3, to help reduce computational complexity.

Instead of directly producing an SPD rank-structured approximation of A, there currently exist methods that construct approximate Cholesky factors of A in HSS form [71, 72]. These algorithms are sequential in the sense that they all involve sequential updates to Schur complements. Our algorithms are different in that they construct SPD HSS matrices directly and are much more parallel.

Another approach to avoid the loss of positive definiteness is diagonal compensation. Bebendorf and Hackbusch [73] builds an SPD $\mathcal{H}$ approximation by adding corrections to the diagonal blocks based on the approximation of each off-diagonal block. Xia [74] adds diagonal shifts to the intermediate blocks in the symmetric ULV decomposition [26, 5] of HSS approximations when breakdowns occur. In comparison, our algorithms construct an SPD HSS approximation only through the compression of off-diagonal blocks.

Of our two methods, the second (called Method 2) is related to the recently developed "structured incomplete factorization" (SIF) from Xia's group [75], which also targets SPD matrices. Both Method 2 and SIF use scalings by diagonal blocks, which for us is one way to preserve positive definiteness. SIF produces a ULV-type factorization [5], which can be constructed with much more parallelism than Cholesky factorizations. Although SIF is likely but not guaranteed to be positive definite when more than one level is used, the framework our two methods are based on can also be used to establish SIF variants that are positive definite by construction.

This chapter also gives a new way of estimating the HSS approximation error, based on the recursive description above. In fact, it can be proved that the errors at each stage are orthogonal, giving

$$\|A - A_{\text{hss}}\|_F^2 = \|A^{(0)} - A^{(1)}\|_F^2 + \|A^{(1)} - A^{(2)}\|_F^2 + \ldots + \|A^{(L-1)} - A^{(L)}\|_F^2,$$

which is also closely related to the error analysis of $\mathcal{H}^2$ approximations in Ref. [3]. By working with $\|A^{(k)} - A^{(k-1)}\|_F^2$, both lower and upper bounds of $\|A - A_{\text{hss}}\|_F^2$ are obtained. It turns out that $\|A^{(k)} - A^{(k-1)}\|_F^2$ at each stage can be bounded above and below within a factor of 2. These bounds are directly related to the corresponding low-rank approximation errors in the HSS construction process. Thus this error analysis justifies existing methods (e.g., [26, 70]) in that they minimize the square of the approximation error at each stage within a factor of 2.

By directly constructing an SPD HSS approximation rather than an approximation in factored form, the errors incurred by the approximation process are better understood, especially by using the orthogonality of the errors at each stage, as shown above. Once an SPD HSS approximation is constructed, an *exact* symmetric ULV decomposition [26] can be computed, if needed, to be applied in various applications.

The rest of the chapter is organized as follows.

- Section 6.1 describes the structure and notation of symmetric HSS matrices.
- Section 6.2 explains the new recursive description of the HSS construction process.
- Section 6.3 proposes two methods of constructing HSS approximations that preserve positive definiteness based on the recursive description of HSS construction.
- Section 6.4 describes the implementation of the first proposed method.
- Section 6.5 describes the implementation of the second proposed method.
- Section 6.6 describes the error analysis of the general HSS construction process and the two proposed methods based on the recursive description of HSS construction.
- Section 6.7 describes numerical experiments for preconditioning SPD matrices by the two proposed methods in two general problems: solving linear systems $Ax = b$ and sampling correlated random vectors $y \in \mathcal{N}(0, A)$.
- Section 6.8 concludes this chapter with an overview of the future work for the proposed methods.

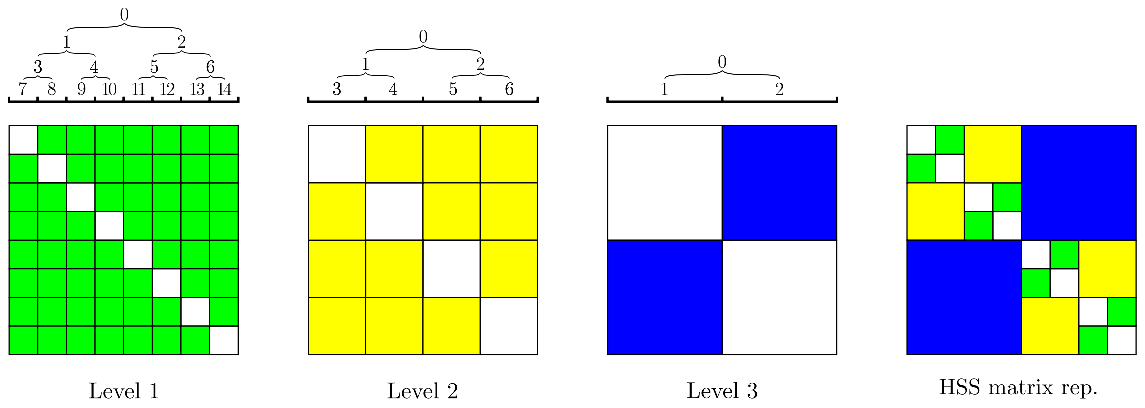
6.1 Symmetric HSS definition and notation

Any symmetric matrix $A \in \mathbb{R}^{n \times n}$ can be associated with an index set $I = \{1, 2, \ldots, n\}$. Let $\mathcal{T}$ be a partition tree associated with a hierarchical partitioning of I. Let I_i denote the index subset of I that corresponds to node i in $\mathcal{T}$. For each non-leaf node i, $I_i = \cup_{i_a \in \{\text{children of node } i\}} I_{i_a}$ and $I_{i_1} \bigcap I_{i_2} = \varnothing$ for any two children i_1 and i_2 of i. The index set associated with the root node is I, containing all the indices. Matrix A is thus hierarchically partitioned into blocks A_{I_i, I_j} with any two nodes $i, j \in \mathcal{T}$.

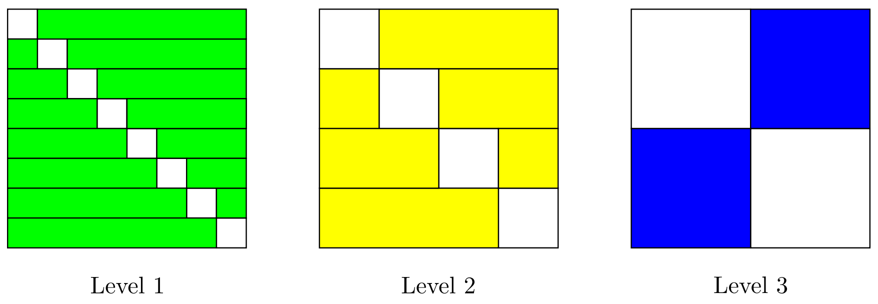
Usually, HSS matrices build upon binary partition trees. For simplicity, we assume $\mathcal{T}$ is a perfect binary partition tree. Figure 6.1 shows an example of a 3-level binary partition tree and the associated hierarchical partitioning of a matrix. We further use the following notations.

- Let n denote the dimension of matrix A and n_i denote the number of elements in I_i.
- For all $i, j \in \mathcal{T}$, denote A_{I_i, I_j} as A_{ij} when there is no ambiguity.
- Let level(k) denote the set of nodes at the kth level.
- For each $i \in$ level(k), let i^c denote the complement set level$(k) \setminus \{i\}$.
- For each non-leaf node i, denote its left and right children as l_i, r_i.
- The block $A_{I_i, I \setminus I_i}$ (which is not a contiguous block of A in general) is called the *HSS block row* of index set I_i, which we abbreviate as A_{ii^c}. It has dimensions $n_i \times (n - n_i)$. (An *HSS block column* can be defined similarly, but due to symmetry of A, this concept will not be needed in this chapter.) The HSS representation exploits the low-rank nature of these blocks. Figure 6.1 illustrates the HSS block rows in a 3-level HSS matrix. As an example, A_{33^c} in the figure is (A_{34}, A_{35}, A_{36}).

The *level* structure of the binary tree is important in this chapter. Nodes at the leaf level are in level 1; their parents are in level 2, etc. The last level is the level just below the



(a) hierarchical partitioning of A and an HSS matrix representation of A



(b) HSS block rows A_{ii^c}

Figure 6.1: Illustration of a 3-level HSS matrix representation of a matrix A with a perfect binary partition tree. In this chapter, the levels are numbered upwards from the leaf level. Subfigure (a) plots the admissible blocks (colored) and inadmissible blocks (white) at each level. The HSS matrix representation is composed of all the inadmissible blocks at level 1 and some of the admissible blocks at levels 1, 2, and 3. Subfigures (b) plots the HSS block rows A_{ii^c} at each level which are to be compressed in the HSS matrix construction.

root, called level L. This numbering of the levels is the reverse of what is generally used in the HSS literature and in the previous chapters, but is more natural in this chapter, as the construction of HSS representations is conceptually from the leaves towards the root.

HSS format is equivalent to $\mathcal{H}^2$ format with the weak admissibility condition where HSS compresses the concatenation of all the off-diagonal blocks that have rows associated with I_i, i.e., the HSS block row A_{ii^c}, for each node $i \in \mathcal{T}$. Meanwhile, $\mathcal{H}^2$ format with the strong admissibility condition discussed in Chapter 2 only compresses the concatenation of some of these off-diagonal blocks. More specifically, at each level k, denote the low-rank approximation of A_{ii^c} with each node $i \in \text{level}(k)$ as

$$A_{ii^c} \approx U_i E_{ii^c}^T. \tag{6.1}$$

For any node $j \in \text{level}(k)$ other than i, A_{ij} is a subblock of both A_{ii^c} and $A_{j^c j}$. Thus, A_{ij} is an *admissible block* and compressed in low-rank form as

$$A_{ij} \approx U_i B_{ij} U_j^T, \quad i \neq j \in \text{level}(k), \tag{6.2}$$

with an intermediate matrix B_{ij} to be computed. The only *inadmissible block* associated with node i in the HSS format is the diagonal block A_{ii}. The low-rank approximation eq. (6.1) of A_{ii^c} for a non-leaf node i is constructed by the same nested approach as in general $\mathcal{H}^2$ matrix construction using the existing low-rank approximations eq. (6.1) of $A_{l_i l_i^c}$ and $A_{r_i r_i^c}$, which leads to the nested representation of U_i,

$$U_i = \begin{pmatrix} U_{l_i} & \\ & U_{r_i} \end{pmatrix} R_i. \tag{6.3}$$

An HSS representation of A is thus made up by dense inadmissible (diagonal) blocks at the leaf level and low-rank approximations eq. (6.2) of the admissible (off-diagonal) blocks at any level that are not contained in larger admissible (off-diagonal) blocks. See Figure 6.1 for a 3-level HSS matrix example. As can be easily verified, an admissible block A_{ij} in an HSS matrix is not contained in a larger admissible block if and only if nodes i and j are siblings and share the same parent node. Based on the above description, we can also derive the original recursive definition of a symmetric HSS matrix from Ref. [76] as follows.

Definition 2. *Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$, a binary partition tree $\mathcal{T}$, and hierarchical index sets $\{I_i\}_{i\in\mathcal{T}}$, the matrix A is a* symmetric HSS matrix *if there are generators D_i, U_i, R_i associated with each non-root node $i \in \mathcal{T}$ and generators B_{ij} associated with* *each pair of siblings $i, j \in \mathcal{T}$ that can be defined recursively from the leaf level to the root level as follows. For each leaf node i, we define the generator $D_i = A_{ii}$. For each non-leaf node i with children l_1 and r_2, the generator $D_i = A_{ii}$ has the structure*

$$D_i = \begin{pmatrix} D_{l_i} & U_{l_i} B_{l_i r_i} U_{r_i}^T \\ U_{r_i} B_{l_i r_i}^T U_{l_i}^T & D_{r_i} \end{pmatrix}$$

with U_i and R_i satisfying the nested basis property as in eq. (6.3).

In the above recursive definition, the generators D_i with leaf nodes i exactly correspond to the inadmissible blocks at the leaf level in the HSS matrix. As an example, a 2-level HSS matrix structure (see the partitioning in Figure 6.1 at levels 2 and 3) writes as

$$\left(\begin{array}{c|c} \begin{matrix} D_3 & U_3 B_{34} U_4^T \\ U_4 B_{34}^T U_3^T & D_4 \end{matrix} & U_1 B_{12} U_2^T \\ \hline U_2 B_{12}^T U_1^T & \begin{matrix} D_5 & U_5 B_{56} U_6^T \\ U_6 B_{56}^T U_5^T & D_6 \end{matrix} \end{array}\right). \tag{6.4}$$

with U_3 and U_6 represented in the nested forms

$$U_1 = \begin{pmatrix} U_3 & \\ & U_4 \end{pmatrix} R_1 \qquad \text{and} \qquad U_2 = \begin{pmatrix} U_5 & \\ & U_6 \end{pmatrix} R_2,$$

For simplicity, we construct HSS approximations using a fixed approximation rank r for HSS block rows and using perfect binary partition trees in the following discussion, but these simplifications are easily lifted.

6.2 Recursive description of HSS construction

In this chapter, we focus on compressing each HSS block row A_{ii^c} into the form $U_i U_i^T A_{ii^c}$ where the columns of U_i are orthonormal and $U_i U_i^T$ is a projection operator. Moreover, each

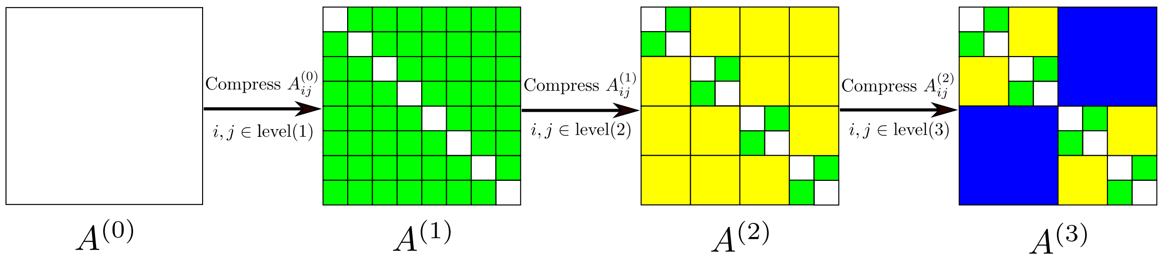


Figure 6.2: Recursive construction of a 3-level HSS approximation (corresponding to Figure 6.1). Green, yellow, and blue denote blocks that are compressed at each level. In general, we say that $A^{(k-1)}$ is compressed at the kth level to obtain $A^{(k)}$, for $k = 1, \ldots, L$. To illustrate our notation, if $k = 2$ and $i \neq j \in \text{level}(k)$, then $A_{ij}^{(k)}$ is a yellow block in $A^{(2)}$, while $A_{ij}^{(k-1)}$ is the corresponding 4 green blocks in $A^{(1)}$ to be compressed to form the yellow block.

off-diagonal block A_{ij} is compressed into the form $U_i U_i^T A_{ij} U_j U_j^T$ and thus the associated intermediate matrix B_{ij} is set to $U_i^T A_{ij} U_j$. We refer to this as the "projection method" for compressing off-diagonal blocks and it is applied in many existing HSS construction methods, e.g., [3, 5, 26, 70]. Algebraic compression methods such as SVD, the pivoted QR decomposition, and randomized methods [57] can be used to construct such U_i.

We now introduce a recursive description of HSS construction using the projection method that will simplify our derivation of SPD HSS approximations. This construction process is described using the bottom-up level-by-level order.

Figure 6.2 gives an overview of the recursive description. Denote the original matrix A as $A^{(0)}$. The blocks of $A^{(0)}$ partitioned at the first level (the leaf level) are denoted as $A_{ij}^{(0)}$ where i and j refer to index sets I_i and I_j, where $i, j \in \text{level}(1)$. These non-diagonal blocks are compressed into the form $U_i U_i^T A_{ij}^{(0)} U_j U_j^T$ and the overall compressed matrix (with its diagonal blocks untouched) is called $A^{(1)}$ as shown in Figure 6.2. The process is then repeated using the index sets at levels 2 and 3, etc., corresponding to partitioning the matrix by coarser and coarser blocks, until $A^{(L)}$ is obtained as the HSS approximation.

Formally, for levels k from 1 to L,

$$A^{(k)} = \text{diag}(\{A_{ii}^{(k-1)}\}_{i \in \text{level}(k)})$$

$$+\,\text{diag}(\{U_iU_i^T\}_{i\in\text{level}(k)})(A^{(k-1)} - \text{diag}(\{A_{ii}^{(k-1)}\}_{i\in\text{level}(k)}))\text{diag}(\{U_iU_i^T\}_{i\in\text{level}(k)}) \tag{6.5}$$

where the notation $\text{diag}(\{M_i\}_{i\in\text{level}(k)})$ denotes the block diagonal matrix composed of all the blocks $\{M_i\}_{i\in\text{level}(k)}$ in order. For example, for the 2-level HSS matrix in eq. (6.4),

$$\text{diag}(\{A_{ii}\}_{i\in\text{level}(1)}) = \begin{pmatrix} A_{33} & & & \\ & A_{44} & & \\ & & A_{55} & \\ & & & A_{66} \end{pmatrix}.$$

This notation will be abusively simplified as $\text{diag}(M_i)$ with $i \in \text{level}(k)$ implied for a given level k.

We note that by the nested basis property, it can be proved that

$$U_iU_i^T A_{ij}^{(k-1)} U_jU_j^T = U_iU_i^T A_{ij}^{(0)} U_jU_j^T, \quad \forall i \neq j \in \text{level}(k)$$

which says that it did not matter that, for example, blocks at level 2 were compressed using compressed blocks at level 1; the result of the compression at level 2 is the same as if we compressed the blocks of the original matrix directly. Practically, this means that for each pair of siblings i and j belonging to $\text{level}(k)$, the intermediate matrix B_{ij} satisfies

$$B_{ij} = U_i^T A_{ij}^{(k-1)} U_j = U_i^T A_{ij}^{(0)} U_j.$$

6.3 New HSS approximations that preserve positive definiteness

Given an SPD matrix A, our goal is to find an SPD HSS approximation. The recursive description of HSS approximation presented in the last section allows us to simplify this task. The recursive description can be written abstractly as

$$A = A^{(0)} \underset{i\in\text{level}(1)}{\overset{U_i}{\Longrightarrow}} A^{(1)} \underset{i\in\text{level}(2)}{\overset{U_i}{\Longrightarrow}} A^{(2)} \ldots \underset{i\in\text{level}(L)}{\overset{U_i}{\Longrightarrow}} A^{(L)} = A_{\text{hss}}.$$

To construct an SPD HSS approximation A_{hss}, it suffices to make sure that each stage, $A^{(k-1)} \Rightarrow A^{(k)}$, computed by update formula eq. (6.5) maintains positive definiteness. Following this approach, two methods are now presented.

6.3.1 Method 1

First, rewrite the update formula eq. (6.5) for level k as

$$A^{(k)} = \text{diag}(U_i U_i^T) A^{(k-1)} \text{diag}(U_i U_i^T) + \text{diag}(A_{ii}^{(k-1)} - U_i U_i^T A_{ii}^{(k-1)} U_i U_i^T) \qquad (6.6)$$

where we remind the reader that $i \in \text{level}(k)$ within each $\text{diag}(\cdot)$ is implied. The first term on the right-hand side is positive semidefinite as we assume $A^{(k-1)}$ to be positive definite. Meanwhile, the second term is a block diagonal matrix and thus it suffices to make sure that each block in that matrix is positive definite.

As shown by Proposition 1 below, the only possible choice of U_i that makes $A_{ii}^{(k-1)} - U_i U_i^T A_{ii}^{(k-1)} U_i U_i^T$ positive semidefinite is the one where $\text{col}(U_i)$ is an invariant subspace of $A_{ii}^{(k-1)}$. In this chapter, we focus on the simplest invariant subspaces, those spanned by any set of eigenvectors. The columns of U_i are chosen to be orthonormal eigenvectors of $A_{ii}^{(k-1)}$. In addition, Proposition 2 shows that this choice of U_i can guarantee $A^{(k)}$ to be positive definite even though $A_{ii}^{(k-1)} - U_i U_i^T A_{ii}^{(k-1)} U_i U_i^T$ can only be guaranteed to be positive semidefinite. It is worth noting that a more general method might exist by exploiting different invariant subspaces of $A_{ii}^{(k-1)}$.

Proposition 1. *Given a symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$ and a tall and skinny matrix $U \in \mathbb{R}^{n\times r}$ with orthonormal columns, $A - UU^TAUU^T$ is positive semidefinite if and only if $\text{col}(U)$ is an invariant subspace of A.*

Proof. Define $\tilde{U} \in \mathbb{R}^{n\times(n-r)}$ with columns forming an orthonormal basis of $\text{col}(U)^\perp$.

Then,

$$\begin{aligned} A - UU^T AUU^T &= (UU^T + \tilde{U}\tilde{U}^T)A(UU^T + \tilde{U}\tilde{U}^T) - UU^T AUU^T \\ &= \tilde{U}\tilde{U}^T A\tilde{U}\tilde{U}^T + \tilde{U}\tilde{U}^T AUU^T + UU^T A\tilde{U}\tilde{U}^T. \end{aligned} \tag{6.7}$$

For any vector $z = Ux + \tilde{U}y$ with $x \in \mathbb{R}^r$, $y \in \mathbb{R}^{n-r}$, the quadratic form is written as $z^T(A - UU^T AUU^T)z = y^T\tilde{U}^T A\tilde{U}y + 2x^T U^T A\tilde{U}y$. As long as $U^T A\tilde{U}y$ is not zero, there exists $x \in \mathbb{R}^r$ such that the quadratic form is negative. Thus, $A - UU^T AUU^T$ is positive semidefinite if and only if $U^T A\tilde{U} = 0$. According to the definition of $\tilde{U}$, $(AU)^T\tilde{U} = 0$ holds true if and only if col$(AU) \subset$ col(U) which is equivalent to col(U) being an invariant subspace of A. $\square$

Proposition 2. *The choice of U_i being composed of orthonormal eigenvectors of $A_{ii}^{(k-1)}$ for any $i \in$ level(k) guarantees that $A^{(k)}$ constructed by formula eq.* (6.6) *is always positive definite.*

Proof. Based on eq. (6.7), if the columns of U_i are orthonormal eigenvectors of $A_{ii}^{(k-1)}$, then $A_{ii}^{(k-1)} - U_iU_i^T A_{ii}^{(k-1)}U_iU_i^T = \tilde{U}_i\tilde{U}_i^T A_{ii}^{(k-1)}\tilde{U}_i\tilde{U}_i^T$ where the columns of $\tilde{U}_i$ form an orthonormal basis of col$(U_i)^\perp$ as before. The update formula eq. (6.6) for level k becomes

$$A^{(k)} = \text{diag}(U_iU_i^T)A^{(k-1)}\text{diag}(U_iU_i^T) + \text{diag}(\tilde{U}_i\tilde{U}_i^T)\text{diag}(A_{ii}^{(k-1)})\text{diag}(\tilde{U}_i\tilde{U}_i^T).$$

Note that both $A^{(k-1)}$ and diag$(A_{ii}^{(k-1)})$ are positive definite. In addition, diag$(U_iU_i^T)$ and diag$(\tilde{U}_i\tilde{U}_i^T)$ are exactly the projection matrices that correspond to a pair of complementary subspaces in $\mathbb{R}^n$. As a result, for any nonzero $x \in \mathbb{R}^n$, at least one of its projections $z_1 =$ diag$(U_iU_i^T)x$ and $z_2 =$ diag$(\tilde{U}_i\tilde{U}_i^T)x$ is nonzero and hence the quadratic form $x^T A^{(k)}x$ written as $x^T A^{(k)}x = z_1^T A^{(k-1)}z_1 + z_2^T\text{diag}(A_{ii}^{(k-1)})z_2$ is always positive. Thus, $A^{(k)}$ is positive definite. $\square$

In summary, Method 1 is to choose each U_i to be composed of r orthonormal eigenvectors of $A_{ii}^{(k-1)}$. How to do this while minimizing the projection error and to also satisfy the nested basis property at non-leaf levels will be discussed in Section 6.4. Pseudocode for Method 1 thus far using the bottom-up level-by-level construction order is shown in Algorithm 8.

Algorithm 8 Method 1 (abstract version)

```
Input: Original SPD matrix $A$
Output: SPD HSS approximation $A^{(L)}$
  • set $A^{(0)} = A$
  for $k = 1, 2, \ldots, L$ do
      • compute $U_i$, $\forall i \in$ level$(k)$ satisfying
        – columns of $U_i$ are orthonormal eigenvectors of $A_{ii}^{(k-1)}$
        – $U_i$ should minimize the projection error $\|A_{ii^c}^{(k-1)} - U_i U_i^T A_{ii^c}^{(k-1)}\|_F$
        – if $k > 1$, the nested basis property must also be satisfied
      • compress $A^{(k-1)}$ by formula eq. (6.5) to obtain $A^{(k)}$
  end for
```

6.3.2 Method 2

From eqs. (6.6) and (6.7), the update formula eq. (6.5) for level k can be written as

$$\begin{aligned} A^{(k)} &= \text{diag}(U_i U_i^T) A^{(k-1)} \text{diag}(U_i U_i^T) + \text{diag}(\tilde{U}_i \tilde{U}_i^T A_{ii}^{(k-1)} \tilde{U}_i \tilde{U}_i^T) \\ &\quad + \text{diag}(U_i U_i^T A_{ii}^{(k-1)} \tilde{U}_i \tilde{U}_i^T + \tilde{U}_i \tilde{U}_i^T A_{ii}^{(k-1)} U_i U_i^T), \quad i \in \text{level}(k), \qquad (6.8) \end{aligned}$$

where the columns of $\tilde{U}_i$ form an orthonormal basis of $\text{col}(U_i)^\perp$ as before. The sum of the first and second terms can be proved to be positive definite by an argument similar to that in Proposition 2. Thus, the block diagonal matrix diag$(U_i U_i^T A_{ii}^{(k-1)} \tilde{U}_i \tilde{U}_i^T + \tilde{U}_i \tilde{U}_i^T A_{ii}^{(k-1)} U_i U_i^T)$ is the term that might make $A^{(k)}$ indefinite.

In fact, the constraint that the columns of U_i are orthonormal eigenvectors of $A_{ii}^{(k-1)}$ in Method 1 makes $U_i U_i^T A_{ii}^{(k-1)} \tilde{U}_i \tilde{U}_i^T + \tilde{U}_i \tilde{U}_i^T A_{ii}^{(k-1)} U_i U_i^T$ exactly zero. From this point of view, the key is to try to get rid of this term. Thus, Method 2 can be designed as follows.

Notice that if $A_{ii}^{(k-1)}$ is an identity matrix, then $U_i U_i^T A_{ii}^{(k-1)} \tilde{U}_i \tilde{U}_i^T$ will be zero and $A^{(k)}$ will be positive definite. Identity diagonal blocks remind us of scaling by diagonal blocks.

Ignoring for now the nested basis property and focusing on one update, $A^{(k-1)} \Rightarrow A^{(k)}$, the idea is to symmetrically scale $A^{(k-1)}$ by its diagonal blocks, and then to compress the scaled off-diagonal blocks using formula eq. (6.5).

Formally, first calculate the symmetric factorization of $A_{ii}^{(k-1)} = S_i S_i^T$ for each node $i \in \text{level}(k)$, and scale the matrix $A^{(k-1)}$ as $C^{(k-1)} = \text{diag}(S_i^{-1}) A^{(k-1)} \text{diag}(S_i^{-T})$. Then, compress the off-diagonal blocks of $C^{(k-1)}$ using formula eq. (6.5) to obtain

$$C^{(k)} = \text{diag}(C_{ii}^{(k-1)}) + \text{diag}(V_i V_i^T)(C^{(k-1)} - \text{diag}(C_{ii}^{(k-1)}))\text{diag}(V_i V_i^T) \tag{6.9}$$

with $i \in \text{level}(k)$, where $V_i \in \mathbb{R}^{n_i \times r}$ has orthonormal columns. Intuitively, V_i should be chosen to minimize the projection error $\|C_{ii^c}^{(k-1)} - V_i V_i^T C_{ii^c}^{(k-1)}\|_F$. Finally, define $A^{(k)} = \text{diag}(S_i) C^{(k)} \text{diag}(S_i^T)$ which can be generated recursively as

$$A^{(k)} = \text{diag}(A_{ii}^{(k-1)}) + \text{diag}(S_i V_i V_i^T S_i^{-1})(A^{(k-1)} - \text{diag}(A_{ii}^{(k-1)}))\text{diag}(S_i^{-T} V_i V_i^T S_i^T) \tag{6.10}$$

where each off-diagonal block is compressed as $A_{ij}^{(k)} = S_i V_i V_i^T S_i^{-1} A_{ij}^{(k-1)} S_j^{-T} V_j V_j^T S_j^T$. Denote $U_i = S_i V_i$ and $W_i^T = V_i^T S_i^{-1}$. Unlike before, this newly-defined U_i does not have orthonormal columns and $U_i W_i^T$ is not a projection matrix. However, the matrix $S_i V_i V_i^T S_i^{-1}$ can be proved to be the projection onto the subspace $\text{col}(S_i V_i)$ with inner product defined as $(x, y) = x^T S_i^{-T} S_i^{-1} y$.

The update formula eq. (6.10) gives a positive definite matrix and the only requirement so far has been that the columns of V_i are orthonormal. However, we still must guarantee that the nested basis property is satisfied, which will place an additional condition on V_i.

Based on the definition of the nested basis property in eq. (6.3), there should exist R_i such that

$$V_i = S_i^{-1} \begin{pmatrix} U_{l_i} & \\ & U_{r_i} \end{pmatrix} R_i$$

which is equivalent to $\text{col}(V_i) \subset \text{col}\left(S_i^{-1} \begin{pmatrix} U_{l_i} & \\ & U_{r_i} \end{pmatrix}\right)$. By the definition of $C^{(k-1)}$ and the update of $A^{(k-2)}$ to $A^{(k-1)}$ through eq. (6.10), it can be noted that

$$\mathrm{col}(C_{ii^c}^{(k-1)}) \subset \mathrm{col}(S_i^{-1} A_{ii^c}^{(k-1)}) \subset \mathrm{col}\left(S_i^{-1} \begin{pmatrix} U_{l_i} W_{l_i}^T A_{l_i i^c}^{(k-2)} \\ U_{r_i} W_{r_i}^T A_{r_i i^c}^{(k-2)} \end{pmatrix} \right) \subset \mathrm{col}\left(S_i^{-1} \begin{pmatrix} U_{l_i} & \\ & U_{r_i} \end{pmatrix} \right).$$

Previously, V_i is chosen to compress $C_{ii^c}^{(k-1)}$ as $V_i V_i^T C_{ii^c}^{(k-1)}$. Thus the sufficient condition $\mathrm{col}(V_i) \subset \mathrm{col}(C_{ii^c}^{(k-1)})$ to satisfy the nested basis property can be enforced naturally.

Pseudocode for Method 2 thus far using the bottom-up level-by-level construction order is shown in Algorithm 9.

Algorithm 9 Method 2 (abstract version)

```
Input: Original SPD matrix $A$
Output: SPD HSS approximation $A^{(L)}$
  • set $A^{(0)} = A$
  for $k = 1, 2, \dots, L$ do
      • construct a symmetric factorization $A_{ii}^{(k-1)} = S_i S_i^T, \forall i \in \mathrm{level}(k)$
      • calculate the scaled off-diagonal block $C_{ij}^{(k-1)} = S_i^{-1} A_{ij}^{(k-1)} S_j^{-T}, \forall i \neq j \in \mathrm{level}(k)$
      • compute $V_i, \forall i \in \mathrm{level}(k)$ satisfying
          – columns of $V_i$ are orthonormal and $\mathrm{col}(V_i) \subset \mathrm{col}(C_{ii^c}^{(k-1)})$
          – $V_i$ should minimize the projection error $\|C_{ii^c}^{(k-1)} - V_i V_i^T C_{ii^c}^{(k-1)}\|_F$
      • compress $A^{(k-1)}$ by formula eq. (6.10) to obtain $A^{(k)}$
  end for
```

6.4 Implementation of Method 1

The previous section gave the description of two SPD HSS construction methods without implementation details. Here, an efficient implementation of Method 1 is presented. The implementation is related to building an HSS representation using projection method, which we explain first. This method, which does not try to preserve positive definiteness, will be called the *standard HSS* method in this chapter and it is exactly the symmetric version of the method in Ref. [70].

6.4.1 Standard HSS construction using projection

For level k from 1 to L, define the $r \times r$ matrix $M_{ij}^{(k)} = U_i^T A_{ij}^{(k-1)} U_j$, $\forall i \neq j \in \text{level}(k)$ which satisfies $A_{ij}^{(k)} = U_i M_{ij}^{(k)} U_j^T$ by the update formula eq. (6.5). Here, i, j for $M_{ij}^{(k)}$ are used as partition tree node indices like those in U_i and B_{ij} and do not refer to the pair of index subsets $I_i \times I_j$ like in $A_{ij}^{(k)}$. Notice that for each pair of siblings $i, j \in \text{level}(k)$, the associated intermediate matrix $B_{ij} = U_i^T A_{ij}^{(k-1)} U_j = M_{ij}^{(k)}$.

We also define $M_{ij}^{(k-1)}$ where i and j (with $i \neq j$) belong to level(k) rather than level$(k-1)$ as the $2r \times 2r$ matrix that satisfies

$$A_{ij}^{(k-1)} = \begin{pmatrix} A_{l_i l_j}^{(k-1)} & A_{l_i r_j}^{(k-1)} \\ A_{r_i l_j}^{(k-1)} & A_{r_i r_j}^{(k-1)} \end{pmatrix} = \begin{pmatrix} U_{l_i} & \\ & U_{r_i} \end{pmatrix} \begin{pmatrix} M_{l_i l_j}^{(k-1)} & M_{l_i r_j}^{(k-1)} \\ M_{r_i l_j}^{(k-1)} & M_{r_i r_j}^{(k-1)} \end{pmatrix} \begin{pmatrix} U_{l_j}^T & \\ & U_{r_j}^T \end{pmatrix}.$$

At the leaf level, U_i for each $i \in \text{level}(1)$ is computed to minimize the projection error when compressing $A_{ii^c}^{(0)}$. At level $k > 1$, U_i for each $i \in \text{level}(k)$ is defined implicitly by R_i which is computed to minimize the projection error when compressing $A_{ii^c}^{(k-1)}$.

By exploiting the nested basis property and the above relation between $A_{ij}^{(k-1)}$ and $M_{ij}^{(k-1)}$ for $i, j \in \text{level}(k)$, the projection error is

$$\left\| A_{ii^c}^{(k-1)} - U_i U_i^T A_{ii^c}^{(k-1)} \right\|_F = \left\| M_{ii^c}^{(k-1)} - R_i R_i^T M_{ii^c}^{(k-1)} \right\|_F \tag{6.11}$$

where $M_{ii^c}^{(k-1)}$ is naturally defined as $\left(M_{ij_1}^{(k-1)}, M_{ij_2}^{(k-1)} \ \ldots\right)$, with $\{j_1, j_2, \ldots\} = i^c$. In addition, by the nested basis property, the columns of U_i being orthonormal is equivalent to the columns of R_i being orthonormal. Thus the thin matrix $M_{ii^c}^{(k-1)}$ of size $2r \times 2r(|\text{level}(k)| - 1)$ is the target matrix to compress to obtain R_i. After that, $\{M_{ij}^{(k)}\}_{i,j \in \text{level}(k)}$ can be calculated from $\{M_{pq}^{(k-1)}\}_{p,q \in \text{level}(k-1)}$ as

$$M_{ij}^{(k)} = R_i^T M_{ij}^{(k-1)} R_j.$$

To summarize, the standard HSS approximation is given in Algorithm 10 and it has $O(rn^2)$ computational complexity with fixed rank r.

Algorithm 10 Standard bottom-up level-by-level HSS construction

```
Input: HSS rank $r$, original matrix $A$
Output: HSS approximation with components $\{D_i\}, \{B_{ij}\}, \{U_i\}, \{R_i\}$
  At the leaf level
    • set $D_i = A_{ii}, \forall i \in$ level$(1)$
    • compute $U_i \in \mathbb{R}^{n_i \times r}, \forall i \in$ level$(1)$ satisfying
      – columns of $U_i$ are orthonormal
      – $U_i$ should minimize $\|A_{ii^c} - U_i U_i^T A_{ii^c}\|_F$
    • compute $M_{ij}^{(1)} = U_i^T A_{ij} U_j, \forall i \neq j \in$ level$(1)$
    • set $B_{ij} = M_{ij}^{(1)}$ for every pair of siblings $i, j \in$ level$(1)$
  for $k = 2, 3, \ldots, L$ do
    • compute $R_i \in \mathbb{R}^{2r \times r}, \forall i \in$ level$(k)$ satisfying
      – columns of $R_i$ are orthonormal
      – $R_i$ should minimize $\|M_{ii^c}^{(k-1)} - R_i R_i^T M_{ii^c}^{(k-1)}\|_F$
    • compute $M_{ij}^{(k)} = R_i^T M_{ij}^{(k-1)} R_j, \forall i \neq j \in$ level$(k)$
    • set $B_{ij} = M_{ij}^{(k)}$ for every pair of siblings $i, j \in$ level$(k)$
  end for
```

6.4.2 Method 1: constrained optimization problem

In Method 1, at level k, we seek U_i that minimizes the projection error $\|A_{ii^c}^{(k-1)} - U_i U_i^T A_{ii^c}^{(k-1)}\|_F$, for $i \in$ level(k), while U_i is also constrained to be exactly r orthonormal eigenvectors of the diagonal block $A_{ii}^{(k-1)}$. Ignoring for now the nested basis property that must also be satisfied at non-leaf levels, this leads to the following constrained optimization problem.

Problem 1. *Given an orthogonal matrix $V \in \mathbb{R}^{m \times m}$ and a target matrix $B \in \mathbb{R}^{m \times l}$, find r columns of V, which we call $U \in \mathbb{R}^{m \times r}$, such that U minimizes the projection error $\|B - UU^T B\|_F$.*

The solution is as follows. Any r columns of V can be represented as $U = VP\,(I_r,\ 0)^T$, where P is a permutation matrix. Substituting this U into the projection error expression, the problem becomes finding the permutation P that minimizes

$$\|B - UU^TB\|_F = \left\|B - VP\left(\begin{smallmatrix} I_r & 0 \\ 0 & 0 \end{smallmatrix}\right)P^TV^TB\right\|_F = \left\|P^TG - \left(\begin{smallmatrix} I_r & 0 \\ 0 & 0 \end{smallmatrix}\right)P^TG\right\|_F$$

where $G = V^TB \in \mathbb{R}^{m\times l}$. Thus we only need to choose a permutation of the rows of G such that the first r rows of P^TG have the largest norms.

As $\|P^TG\|_F = \|B\|_F$, we have $\|B - UU^TB\|_F \leqslant (1 - \frac{r}{n})^{\frac{1}{2}}\|B\|_F$ where equality holds only when all the rows of G have the same norm.

6.4.3 Method 1: full implementation

At level 1, the diagonal blocks $A_{ii}^{(0)} \in \mathbb{R}^{n_i\times n_i}$ for $i \in \text{level}(1)$, are typically small, and their eigen-decompositions are readily computed. The U_i that are sought can be computed by solving Problem 1 where V is the matrix of eigenvectors and the target matrix B is the HSS block row $A_{ii^c}^{(0)}$.

At non-leaf levels k, we need U_i to satisfy the nested basis property in addition to minimizing the projection error and the columns of U_i being constrained to be selected orthonormal eigenvectors of the diagonal block $A_{ii}^{(k-1)}$.

For $i \in \text{level}(k)$, the diagonal block $A_{ii}^{(k-1)}$ can be written as

$$A_{ii}^{(k-1)} = \begin{pmatrix} A_{l_il_i}^{(k-2)} & U_{l_i}U_{l_i}^TA_{l_ir_i}^{(k-2)}U_{r_i}U_{r_i}^T \\ U_{r_i}U_{r_i}^TA_{r_il_i}^{(k-2)}U_{l_i}U_{l_i}^T & A_{r_ir_i}^{(k-2)} \end{pmatrix}.$$

The columns of $U_i \in \mathbb{R}^{n_i\times r}$ being eigenvectors of $A_{ii}^{(k-1)}$ is equivalent to there being a $r\times r$ diagonal matrix Σ_i such that

$$A_{ii}^{(k-1)}U_i = U_i\Sigma_i$$

$$A_{ii}^{(k-1)}\begin{pmatrix} U_{l_i} & \\ & U_{r_i} \end{pmatrix}R_i = \begin{pmatrix} U_{l_i} & \\ & U_{r_i} \end{pmatrix}R_i\Sigma_i$$

$$\begin{pmatrix} A_{l_il_i}^{(k-2)}U_{l_i} & U_{l_i}U_{l_i}^TA_{l_ir_i}^{(k-2)}U_{r_i} \\ U_{r_i}U_{r_i}^TA_{r_il_i}^{(k-2)}U_{l_i} & A_{r_ir_i}^{(k-2)}U_{r_i} \end{pmatrix} R_i = \begin{pmatrix} U_{l_i} & \\ & U_{r_i} \end{pmatrix} R_i\Sigma_i.$$

By induction, $A_{l_il_i}^{(k-2)}U_{l_i} = U_{l_i}\Sigma_{l_i}$, and $A_{r_ir_i}^{(k-2)}U_{r_i} = U_{r_i}\Sigma_{r_i}$. Thus the calculation continues as

$$\begin{pmatrix} U_{l_i}\Sigma_{l_i} & U_{l_i}U_{l_i}^TA_{l_ir_i}^{(k-2)}U_{r_i} \\ U_{r_i}U_{r_i}^TA_{r_il_i}^{(k-2)}U_{l_i} & U_{r_i}\Sigma_{r_i} \end{pmatrix} R_i = \begin{pmatrix} U_{l_i} & \\ & U_{r_i} \end{pmatrix} R_i\Sigma_i$$

$$\begin{pmatrix} \Sigma_{l_i} & U_{l_i}^TA_{l_ir_i}^{(k-2)}U_{r_i} \\ U_{r_i}^TA_{r_il_i}^{(k-2)}U_{l_i} & \Sigma_{r_i} \end{pmatrix} R_i = R_i\Sigma_i. \tag{6.12}$$

The term $U_{l_i}^TA_{l_ir_i}^{(k-2)}U_{r_i}$ is exactly $B_{l_ir_i}$ or $M_{l_ir_i}^{(k-1)}$ and has been calculated at the previous level. Denote the leading matrix in eq. (6.12) as $E_i = \begin{pmatrix} \Sigma_{l_i} & B_{l_ir_i} \\ B_{l_ir_i}^T & \Sigma_{r_i} \end{pmatrix}$.

Every step above is invertible. Thus, the columns of U_i being orthonormal eigenvectors of $A_{ii}^{(k-1)}$ is equivalent to the columns of R_i being orthonormal eigenvectors of the small $2r \times 2r$ symmetric matrix E_i.

Now, consider computing U_i to minimize the projection error of $A_{ii^c}^{(k-1)}$. From eq. (6.11), minimizing $\|A_{ii^c}^{(k-1)} - U_iU_i^TA_{ii^c}^{(k-1)}\|_F$ under the nested basis constraint is equivalent to minimizing $\|M_{ii^c}^{(k-1)} - R_iR_i^TM_{ii^c}^{(k-1)}\|_F$. By the analysis above, with eigen-decomposition $E_i = V_i\Lambda_iV_i^T$, the optimal R_i can be obtained by solving Problem 1 with orthogonal matrix V_i, target matrix $M_{ii^c}^{(k-1)}$, and desired rank r.

The efficient implementation of Method 1 can now be given in Algorithm 11. With fixed rank r, the complexity of the algorithm is $O(rn^2)$.

6.5 Implementation of Method 2

The main difference between Method 2 and the standard HSS method is the symmetric scaling by diagonal blocks at each level. At first glance, the decomposition $A_{ii}^{(k-1)} = S_iS_i^T$

Algorithm 11 Method 1 with bottom-up level-by-level construction

```
Input: HSS rank $r$, original SPD matrix $A$
Output: SPD HSS approximation with components $\{D_i\}, \{B_{ij}\}, \{U_i\}, \{R_i\}$
  At the leaf level
      • set $D_i = A_{ii}, \forall i \in$ level$(1)$
      • compute eigen-decomposition $A_{ii} = V_i \Lambda_i V_i^T, \forall i \in$ level$(1)$
      • compute $U_i$ by solving problem 1 with $V_i$, $A_{ii^c}$ and $r, \forall i \in$ level$(1)$
      • store diagonal matrix $\Sigma_i = U_i^T A_{ii} U_i, \forall i \in$ level$(1)$
      • compute $M_{ij}^{(1)} = U_i^T A_{ij} U_j, \forall i \neq j \in$ level$(1)$
      • set $B_{ij} = M_{ij}^{(1)}$ for each pair of siblings $i, j \in$ level$(1)$
  for $k = 2, 3, \dots, L$ do
      • compute eigen-decomposition $E_i = \begin{pmatrix} \Sigma_{l_i} & B_{l_i r_i} \\ B_{l_i r_i}^T & \Sigma_{r_i} \end{pmatrix} = V_i \Lambda_i V_i^T, \forall i \in$ level$(k)$
      • compute $R_i$ by solving problem 1 with $V_i$, $M_{ii^c}^{(k-1)}$ and $r, \forall i \in$ level$(k)$
      • store diagonal matrix $\Sigma_i = R_i^T E_i R_i, \forall i \in$ level$(k)$
      • compute $M_{ij}^{(k)} = R_i^T M_{ij}^{(k-1)} R_j, \forall i \neq j \in$ level$(k)$
      • set $B_{ij} = M_{ij}^{(k)}$ for each pair of siblings $i, j \in$ level$(k)$
  end for
```

and the application of S_i^{-1} do not appear to be practical due to the large size of these blocks at higher levels. However, as shown below, the storage cost and computational complexity to obtain S_i and to apply S_i^{-1} at each level can be reduced and are only related to the off-diagonal block rank r. The complexity of Method 2 is still of the same order as that of the standard HSS method.

Similar to the standard HSS construction procedure, for level k from 1 to L, define $M_{ij}^{(k)} = V_i^T S_i^{-1} A_{ij}^{(k-1)} S_j^{-T} V_j \in \mathbb{R}^{r \times r}$ for $i \neq j \in$ level(k) which satisfies $A_{ij}^{(k)} = U_i M_{ij}^{(k)} U_j^T$. If i and j are siblings, also define $B_{ij} = M_{ij}^{(k)}$ by the update formula eq. (6.10).

At the leaf level, all calculations can be performed directly because the matrices are small. For compressions at non-leaf level k, i.e., to obtain $A^{(k)}$ from $A^{(k-1)}$, the following quantities need to be calculated.

- For $i \in$ level(k), the symmetric decomposition $A_{ii}^{(k-1)} = S_i S_i^T$.
- For $i \neq j \in$ level(k), the scaled off-diagonal block $C_{ij}^{(k-1)} = S_i^{-1} A_{ij}^{(k-1)} S_j^{-T}$.
- For $i \neq j \in$ level(k), $M_{ij}^{(k)} = V_i^T S_i^{-1} A_{ij}^{(k-1)} S_j^{-T} V_j = V_i^T C_{ij}^{(k-1)} V_j$.

- For $i \in \text{level}(k)$,

$$R_i = \begin{pmatrix} V_{l_i}^T & \\ & V_{r_i}^T \end{pmatrix} \begin{pmatrix} S_{l_i}^{-1} & \\ & S_{r_i}^{-1} \end{pmatrix} S_i V_i \tag{6.13}$$

  to satisfy the nested basis property for $U_i = S_i V_i$.

We now show how each of these four quantities can be computed efficiently.

Symmetric decomposition $A_{ii}^{(k-1)} = S_i S_i^T$ At the leaf level, the Cholesky decomposition $A_{ii}^{(0)} = S_i S_i^T$ can be directly calculated. At non-leaf levels k, using compression at the previous level, $A_{ii}^{(k-1)}$ for $i \in \text{level}(k)$ can be written as

$$A_{ii}^{(k-1)} = \begin{pmatrix} A_{l_i l_i}^{(k-2)} & U_{l_i} B_{l_i r_i} U_{r_i}^T \\ U_{r_i} B_{l_i r_i}^T U_{l_i}^T & A_{r_i r_i}^{(k-2)} \end{pmatrix},$$

with $U_{l_i} = S_{l_i} V_{l_i}$, $U_{r_i} = S_{r_i} V_{r_i}$, and $B_{l_i r_i} = V_{l_i}^T S_{l_i}^{-1} A_{l_i r_i}^{(k-2)} S_{r_i}^{-T} V_{r_i}$. Knowing that $A_{l_i l_i}^{(k-2)} = S_{l_i} S_{l_i}^T$ and $A_{r_i r_i}^{(k-2)} = S_{r_i} S_{r_i}^T$, diagonal block $A_{ii}^{(k-1)}$ can be decomposed as

$$A_{ii}^{(k-1)} = \begin{pmatrix} S_{l_i} & \\ & S_{r_i} \end{pmatrix} \begin{pmatrix} I & V_{l_i} B_{l_i r_i} V_{r_i}^T \\ V_{r_i} B_{l_i r_i}^T V_{l_i}^T & I \end{pmatrix} \begin{pmatrix} S_{l_i}^T & \\ & S_{r_i}^T \end{pmatrix}. \tag{6.14}$$

Hence, only a symmetric decomposition of the middle matrix is needed. For this, we will use the following proposition.

Proposition 3. *Consider a matrix $H = \left(\begin{smallmatrix} I_{n_1} & M \\ M^T & I_{n_2} \end{smallmatrix}\right) \in \mathbb{R}^{(n_1+n_2)\times(n_1+n_2)}$, where sub-block $M \in \mathbb{R}^{n_1 \times n_2}$ is rank r with SVD $M = U\Sigma V^T$, $U \in \mathbb{R}^{n_1 \times r}$, $\Sigma \in \mathbb{R}^{r \times r}$, $V \in \mathbb{R}^{n_2 \times r}$. The* *eigen-decomposition of this matrix is*

$$\begin{pmatrix} \frac{U}{\sqrt{2}} & \frac{U}{\sqrt{2}} & \tilde{U} & 0 \\ \frac{V}{\sqrt{2}} & \frac{-V}{\sqrt{2}} & 0 & \tilde{V} \end{pmatrix} \begin{pmatrix} I_r + \Sigma & & & \\ & I_r - \Sigma & & \\ & & I_{n_1 - r} & \\ & & & I_{n_2 - r} \end{pmatrix} \begin{pmatrix} \frac{U}{\sqrt{2}} & \frac{U}{\sqrt{2}} & \tilde{U} & 0 \\ \frac{V}{\sqrt{2}} & \frac{-V}{\sqrt{2}} & 0 & \tilde{V} \end{pmatrix}^T,$$

where the columns of $\tilde{U}$ and $\tilde{V}$ form orthonormal bases of $\mathrm{col}(U)^\perp$ and $\mathrm{col}(V)^\perp$ respectively. Furthermore, based on this eigen-decomposition, if H is SPD, singular values of M must be less than one and hence the eigenvalues of H are within $(0, 2)$.

The proposition can be verified by direct calculation. Now, $B_{l_i r_i}$ is a small $r \times r$ matrix and its full SVD can be readily calculated as $B_{l_i r_i} = Q_{l_i} \Sigma_{l_i r_i} Q_{r_i}^T$. As $A_{ii}^{(k-1)}$ is SPD, the matrix in the middle of eq. (6.14) is also SPD and can be decomposed as $\bar{S}_i \bar{S}_i^T$ where

$$\bar{S}_i = \begin{pmatrix} \frac{V_{l_i} Q_{l_i}}{\sqrt{2}} & \frac{V_{l_i} Q_{l_i}}{\sqrt{2}} & \tilde{V}_{l_i} & \\ \frac{V_{r_i} Q_{r_i}}{\sqrt{2}} & \frac{-V_{r_i} Q_{r_i}}{\sqrt{2}} & & \tilde{V}_{r_i} \end{pmatrix} \begin{pmatrix} (I_r + \Sigma_{l_i r_i})^{\frac{1}{2}} & & \\ & (I_r - \Sigma_{l_i r_i})^{\frac{1}{2}} & \\ & & I_{n_1 + n_2 - 2r} \end{pmatrix} \tag{6.15}$$

based on Proposition 3 and the columns of $\tilde{V}_{l_i}$ and $\tilde{V}_{r_i}$ form orthonormal bases of col$(V_{l_i})^\perp$ and col$(V_{r_i})^\perp$ respectively. Thus, a recursive definition of S_i by eq. (6.14) is obtained as

$$S_i = \begin{pmatrix} S_{l_i} & \\ & S_{r_i} \end{pmatrix} \bar{S}_i \tag{6.16}$$

where $\bar{S}_i$ only requires V_{l_i}, V_{r_i} and $B_{l_i r_i}$ at the previous level.

Scaled off-diagonal block $C_{ij}^{(k-1)} = S_i^{-1} A_{ij}^{(k-1)} S_j^{-T}$ First consider the one-sided multiplication $S_i^{-1} A_{ij}^{(k-1)}$ for $i \neq j \in$ level(k). Using compression at the previous level, $A_{ij}^{(k-1)}$

can be written as

$$\begin{aligned}
A_{ij}^{(k-1)} &= \begin{pmatrix} S_{l_i}V_{l_i}V_{l_i}^TS_{l_i}^{-1} & \\ & S_{r_i}V_{r_i}V_{r_i}^TS_{r_i}^{-1} \end{pmatrix} A_{ij}^{(k-2)} \begin{pmatrix} S_{l_j}V_{l_j}V_{l_j}^TS_{l_j}^{-1} & \\ & S_{r_j}V_{r_j}V_{r_j}^TS_{r_j}^{-1} \end{pmatrix}^T \\
&= \begin{pmatrix} S_{l_i}V_{l_i}V_{l_i}^TS_{l_i}^{-1} & \\ & S_{r_i}V_{r_i}V_{r_i}^TS_{r_i}^{-1} \end{pmatrix} \tilde{A}_{ij}^{(k-2)},
\end{aligned}$$

where $\tilde{A}_{ij}^{(k-2)}$ denotes the multiplication of the last two matrices in the first line above. Combining this expression with eq. (6.15) and eq. (6.16), we obtain

$$\begin{aligned}
& S_i^{-1}A_{ij}^{(k-1)} \\
&= \bar{S}_i^{-1} \begin{pmatrix} V_{l_i}V_{l_i}^TS_{l_i}^{-1} & \\ & V_{r_i}V_{r_i}^TS_{r_i}^{-1} \end{pmatrix} \tilde{A}_{ij}^{(k-2)} \\
&= \begin{pmatrix} (I_r+\Sigma_{l_ir_i})^{-\frac{1}{2}} & & \\ & (I_r-\Sigma_{l_ir_i})^{-\frac{1}{2}} & \\ & & I \end{pmatrix} \begin{pmatrix} \frac{Q_{l_i}^TV_{l_i}^T}{\sqrt{2}} & \frac{Q_{r_i}^TV_{r_i}^T}{\sqrt{2}} \\ \frac{Q_{l_i}^TV_{l_i}^T}{\sqrt{2}} & \frac{-Q_{r_i}^TV_{r_i}^T}{\sqrt{2}} \\ \tilde{V}_{l_i}^T & 0 \\ 0 & \tilde{V}_{r_i}^T \end{pmatrix} \begin{pmatrix} V_{l_i}V_{l_i}^TS_{l_i}^{-1} & \\ & V_{r_i}V_{r_i}^TS_{r_i}^{-1} \end{pmatrix} \tilde{A}_{ij}^{(k-2)} \\
&= \begin{pmatrix} (I_r+\Sigma_{l_ir_i})^{-\frac{1}{2}} & & \\ & (I_r-\Sigma_{l_ir_i})^{-\frac{1}{2}} & \\ & & I \end{pmatrix} \begin{pmatrix} \frac{Q_{l_i}^T}{\sqrt{2}} & \frac{Q_{r_i}^T}{\sqrt{2}} \\ \frac{Q_{l_i}^T}{\sqrt{2}} & \frac{-Q_{r_i}^T}{\sqrt{2}} \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} V_{l_i}^TS_{l_i}^{-1} & \\ & V_{r_i}^TS_{r_i}^{-1} \end{pmatrix} \tilde{A}_{ij}^{(k-2)} \\
&= \begin{pmatrix} I_{2r} \\ 0 \end{pmatrix} \begin{pmatrix} (I_r+\Sigma_{l_ir_i})^{-\frac{1}{2}} & \\ & (I_r-\Sigma_{l_ir_i})^{-\frac{1}{2}} \end{pmatrix} \begin{pmatrix} \frac{Q_{l_i}^T}{\sqrt{2}} & \frac{Q_{r_i}^T}{\sqrt{2}} \\ \frac{Q_{l_i}^T}{\sqrt{2}} & \frac{-Q_{r_i}^T}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} V_{l_i}^TS_{l_i}^{-1} & \\ & V_{r_i}^TS_{r_i}^{-1} \end{pmatrix} \tilde{A}_{ij}^{(k-2)}.
\end{aligned}$$

Denote $T_i = \begin{pmatrix} (I_r+\Sigma_{l_ir_i})^{-\frac{1}{2}} & \\ & (I_r-\Sigma_{l_ir_i})^{-\frac{1}{2}} \end{pmatrix} \begin{pmatrix} \frac{Q_{l_i}^T}{\sqrt{2}} & \frac{Q_{r_i}^T}{\sqrt{2}} \\ \frac{Q_{l_i}^T}{\sqrt{2}} & \frac{-Q_{r_i}^T}{\sqrt{2}} \end{pmatrix}$ which is a $2r \times 2r$ matrix. Multiplying the equation above by S_j^{-T} on the right and unraveling $\tilde{A}_{ij}^{(k-2)}$, we obtain

$$\begin{aligned} S_i^{-1}A_{ij}^{(k-1)}S_j^{-T} &= \begin{pmatrix} I_{2r} \\ 0 \end{pmatrix} T_i \begin{pmatrix} V_{l_i}^T S_{l_i}^{-1} & \\ & V_{r_i}^T S_{r_i}^{-1} \end{pmatrix} A_{ij}^{(k-2)} \begin{pmatrix} S_{l_j}^{-T}V_{l_j} & \\ & S_{r_j}^{-T}V_{r_j} \end{pmatrix} T_j^T \begin{pmatrix} I_{2r} & 0 \end{pmatrix} \\ &= \begin{pmatrix} I_{2r} \\ 0 \end{pmatrix} T_i M_{ij}^{(k-1)} T_j^T \begin{pmatrix} I_{2r} & 0 \end{pmatrix} \qquad (6.17) \end{aligned}$$

where $M_{ij}^{(k-1)} = \begin{pmatrix} M_{l_il_j}^{(k-1)} & M_{l_ir_j}^{(k-1)} \\ M_{r_il_j}^{(k-1)} & M_{r_ir_j}^{(k-1)} \end{pmatrix}$. From this equation, each scaled off-diagonal block $C_{ij}^{(k-1)}$ only has the top-left $2r \times 2r$ sub-block being nonzero. This structure of $C_{ij}^{(k-1)}$ arises from the ordering of the eigenvalues in the eigen-decomposition in Proposition 3. Evidently, the calculation of $C_{ij}^{(k-1)}$ only requires multiplications of matrices with dimensions $2r \times 2r$.

Let us also briefly consider the cost of choosing V_i to minimize the projection error $\|C_{ii^c}^{(k-1)} - V_iV_i^TC_{ii^c}^{(k-1)}\|_F$. With the constraint $\text{col}(V_i) \subset \text{col}(C_{ii^c}^{(k-1)})$, only the first $2r$ rows of V_i are nonzero. Thus both the storage and computational complexity of choosing V_i are similar to that of choosing R_i in the standard HSS construction procedure.

Calculation of $M_{ij}^{(k)}$ After finding V_i to compress the scaled off-diagonal blocks $C_{ij}^{(k-1)}$, matrix $M_{ij}^{(k)}$ at the kth level can be efficiently obtained by

$$M_{ij}^{(k)} = V_i^T \begin{pmatrix} I_{2r} \\ 0 \end{pmatrix} T_i M_{ij}^{(k-1)} T_j^T \begin{pmatrix} I_{2r} & 0 \end{pmatrix} V_j. \qquad (6.18)$$

Calculation of R_i By the definition of S_i in eq. (6.16) and eq. (6.15), the calculation of R_i through eq. (6.13) can be simplified as

$$R_i = \begin{pmatrix} V_{l_i}^T & \\ & V_{r_i}^T \end{pmatrix} \bar{S}_i V_i$$

$$= \begin{pmatrix} V_{l_i}^T & \\ & V_{r_i}^T \end{pmatrix} \begin{pmatrix} \frac{V_{l_i}Q_{l_i}}{\sqrt{2}} & \frac{V_{l_i}Q_{l_i}}{\sqrt{2}} & \tilde{V}_{l_i} & \\ \frac{V_{r_i}Q_{r_i}}{\sqrt{2}} & \frac{-V_{r_i}Q_{r_i}}{\sqrt{2}} & & \tilde{V}_{r_i} \end{pmatrix} \begin{pmatrix} (I_r+\Sigma_{l_ir_i})^{\frac{1}{2}} & & \\ & (I_r-\Sigma_{l_ir_i})^{\frac{1}{2}} & \\ & & I \end{pmatrix} V_i$$

$$= \begin{pmatrix} \frac{Q_{l_i}}{\sqrt{2}} & \frac{Q_{l_i}}{\sqrt{2}} & 0 & 0 \\ \frac{Q_{r_i}}{\sqrt{2}} & \frac{-Q_{r_i}}{\sqrt{2}} & 0 & 0 \end{pmatrix} \begin{pmatrix} (I_r+\Sigma_{l_ir_i})^{\frac{1}{2}} & & \\ & (I_r-\Sigma_{l_ir_i})^{\frac{1}{2}} & \\ & & I \end{pmatrix} V_i. \tag{6.19}$$

Due to the zeros in the leading matrix above, the calculation of R_i only requires the multiplication of a $2r \times 2r$ matrix by a $2r \times r$ matrix.

The efficient implementation of Method 2 is now given in Algorithm 12. With fixed rank r, the complexity of the algorithm is also $O(rn^2)$. We remark that for the general case where HSS off-diagonal blocks have different ranks, the above procedures can be easily adapted.

6.6 HSS approximation error analysis

For all the construction methods discussed above, the HSS approximation is constructed level-by-level as $A = A^{(0)} \Rightarrow A^{(1)} \Rightarrow ... \Rightarrow A^{(L)} = A_{\text{hss}}$ using update formula eq. (6.5) or eq. (6.10). The approximation error $\|A^{(0)} - A^{(L)}\|_F$ can be bounded as

$$\|A^{(0)} - A^{(L)}\|_F \leqslant \sum_{k=1}^{L} \|A^{(k-1)} - A^{(k)}\|_F, \tag{6.20}$$

which can be found by computing $\|A^{(k-1)} - A^{(k)}\|_F$ at each level.

6.6.1 Error estimation for the standard HSS method

For update formula eq. (6.5), we now provide an exact expression for $\|A^{(0)} - A^{(L)}\|_F^2$, demonstrating that the errors at each level are orthogonal to each other in a specific sense.

Algorithm 12 Method 2 with bottom-up level-by-level construction

```
Input: HSS rank $r$, original SPD matrix $A$
Output: SPD HSS approximation with generators $\{D_i\}, \{B_{ij}\}, \{U_i\}, \{R_i\}$
  At the leaf level
    • set $D_i = A_{ii}, \forall i \in \text{level}(1)$
    • compute the Cholesky decomposition $A_{ii} = S_i S_i^T, \forall i \in \text{level}(1)$
    • compute the scaled off-diagonal block $C_{ij}^{(0)} = S_i^{-1} A_{ij} S_j^{-T}, \forall i \neq j \in \text{level}(1)$
    • compute $V_i \in \mathbb{R}^{n_i \times r}, \forall i \in \text{level}(1)$ satisfying
      – columns of $V_i$ are orthonormal and col$(V_i) \subset$ col$(C_{ii^c}^{(0)})$
      – $V_i$ should minimize $\|C_{ii^c}^{(0)} - V_i V_i^T C_{ii^c}^{(0)}\|_F$
    • set $U_i = S_i V_i, \forall i \in \text{level}(1)$
    • compute $M_{ij}^{(1)} = V_i^T C_{ij}^{(0)} V_j, \forall i \neq j \in \text{level}(1)$
    • set $B_{ij} = M_{ij}^{(1)}$ for every pair of siblings $i, j \in \text{level}(1)$
  for $k = 2, 3, \ldots, L$ do
    • compute SVD $B_{l_i r_i} = Q_{l_i} \Sigma_{l_i r_i} Q_{r_i}^T, \forall i \in \text{level}(k)$
    • construct $T_i$ in eq. (6.17) by $\Sigma_{l_i r_i}, Q_{l_i}$ and $Q_{r_i}, \forall i \in \text{level}(k)$
    • compute the top-left $2r \times 2r$ nonzero sub-block of $C_{ij}^{(k-1)}$ as $\overline{C}_{ij}^{(k-1)} = T_i M_{ij}^{(k-1)} T_j^T$
  by eq. (6.17), $\forall i \neq j \in \text{level}(k)$
    • compute the first $2r$ rows $\overline{V}_i$ of $V_i, \forall i \in \text{level}(k)$ satisfying
      – columns of $\overline{V}_i$ are orthonormal and col$(\overline{V}_i) \subset$ col$(\overline{C}_{ii^c}^{(k-1)})$
      – $\overline{V}_i$ should minimize $\|\overline{C}_{ii^c}^{(k-1)} - \overline{V}_i \overline{V}_i^T \overline{C}_{ii^c}^{(k-1)}\|_F$
    • compute $R_i$ by eq. (6.19) using $\Sigma_{l_i r_i}, Q_{l_i}, Q_{r_i}$ and $\overline{V}_i, \forall i \in \text{level}(k)$
    • compute $M_{ij}^{(k)} = \overline{V}_i^T \overline{C}_{ij}^{(k-1)} \overline{V}_j, \forall i \neq j \in \text{level}(k)$
    • set $B_{ij} = M_{ij}^{(k)}$ for every pair of siblings $i, j \in \text{level}(k)$
  end for
```

Proposition 4. *For the recursive construction formula eq.* (6.5) *with the columns of* U_i *being orthonormal,*

$$\|A^{(0)} - A^{(L)}\|_F^2 = \sum_{k=1}^{L} \|A^{(k-1)} - A^{(k)}\|_F^2. \tag{6.21}$$

Proof. With a matrix partitioning by index subsets at the kth level, $0 < k < L$, the approximation error can be written as

$$\|A^{(0)} - A^{(L)}\|_F^2 = \sum_{i,j \in \text{level}(k)} \|A_{ij}^{(0)} - A_{ij}^{(k)} + A_{ij}^{(k)} - A_{ij}^{(L)}\|_F^2.$$

We will first show that each term in the summation above can be split as

$$\|A_{ij}^{(0)} - A_{ij}^{(k)} + A_{ij}^{(k)} - A_{ij}^{(L)}\|_F^2 = \|A_{ij}^{(0)} - A_{ij}^{(k)}\|_F^2 + \|A_{ij}^{(k)} - A_{ij}^{(L)}\|_F^2 \tag{6.22}$$

which then implies that

$$\|A^{(0)} - A^{(L)}\|_F^2 = \|A^{(0)} - A^{(k)}\|_F^2 + \|A^{(k)} - A^{(L)}\|_F^2, \quad 0 < k < L. \tag{6.23}$$

Consider any $i, j \in \text{level}(k)$. If $i = j$ or if i and j are siblings, then $A_{ij}^{(k)}$ will not be modified by the update formula at any of the subsequent levels. Thus, $A_{ij}^{(k)} = A_{ij}^{(L)}$ and eq. (6.22) holds true in this case. If i and j are not siblings, block $A_{ij}^{(L)}$ is obtained by the compression of some larger block with indices $I_p \times I_q$ where p, q are siblings and $I_i \subset I_p$, $I_j \subset I_q$. By the equation $A_{pq}^{(L)} = U_p U_p^T A_{pq}^{(0)} U_q U_q^T$ and the nested basis property for U_p, it can be proved that $\text{col}(A_{ij}^{(L)}) \subset \text{col}(U_i)$. Symmetrically, the row space of $A_{ij}^{(L)}$ satisfies $\text{col}((A_{ij}^{(L)})^T) \subset \text{col}(U_j)$.

From the construction process, $A_{ij}^{(k)} = U_i U_i^T A_{ij}^{(k-1)} U_j U_j^T = U_i U_i^T A_{ij}^{(0)} U_j U_j^T$ and thus $A_{ij}^{(0)} - A_{ij}^{(k)}$ can be written as

$$A_{ij}^{(0)} - A_{ij}^{(k)} = A_{ij}^{(0)} - U_i U_i^T A_{ij}^{(0)} + U_i U_i^T (A_{ij}^{(0)} - A_{ij}^{(0)} U_j U_j^T).$$

Notice that the columns of $A_{ij}^{(0)} - U_i U_i^T A_{ij}^{(0)}$ are orthogonal to $\text{col}(U_i)$ and the rows of $A_{ij}^{(0)} - A_{ij}^{(0)} U_j U_j^T$ are orthogonal to $\text{col}(U_j)$. Meanwhile, the columns and rows of $A_{ij}^{(k)} - A_{ij}^{(L)}$ are within $\text{col}(U_i)$ and $\text{col}(U_j)$ respectively. By the Pythagorean theorem and properties mentioned above, the splitting in eq. (6.22) can be proved as

$$\begin{aligned}
&\|A_{ij}^{(0)} - A_{ij}^{(k)} + A_{ij}^{(k)} - A_{ij}^{(L)}\|_F^2 \\
&= \|A_{ij}^{(0)} - U_i U_i^T A_{ij}^{(0)}\|_F^2 + \|U_i U_i^T (A_{ij}^{(0)} - A_{ij}^{(0)} U_j U_j^T) + A_{ij}^{(k)} - A_{ij}^{(L)}\|_F^2 \\
&= \|A_{ij}^{(0)} - U_i U_i^T A_{ij}^{(0)}\|_F^2 + \|U_i U_i^T (A_{ij}^{(0)} - A_{ij}^{(0)} U_j U_j^T)\|_F^2 + \|A_{ij}^{(k)} - A_{ij}^{(L)}\|_F^2
\end{aligned}$$

$$= \|A_{ij}^{(0)} - A_{ij}^{(k)}\|_F^2 + \|A_{ij}^{(k)} - A_{ij}^{(L)}\|_F^2,$$

from which eq. (6.23) follows.

Now, it can be observed that in the level-by-level construction process, $A^{(L)}$ can also be regarded as an HSS approximation to $A^{(l)}$ for any $0 < l < L$. Thus, eq. (6.23) also holds true when replacing index 0 by l, allowing us to recursively apply eq. (6.23) for l from 0 to $(L-2)$ with $k = l+1$ to prove the proposition. $\square$

It is worth mentioning that this proposition for HSS approximations can be easily extended to analyze the error of general $\mathcal{H}^2$ approximations with the projection method. This proposition is also closely related the error analysis of $\mathcal{H}^2$ approximations in Refs. [3, 77].

For the HSS approximation with update formula eq. (6.5), an upper bound for the square of the approximation error for one stage can be obtained as

$$\begin{aligned}
\|A^{(k-1)} - A^{(k)}\|_F^2 &= \sum_{i\neq j\in \text{level}(k)} \|A_{ij}^{(k-1)} - U_i U_i^T A_{ij}^{(k-1)} U_j U_j^T\|_F^2 \\
&= \sum_{i\neq j\in \text{level}(k)} \|A_{ij}^{(k-1)} - U_i U_i^T A_{ij}^{(k-1)}\|_F^2 + \|U_i U_i^T (A_{ij}^{(k-1)} - A_{ij}^{(k-1)} U_j U_j^T)\|_F^2 \\
&\leqslant \sum_{i\neq j\in \text{level}(k)} \|A_{ij}^{(k-1)} - U_i U_i^T A_{ij}^{(k-1)}\|_F^2 + \|A_{ij}^{(k-1)} - A_{ij}^{(k-1)} U_j U_j^T\|_F^2 \\
&= 2 \sum_{i\in \text{level}(k)} \|A_{ii^c}^{(k-1)} - U_i U_i^T A_{ii^c}^{(k-1)}\|_F^2. \qquad (6.24)
\end{aligned}$$

Meanwhile, from the 2nd line above, $\|A^{(k-1)} - A^{(k)}\|_F^2$ can be bounded from below as

$$\|A^{(k-1)} - A^{(k)}\|_F^2 \geqslant \sum_{i\in \text{level}(k)} \|A_{ii^c}^{(k-1)} - U_i U_i^T A_{ii^c}^{(k-1)}\|_F^2.$$

With these lower and upper bounds, the strategy of choosing each U_i to minimize the projection error of HSS block rows $A_{ii^c}^{(k-1)}$ is justified in that the strategy minimizes the square of the approximation error at each stage within a factor of 2.

Instead of a fixed rank r for U_i, the compression may be performed with a threshold ε such that $\|A_{ii^c}^{(k-1)} - U_i U_i^T A_{ii^c}^{(k-1)}\|_F^2 \leqslant \varepsilon^2$. In terms of ε, the upper bound on the square of the approximation error for one stage is

$$\|A^{(k-1)} - A^{(k)}\|_F^2 \leqslant 2|\text{level}(k)|\varepsilon^2 = 2^{L-k+2}\varepsilon^2,$$

giving a bound on the error for the entire approximation as

$$\|A - A^{(L)}\|_F \leqslant \left(\sum_{k=1}^{L} 2^{(L-k+2)}\varepsilon^2\right)^{\frac{1}{2}} = \varepsilon\left(2^{L+2} - 4\right)^{\frac{1}{2}} \approx 2\varepsilon(n/n_0)^{\frac{1}{2}},$$

where n_0 is the average size of the index subsets $\{I_i\}_{i\in\text{level}(1)}$ at the leaf level. Meanwhile, we have the approximate lower bound $\|A - A^{(L)}\|_F \geqslant \sqrt{2}\varepsilon(n/n_0)^{\frac{1}{2}}$, if the compression satisfies $\|A_{ii^c}^{(k-1)} - U_i U_i^T A_{ii^c}^{(k-1)}\|_F^2 \approx \varepsilon^2$.

6.6.2 Error estimation for Method 1

In the standard HSS method, U_i is chosen at each level to minimize the projection error $\|A_{ii^c}^{(k-1)} - UU^T A_{ii^c}^{(k-1)}\|_F$ such that the columns of U_i are orthonormal and, for non-leaf levels, U_i satisfies the nested basis property. In Method 1, we have the additional requirement that the columns of U_i are eigenvectors of $A_{ii}^{(k-1)}$. It is clear that the achievable minimum of Method 1 will not be better than that of the standard HSS method. From the error analysis of the constrained optimization problem in Section 6.4.2, the minimum projection error can be as large as $(1 - \frac{r}{n_i})^{\frac{1}{2}}\|A_{ii^c}^{(k-1)}\|_F$. This worst case projection error can be much worse than the bounds on the projection error for the standard HSS method.

However, for kernel matrices defined by many SPD smooth kernel functions, Method 1 may sometimes give good approximations as measured by $\|A_{ii^c}^{(k-1)} - UU^T A_{ii^c}^{(k-1)}\|_F$. A plausible explanation of when this might happen is as follows.

Based on the Mercer's theorem, any SPD smooth kernel $K(x, y)$ on a compact domain

Ω has an eigenfunction expansion as,

$$K(x,y) = \sum_{k=1}^{\infty} \lambda_k \phi_k(x)\phi_k(y), \quad \forall x, y \in \Omega,$$

where $\{\phi_k(x)\}$ are orthonormal in $L_2(\Omega)$ and $\{\lambda_k\}$ decreases monotonically to zero. Denote the sum of the first r terms as $K^{(r)}(x,y)$ and the remainder term as $R^{(r)}(x,y)$. Their L_2-norms in $\Omega \times \Omega$ are $\|K^{(r)}\|_{L_2}^2 = \sum_{k=1}^{r} \lambda_k^2$ and $\|R^{(r)}\|_{L_2}^2 = \sum_{k=r+1}^{\infty} \lambda_k^2$.

Consider two point sets $I = \{x_j\}_{j=1}^p$ and $J = \{y_j\}_{j=1}^q$ in domains Ω_1 and Ω_2 respectively. Define $\Omega = \Omega_1 \cup \Omega_2$ for the above eigenfunction decomposition. Choose r such that $\|R^{(r)}\|_{L_2}/\|K^{(r)}\|_{L_2}$ is relatively small, say 10^{-2}. We can write the diagonal block A_{II} and off-diagonal block A_{IJ} as,

$$\begin{aligned}
A_{II} &= (K(x_j, x_l))_{x_j, x_l \in I} = K_{II}^{(r)} + R_{II}^{(r)} \\
&= \begin{pmatrix} \lambda_1\phi_1(x_1) & \dots & \lambda_r\phi_r(x_1) \\ \lambda_1\phi_1(x_2) & \dots & \lambda_r\phi_r(x_2) \\ \vdots & & \vdots \\ \lambda_1\phi_1(x_p) & \dots & \lambda_r\phi_r(x_p) \end{pmatrix} \begin{pmatrix} \phi_1(x_1) & \dots & \phi_r(x_1) \\ \phi_1(x_2) & \dots & \phi_r(x_2) \\ \vdots & & \vdots \\ \phi_1(x_p) & \dots & \phi_r(x_p) \end{pmatrix}^T + (R^{(r)}(x_j, x_l))_{x_j, x_l \in I}, \\
A_{IJ} &= (K(x_j, y_l))_{x_j \in I, y_l \in J} = K_{IJ}^{(r)} + R_{IJ}^{(r)} \\
&= \begin{pmatrix} \lambda_1\phi_1(x_1) & \dots & \lambda_r\phi_r(x_1) \\ \lambda_1\phi_1(x_2) & \dots & \lambda_r\phi_r(x_2) \\ \vdots & & \vdots \\ \lambda_1\phi_1(x_p) & \dots & \lambda_r\phi_r(x_p) \end{pmatrix} \begin{pmatrix} \phi_1(y_1) & \dots & \phi_r(y_1) \\ \phi_1(y_2) & \dots & \phi_r(y_2) \\ \vdots & & \vdots \\ \phi_1(y_q) & \dots & \phi_r(y_q) \end{pmatrix}^T + (R^{(r)}(x_j, y_l))_{x_j \in I, y_l \in J}.
\end{aligned}$$

From the viewpoint of numerical integration, $\|R_{II}^{(r)}\|_F^2$ can be roughly estimated as

$$\|R_{II}^{(r)}\|_F^2 = \frac{|I|^2}{|\Omega_1 \times \Omega_1|} \sum \frac{|\Omega_1 \times \Omega_1|}{|I|^2} |R^{(r)}(x_j, x_l)|^2 \approx \frac{|I|^2}{|\Omega_1 \times \Omega_1|} \|R^{(r)}|_{\Omega_1 \times \Omega_1}\|_{L_2}^2.$$

Thus, similar to the other matrices above, it is likely to hold that

$$\begin{aligned} \|R_{II}^{(r)}\|_F &\sim O(|I|\|R^{(r)}|_{\Omega_1\times\Omega_1}\|_{L_2}) & \|K_{II}^{(r)}\|_F &\sim O(|I|\|K^{(r)}|_{\Omega_1\times\Omega_1}\|_{L_2}) \\ \|R_{IJ}^{(r)}\|_F &\sim O(\sqrt{|I||J|}\|R^{(r)}|_{\Omega_1\times\Omega_2}\|_{L_2}) & \|K_{IJ}^{(r)}\|_F &\sim O(\sqrt{|I||J|}\|K^{(r)}|_{\Omega_1\times\Omega_2}\|_{L_2}). \end{aligned}$$

Call Φ the matrix that is common in the above expressions for A_{II} and A_{IJ}. If we assume that both $\|K^{(r)}|_{\Omega_1\times\Omega_1}\|_{L_2}$ and $\|K^{(r)}|_{\Omega_1\times\Omega_2}\|_{L_2}$ are of the same scale as $\|K^{(r)}\|_{L_2}$, $\|R_{II}^{(r)}\|_F$ will be relatively small compared to $\|K_{II}^{(r)}\|_F$. Thus, based on $A_{II} = K_{II}^{(r)} + R_{II}^{(r)}$ and $K_{II}^{(r)}$ being rank r, it is likely that the rank-r principal eigenvector space of A_{II} is close to col$(K_{II}^{(r)}) =$ col(Φ). Similarly, the rank-r principal left-singular vector space of A_{IJ} is also likely close to col$(K_{IJ}^{(r)}) =$ col(Φ). As a result, it is possible that the rank-r principal eigenvector space of A_{II} and the rank-r principal left-singular vector space of A_{IJ} are close. If the spaces are close, then choosing U as some principal orthonormal eigenvectors of A_{II} may give a small projection error $\|A_{IJ} - UU^TA_{IJ}\|_F$.

The above assumption about $\|K^{(r)}|_{\Omega_1\times\Omega_1}\|_{L_2}$ and $\|K^{(r)}|_{\Omega_1\times\Omega_2}\|_{L_2}$ may not hold in general. A common example is for $K(x,y) = e^{-\|x-y\|^2}$ with two domains, Ω_1 and Ω_2, that are far apart. As a good approximation of $K(x,y)$, $K^{(r)}(x,y)$ on $\Omega_1 \times \Omega_2$ should have much smaller L_2 norm than on $\Omega_1 \times \Omega_1$. However, it is worth noting that, in this case, a highly accurate approximation to A_{IJ} may not be necessary for preconditioning. Numerical tests are now presented to illustrate the above argument.

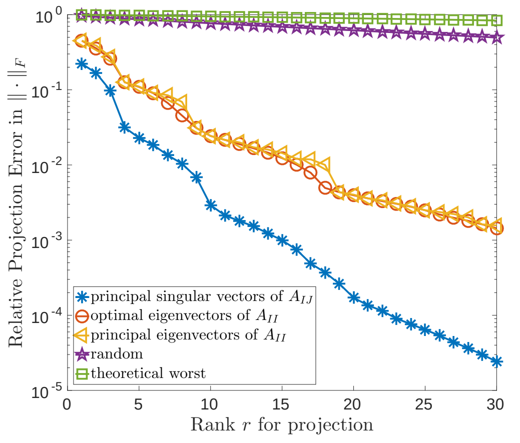
Consider two clusters I and J where each contains 100 uniformly randomly distributed points within a unit cube. The centers of the two cubes lie at $(0,0,0)$ and $(L,0,0)$. For $K(x,y) = e^{-\|x-y\|^2}$, the diagonal block A_{II} and off-diagonal block A_{IJ} are defined as above. The relative projection error $\|A_{IJ} - UU^TA_{IJ}\|_F/\|A_{IJ}\|_F$ vs. rank r is shown in Figure 6.3 with two different cluster locations $(L,0,0)$ for cluster J. The columns of U are chosen as:

- r left singular-vectors of A_{IJ} associated with the largest singular values,

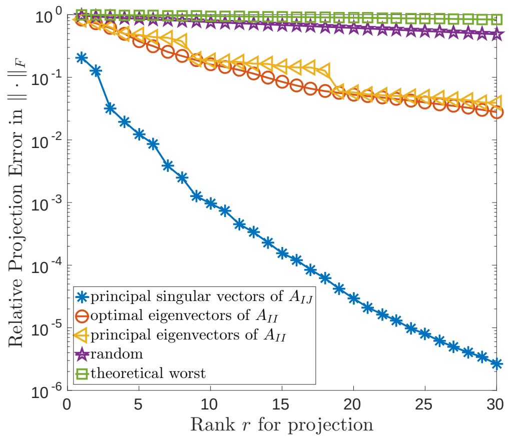
- optimal r orthonormal eigenvectors of A_{II} chosen by solving Problem 1,
- r orthonormal eigenvectors of A_{II} associated with the largest eigenvalues,
- optimal r columns chosen by solving Problem 1 using a random orthogonal matrix generated by the QR decomposition of a Gaussian random matrix. The mean value from 10 tests is plotted in the figure.

In addition, the curve $(1 - \frac{r}{n})^{\frac{1}{2}}$ for the worst case error is also plotted for comparison.

Figure 6.3 shows that using eigenvectors of A_{II} gives good approximation errors especially when compared to using columns of a random orthogonal matrix. In addition, the closeness between using optimal and principal eigenvectors of A_{II}, as well as the difference between results for nearby and distant clusters, both support the argument above.



(a) $(L, 0, 0) = (1, 0, 0)$, $\frac{\|A_{IJ}\|_F}{\|A_{II}\|_F} = 0.53$



(b) $(L, 0, 0) = (3, 0, 0)$, $\frac{\|A_{IJ}\|_F}{\|A_{II}\|_F} = 2.2\text{e-}3$

Figure 6.3: Relative projection error with $K(x, y) = e^{-\|x-y\|^2}$ for the cases of (a) nearby and (b) distant clusters.

6.6.3 Error estimation for Method 2

Due to the scaling of the off-diagonal blocks, Proposition 4 does not work for Method 2. Here, we directly estimate the HSS approximation error of Method 2 using the inequality eq. (6.20). The square of the error at each level k can be bounded as

$$\begin{aligned}
\|A^{(k-1)} - A^{(k)}\|_F^2 &= \sum_{i\neq j\in \text{level}(k)} \|A_{ij}^{(k-1)} - S_iV_iV_i^TS_i^{-1}A_{ij}^{(k-1)}S_j^{-T}V_jV_j^TS_j^T\|_F^2 \\
&= \sum_{i\neq j\in \text{level}(k)} \|S_i(C_{ij}^{(k-1)} - V_iV_i^TC_{ij}^{(k-1)}V_jV_j^T)S_j^T\|_F^2 \\
&\leqslant \sum_{i\neq j\in \text{level}(k)} \|S_i\|_2^2\,\|S_j\|_2^2\,\|C_{ij}^{(k-1)} - V_iV_i^TC_{ij}^{(k-1)}V_jV_j\|_F^2 \\
&\leqslant \max_{i\in\text{level}(k)} \|S_i\|_2^4 \sum_{i\neq j\in \text{level}(k)} \|C_{ij}^{(k-1)} - V_iV_i^TC_{ij}^{(k-1)}V_jV_j\|_F^2 \\
&\leqslant 2\max_{i\in\text{level}(k)} \|A_{ii}^{(k-1)}\|_2^2 \sum_{i\in \text{level}(k)} \|C_{ii^c}^{(k-1)} - V_iV_i^TC_{ii^c}^{(k-1)}\|_F^2.
\end{aligned}$$

The last inequality above is obtained by the same method used in eq. (6.24).

Assume V_i is chosen to satisfy $\|C_{ii^c}^{(k-1)} - V_iV_i^TC_{ii^c}^{(k-1)}\|_F^2 \leqslant \varepsilon^2$ with error threshold ε. The remaining part is to estimate $\max_{i\in\text{level}(k)}\|A_{ii}^{(k-1)}\|_2$ at each level. For any node $i \in \text{level}(k)$, eq. (6.14) gives the diagonal block as

$$A_{ii}^{(k-1)} = \begin{pmatrix} S_{l_i} & \\ & S_{r_i} \end{pmatrix} \begin{pmatrix} I & V_{l_i}B_{l_ir_i}V_{r_i}^T \\ V_{r_i}B_{l_ir_i}^TV_{l_i}^T & I \end{pmatrix} \begin{pmatrix} S_{l_i}^T & \\ & S_{r_i}^T \end{pmatrix}.$$

As $A_{ii}^{(k-1)}$ is positive definite, the matrix in the middle is also SPD and its largest eigenvalue should be less than 2 based on Proposition 3. Thus, $\|A_{ii}^{(k-1)}\|_2$ can be bounded as

$$\begin{aligned}
\|A_{ii}^{(k-1)}\|_2 &\leqslant \left\| \begin{pmatrix} S_{l_i} & \\ & S_{r_i} \end{pmatrix} \right\|_2^2 \left\| \begin{pmatrix} I & V_{l_i}B_{l_ir_i}V_{r_i}^T \\ V_{r_i}B_{l_ir_i}^TV_{l_i}^T & I \end{pmatrix} \right\|_2 \\
&\leqslant 2\max(\|A_{l_il_i}^{(k-2)}\|_2, \|A_{r_ir_i}^{(k-2)}\|_2) \\
&\leqslant 2\max_{j\in\text{level}(k-1)} \|A_{jj}^{(k-2)}\|_2 \leqslant \ldots \leqslant 2^{k-1}\max_{j\in\text{level}(1)}\|A_{jj}^{(0)}\|_2 \leqslant 2^{k-1}\|A\|_2.
\end{aligned}$$

The square of the approximation error can then be bounded as

$$\|A^{(k-1)} - A^{(k)}\|_F^2 \leqslant 2^{2k-1}\|A\|_2^2|\text{level}(k)|\varepsilon^2 = 2^{L+k}\|A\|_2^2\varepsilon^2.$$

Finally, the HSS approximation error of Method 2 satisfies

$$\|A - A^{(L)}\|_F \leqslant 2^{L/2}\|A\|_2\varepsilon\sum_{k=1}^{L} 2^{k/2} = \|A\|_2\varepsilon\frac{\sqrt{2}}{\sqrt{2}-1}(2^L - 2^{L/2}) \approx \frac{\sqrt{2}}{\sqrt{2}-1}\frac{n}{n_0}\|A\|_2\varepsilon$$

where n_0 is the average size of the index subsets $\{I_i\}_{i\in\text{level}(1)}$ at the leaf level.

Note that the compression in Method 2 is applied to the scaled off-diagonal block $C_{ii^c}^{(k-1)}$, i.e., finding $V_i \in \mathbb{R}^{n_i\times r}$ that minimizes $\|C_{ii^c}^{(k-1)} - V_iV_i^TC_{ii^c}^{(k-1)}\|_F^2$. Assuming that the original HSS block row $A_{ii^c}^{(k-1)}$ has fast-decaying singular values, the scaled block row $C_{ii^c}^{(k-1)}$ may not necessarily have this property. Hence, the rank of the approximation for a given ε may be large. Heuristically, choosing V_i to minimize the approximation error $\|A^{(k)} - A^{(k-1)}\|_F^2$ at each stage might be a better method, but an efficient way to do this is currently unclear.

6.7 Numerical results

As example applications, we are concerned with the preconditioning of symmetric positive definite matrices by HSS approximations in two general problems: solving linear systems $Ax = b$ using the preconditioned conjugate gradient (PCG) method and sampling correlated random vectors $y \sim \mathcal{N}(0, A)$ using a preconditioned Lanczos process [78].

To apply the preconditioners, we use the symmetric ULV factorization [26]. The following settings are shared for all experiments:

- *Hierarchical partitioning of points:* A full binary partition tree is constructed by recursively partitioning a set of points using the principal components analysis algorithm such that leaf nodes have no more than 100 points.

- *Rank of HSS block rows:* Two settings: 1) The rank of U_i is fixed as a constant, r. 2) The rank of U_i is adaptively chosen using a constant relative error threshold τ in the associated compression of $A_{ii^c}^{(k-1)}$ or $C_{ii^c}^{(k-1)}$.

- *Algorithm for the projection method:* The randomized algorithm in [57] is used to estimate the principal column space of the compression target matrix with fixed rank or relative error threshold in all the HSS approximations.

- *Stopping criteria:* A threshold $\varepsilon = 10^{-8}$ is applied for all the experiments. PCG stops at iteration i when the relative reduction of the residual satisfies $\|r_i\|/\|r_0\| \leqslant \varepsilon$ and the Lanczos method stops when the relative difference between two consecutive iterates satisfies $\|y_{i+1} - y_i\|/\|y_i\| \leqslant \varepsilon$.

- *Methods:* Four methods are tested: Block Jacobi, Standard HSS with the projection method, Method 1, and Method 2, denoted as BJ, HSS, SPDHSS1, and SPDHSS2, respectively. The Block Jacobi preconditioner is composed of the diagonal blocks associated with the leaf nodes of the partition tree. All methods are implemented in Matlab.

6.7.1 Inverse multiquadric kernel

The inverse multiquadric kernel is a non-compact radial basis function and is defined as

$$K(x, y) = \frac{1}{\sqrt{1 + c|x - y|^2}}, \quad x, y \in \mathbb{R}^d,$$

where c is a parameter that controls the flatness of the kernel function.

To maintain constant point density, N points are randomly and uniformly distributed in a cube with edge length $\sqrt[3]{N}$ in 3D. We use parameters $c = 0.5$ with fixed rank $r = 50$ or relative error threshold τ= 1e-2. A test with τ = 8e-2 only for SPDHSS2 is included. Results are shown for different values of N in Table 6.1. No preconditioning results can be shown for standard HSS as these approximations are not positive definite in any of our settings. Also, BiCGStab with the standard HSS approximation as a preconditioner usually does not converge within $2N$ steps for most of the test settings.

With fixed rank r, the approximation errors for both SPDHSS1 and SPDHSS2 are always larger than for standard HSS, as expected. The iteration counts for SPDHSS1 and SPDHSS2 both increase with N, as the compression of larger blocks with fixed rank gives less accurate HSS approximations. In addition, SPDHSS1 requires less construction time than standard HSS which could also have been expected.

With τ=1e-2, the iteration count using SPDHSS2 is scalable for both solving and sampling. This is at the price of much larger ranks for off-diagonal blocks as reflected by the storage cost. The results suggest that, in this example, the scaled HSS block rows $C_{ii^c}^{(k-1)}$ in Method 2 have slower-decaying singular values than HSS block rows $A_{ii^c}^{(k-1)}$ in standard HSS. Meanwhile, SPDHSS2 with τ=8e-2 obtains a better balance between construction cost and preconditioner effectiveness.

To compare with standard HSS, we increase the rank r for the cases with $N = 8000$, 12000, 16000 such that standard HSS approximations are also positive definite. Results are shown in Table 6.2. An interesting phenomenon, which also appears with larger r and with other kernels, is that although standard HSS has a smaller approximation error, it has worse preconditioning performance compared to SPDHSS2.

The HSS approximation time and storage cost vs. N are shown in Figure 6.4. With fixed rank, SPDHSS1 and SPDHSS2 both have quadratic computational complexities. With fixed τ, these super-linear storage results indicate that ranks of the off-diagonal blocks in both methods are related to N.

6.7.2 RPY kernel

The Rotne-Prager-Yamakawa (RPY) kernel $D(x, y) : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^{3\times3}$ is a positive definite tensor function that describes the hydrodynamic interactions between particles in a

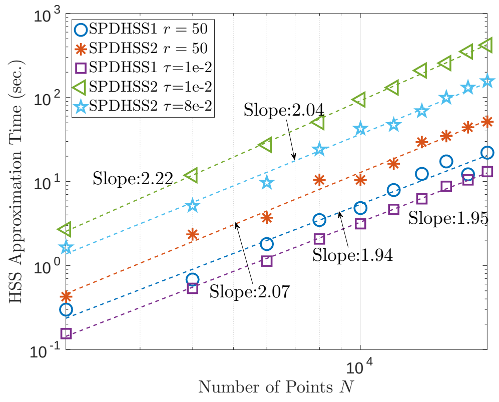
Table 6.1: Numerical results for the inverse multiquadric kernel with $c = 0.5$. The iteration count and consumed time for solving and sampling, relative approximation error $\|A_{\text{apprx}} - A\|_F / \|A\|_F$, construction time of preconditioners (including HSS approximation and symmetric ULV decomposition), and storage cost of the ULV factors of the HSS approximations are presented.

		N	4000	8000	12000	16000	20000
Solving iter/time (sec.)		Unprecond.	5970/20.8	11542/144.8	15708/372.2	15973/741.3	22240/1533.6
		BJ	757/3.1	1299/17.9	1248/34.8	1400/66.8	1540/111.9
	r=50	SPDHSS1	253/3.9	375/15.0	475/28.0	463/38.6	483/51.9
		SPDHSS2	195/2.9	305/10.1	373/20.9	348/29.7	430/46.8
	τ=1e-2	SPDHSS1	184/2.5	294/10.0	329/19.2	308/29.4	348/43.0
		SPDHSS2	11/0.3	11/0.8	15/2.0	11/2.5	13/3.8
	τ=8e-2	SPDHSS2	48/1.0	58/3.4	99/8.7	64/9.3	78/15.7
Sampling iter/time (sec.)		Unprecond.	567/12.1	670/36.2	962/107.1	912/129.9	989/185.8
		BJ	269/2.7	374/12.7	283/15.9	308/23.1	352/41.0
	r=50	SPDHSS1	136/3.3	163/8.2	181/11.9	164/16.9	183/26.0
		SPDHSS2	113/2.3	126/5.9	156/11.3	155/15.6	145/20.0
	τ=1e-2	SPDHSS1	104/1.7	146/5.8	138/9.5	129/14.0	126/17.2
		SPDHSS2	9/0.3	9/0.8	11/1.7	8/2.1	10/3.3
	τ=8e-2	SPDHSS2	33/0.8	37/2.4	53/5.1	35/5.5	41/8.8
Relative error	r=50	SPDHSS1	6.3e-2	7.7e-2	8.7e-2	9.1e-2	9.6e-2
		SPDHSS2	6.9e-2	8.3e-2	1.1e-1	1.1e-1	1.1e-1
		HSS	1.1e-2	1.6e-2	2.1e-2	2.4e-2	2.7e-2
	τ=1e-2	SPDHSS1	2.7e-2	3.0e-2	3.1e-2	3.2-e2	3.2e-2
		SPDHSS2	1.3e-3	1.2e-3	1.2e-3	1.2e-3	1.2e-3
		HSS	2.4e-2	2.6e-2	2.8e-2	3.0e-2	3.0e-2
	τ=8e-2	SPDHSS2	1.6e-2	1.4e-2	1.6e-2	1.5-e2	1.8e-2
Construct. time (sec.) apprx/ulv	r=50	SPDHSS1	0.7/0.1	3.5/0.2	8.0/0.3	17.4/0.5	22.1/0.5
		SPDHSS2	2.3/0.1	10.4/0.3	16.4/0.4	34.8/0.5	51.4/0.4
		HSS	1.7/-	7.4/-	13.4/-	20.4/-	34.6/-
	τ=1e-2	SPDHSS1	0.5/0.1	2.1/0.3	4.6/0.5	8.8/0.6	13.1/0.7
		SPDHSS2	11.9/0.4	50.8/1.2	130.9/2.5	257.4/4.3	421.1/6.4
		HSS	3.0/-	11.1/-	23.3/-	48.5/-	61.5/-
	τ=8e-2	SPDHSS2	5.1/0.2	24.2/0.7	46.9/1.1	98.9/1.8	157.3/2.7
Storage (MB)		Dense Matrix	122	488	1098	1953	3051
		$r = 50^*$	14	28	41	58	68
	τ=1e-2	SPDHSS1	20	45	75	101	134
		SPDHSS2	87	248	485	744	1062
	τ=8e-2	SPDHSS2	44	119	214	334	479

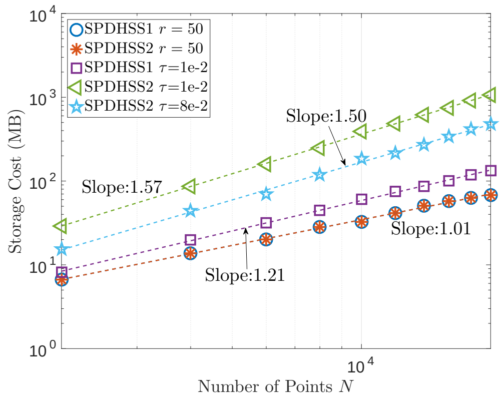
*With a fixed rank, ULV factors of all the HSS approximations have the same storage cost.

Table 6.2: Comparison among different preconditioning methods for the inverse multiquadric kernel for three problem sizes when the standard HSS approximation is positive definite. The relative error (relerr), iteration count for solving (solve) and sampling (sample) are shown. The relative error does not appear to be always correlated with iteration count.

	$r = 518$, $N = 8000$			$r = 665$, $N = 12000$			$r = 730$, $N = 16000$		
	relerr	solve	sample	relerr	solve	sample	relerr	solve	sample
HSS	3.0e-5	12	10	5.3e-5	19	13	7.4e-5	22	16
SPDHSS1	1.2e-2	147	81	1.2e-2	174	85	1.3e-2	143	66
SPDHSS2	8.8e-4	7	6	2.1e-3	7	6	4.2e-3	10	8



(a) HSS approximation time



(b) ULV factor storage

Figure 6.4: HSS approximation time and ULV factor storage cost vs. N for the inverse multiquadric kernel with $c = 0.5$. Linear fittings are drawn with dashed lines.

viscous fluid. We previously used this kernel in coarse-grained macromolecular simulations [79, 80], and the need to construct positive definite preconditioners for sampling in this application was the original motivation for this work. The RPY kernel is defined as

$$D(x,y) = \begin{cases} \frac{k_BT}{6\pi\eta a} I_3 & \text{if } x = y \\ \frac{k_BT}{8\pi\eta|r|}\left[\left(I_3 + \frac{rr^T}{|r|^2}\right) + \frac{2a^2}{|r|^2}\left(\frac{1}{3}I_3 - \frac{rr^T}{|r|^2}\right)\right] & \text{if } |x-y| \geqslant 2a \\ \frac{k_BT}{6\pi\eta a}\left[\left(1 - \frac{9}{32}\frac{|r|}{a}\right) I_3 + \frac{3}{32}\frac{|r|}{a}\frac{rr^T}{|r|^2}\right] & \text{if } |x-y| < 2a \end{cases}, \text{ with } r = x - y,$$

where k_B, T, η are fixed physical quantities and a is the radius of the particles. In this test, $a = 1$ and constant $\frac{k_BT}{6\pi\eta a} = 1$. We place N non-overlapping particles randomly inside a cube with a width chosen such that the volume fraction is 0.3. Note that RPY kernel matrix with N particles is of size $3N \times 3N$.

Results are presented in Table 6.3. The standard HSS approximations with $r = 50$ or τ=1e-2 are not positive definite for any tested N. This is a challenging problem, as none of the methods give scalable preconditioning performance. Although the construction cost is high in some cases, it can be amortized over the many sample vectors that must be computed for the same matrix in real applications.

Table 6.3: Numerical results for RPY kernel with particle volume fraction 0.3.

		N	4000	6000	8000	10000	12000
Sampling iter/time (sec.)		Unprecond.	105/4.4	116/10.2	123/15.3	131/24.9	136/36.4
		BJ	113/5.2	127/12.0	128/16.9	136/27.4	143/40.0
	r=50	SPDHSS1	68/4.3	75/9.6	81/13.3	89/21.4	98/33.5
		SPDHSS2	65/4.0	80/10.6	86/14.3	95/22.8	103/35.3
	τ=2e-2	SPDHSS1	24/3.2	26/5.7	28/8.8	29/12.3	30/16.4
	τ=8e-2	SPDHSS2	19/2.4	22/4.7	24/7.7	24/11.0	26/17.0
Relative error	r=50	SPDHSS1	2.2e-1	2.3e-1	2.4e-1	2.4e-1	2.5e-1
		SPDHSS2	1.9e-1	1.9e-1	2.0e-1	2.1e-1	2.2e-1
	τ=2e-2	SPDHSS1	5.2e-2	5.4e-2	5.8e-2	5.8e-2	6.0e-2
	τ=8e-2	SPDHSS2	2.7e-2	2.9e-2	2.8e-2	2.6e-2	2.9e-2
Construct. time (sec.) apprx/ulv	r=50	SPDHSS1	3.1/0.19	6.6/0.36	12.0/0.39	16.9/0.50	24.6/0.77
		SPDHSS2	6.7/0.19	16.0/0.37	25.8/0.39	37.3/0.51	54.5/0.65
	τ=2e-2	SPDHSS1	9.9/2.11	20.4/3.44	35.8/4.59	50.4/6.14	72.7/7.52
	τ=8e-2	SPDHSS2	87.8/1.89	178.5/3.22	349.6/5.23	581.3/7.71	866.0/12.96
Storage (MB)		Dense Matrix	1099	2472	4395	6867	9888
		$r = 50$	45	76	92	129	155
	τ=2e-2	SPDHSS1	461	731	955	1251	1509
	τ=8e-2	SPDHSS2	380	645	1008	1455	1837

6.7.3 Boundary integral equation

Consider the 3D Laplace equation in a bounded Lipschitz domain Ω with Dirichlet condition $u = u_D$ on $\partial\Omega$. The indirect boundary integral equation [77] for this problem leads to a linear system $Vx = b$ with

$$V_{ij} = \int_{\tau_i}\int_{\tau_j} \frac{1}{4\pi\|x-y\|_2}\mathrm{d}x\mathrm{d}y, \quad b_i = \int_{\tau_i} u_D(x)\mathrm{d}x, \tag{6.25}$$

where $\{\tau_i\}$ is a partitioning of $\partial\Omega$ and matrix V is known to be always SPD.

In this test, $\partial\Omega$ is the unit sphere and we solve the linear system above with uniform triangulations with different numbers of triangles, N. Entries of b are randomly selected from $[-1, 1]$. Results are shown in Table 6.4.

Table 6.4: Numerical results for the boundary integral equation over the unit sphere with different N. The meshes and matrices are constructed by the package BEM++[81].

		N	3206	6500	9008	12600	34184
Solving iter/time (sec.)		Unprecond.	80/0.32	92/0.96	104/1.92	106/3.56	140/34.32
		BJ	35/0.16	37/0.40	39/0.79	44/1.58	51/11.14
	r=50	SPDHSS1	20/0.23	21/0.67	22/0.91	23/1.52	25/7.36
		SPDHSS2	11/0.14	13/0.43	13/0.54	15/1.06	18/5.34
		HSS	11/0.14	13/0.35	14/0.58	15/1.14	17/5.55
	τ=1e-2	SPDHSS1	9/0.36	10/1.28	10/1.74	10/3.00	11/17.26
		SPDHSS2	10/0.13	11/0.35	5/0.33	11/1.00	10/3.82
		HSS	9/0.10	10/0.27	10/0.44	10/0.71	11/3.77
	τ=5e-2	SPDHSS1	16/0.19	18/0.53	18/0.84	20/1.50	22/8.9
Relative error	r=50	SPDHSS1	1.3e-1	1.4e-1	1.4e-1	1.4e-1	1.6e-1
		SPDHSS2	6.0e-2	9.0e-2	9.5e-2	1.0e-1	1.3e-1
		HSS	2.0e-2	2.8e-2	3.3e-2	3.7e-2	4.7e-2
	τ=1e-2	SPDHSS1	1.7e-2	2.0e-2	2.0e-2	2.1e-2	2.4e-2
		SPDHSS2	1.7e-2	2.0e-2	4.0e-3	1.7e-2	7.0e-3
		HSS	1.4e-2	1.7e-2	1.5e-2	1.8e-2	2.1e-2
	τ=5e-2	SPDHSS1	8.7e-2	9.8e-2	1.0e-1	1.1e-1	1.2e-1
Construct. time (sec.) apprx/ulv	r=50	SPDHSS1	0.4/0.07	1.8/0.23	2.8/0.28	6.9/0.45	52.2/1.17
		SPDHSS2	1.1/0.08	4.6/0.17	7.6/0.28	23.9/0.32	125.5/0.82
		HSS	0.8/0.08	3.4/0.17	5.6/0.23	16.6/0.32	91.2/0.82
	τ=1e-2	SPDHSS1	1.7/0.82	8.1/3.19	15.8/6.14	33.0/11.41	410.2/127.95
		SPDHSS2	1.4/0.10	5.4/0.23	12.8/0.37	22.1/0.62	164.1/1.68
		HSS	0.7/7e-2	2.7/0.16	6.1/0.23	10.6/0.32	67.5/0.91
	τ=5e-2	SPDHSS1	0.4/0.10	1.4/0.20	2.6/0.27	5.16/0.41	37.3/1.10
Storage (MB)		Dense Matrix	78	322	619	1211	8915
		$r = 50$	11	23	30	46	128
	τ=1e-2	SPDHSS1	157	552	941	1588	8237
		SPDHSS2	15	35	58	81	275
		HSS	10	21	32	46	143
	τ=5e-2	SPDHSS1	16	32	43	59	151

In contrast to the previous problems, the standard HSS approximations using our chosen rank r and thresholds τ are found to be positive definite for these integral equation problems. Here, standard HSS should be the preconditioner of choice but there is no guarantee that the standard HSS approximations for these problems are always positive definite. Thus, it is still of interest to see how SPDHSS1 and SPDHSS2 perform.

Similar to what was observed in Table 6.2, the preconditioning performance of SPDHSS2 is close to that of standard HSS, even though standard HSS gives more accurate approximations. A disadvantage of SPDHSS1 here is also evident: for the small relative error threshold τ=1e-2, SPDHSS1 needs much larger ranks than standard HSS to compress the off-diagonal blocks as reflected by the enormous storage cost. This is corroborated by Figure 6.3 shown earlier, where the decay of relative errors in Method 1 is much slower than that of the truncated SVD. This disadvantage suggests that SPDHSS1 should only be used for low-accuracy preconditioning operations by controlling its off-diagonal block rank r.

To summarize, based on the three tests above, SPDHSS2 is more effective than SPDHSS1 for preconditioning but requires greater construction time and storage. SPDHSS1 is faster to construct but cannot provide highly accurate approximations with low-rank compression. A careful rank selection for HSS off-diagonal blocks in both methods is needed to balance the trade-off between preconditioner quality and construction cost. Like the standard HSS construction, both methods have $O(rn^2)$ computational complexity. We remark that our implementations are sequential and runtimes can be improved by taking advantage of parallelism. In addition, if the original matrix can be represented by an $\mathcal{H}^2$ matrix with the strong admissibility condition that is accurate enough to be SPD, it is then possible to further reduce the construction cost of both methods by taking advantage of the low-rank structures in these forms.

6.8 Conclusion

In this chapter, we designed two positive-definite-preserving HSS approximation algorithms based on a recursive description of constructing HSS representations. Method 1 is different from all existing methods in that the U_i used to compress the off-diagonal blocks are based on the diagonal blocks and do not require factoring HSS block rows. That this cheaper alternative can provide good approximations in some cases, as well as its generalization to different choices of invariant subspaces for U_i, are worthy of further study.

Method 2 chooses V_i to compress off-diagonal blocks after scaling. It is also worthwhile to explore how to choose V_i to directly minimize the approximation error $\|A^{(k)} - A^{(k-1)}\|$ (before scaling) at each stage, while also preserving positive definiteness.

This chapter also provided a method of understanding the errors incurred at each stage of an HSS approximation when projection is used to compress off-diagonal blocks. This use of projection led to the elegant result that the errors at each stage are orthogonal to each other. Experimentally, we observed that smaller approximation error is not always correlated with better preconditioned convergence rate. Better control of the preconditioned convergence behavior via the approximations chosen in rank-structured representations is a long-term goal of this research.

Appendices

APPENDIX A
RECURSIVE CONSTRUCTION OF INTERMEDIATE MATRICES

This appendix describes the recursive construction of intermediate matrices $B_{i,j}$ in Section 2.6 in order to construct an $\mathcal{H}^2$ matrix representation of a kernel matrix $K(X, X)$. For each block $K(X_i, X_j)$ with $j \in \mathcal{F}_i$ or $i \in \mathcal{F}_j$, the optimal intermediate matrix $B_{i,j}$ in the Frobenius norm can be represented as

$$B_{i,j} = \begin{cases} U_i^\dagger K(X_i, X_j)(V_j^T)^\dagger & j \in \mathcal{F}_i^1 \text{ or } i \in \mathcal{F}_j^1 \\ U_i^\dagger K(X_i, X_j) & j \in \mathcal{F}_i^2 \\ K(X_i, X_j)(V_j^T)^\dagger & i \in \mathcal{F}_j^2 \end{cases}. \tag{A.1}$$

All these optimal $B_{i,j}$ with $j \in \mathcal{F}_i$ or $i \in \mathcal{F}_j$ can be recursively constructed using B_{i_a,j_b} associated with possible children i_a of i and j_b of j. In the end, only $B_{i,j}$ with $(i, j) \in \mathcal{A}$ and $(i, j) \in \mathcal{A}_p$ are kept for the final $\mathcal{H}^2$ matrix representation. This recursive construction is described as follows.

When i and j are both leaf nodes, $B_{i,j}$ can be directly constructed according to eq. (A.1). Otherwise, we first consider the case where both i and j are non-leaf nodes with children $\{i_a\}$ and $\{j_b\}$. In this case, it can be shown that i and j are at the same level, $j \in \mathcal{F}_i^1$, and $i \in \mathcal{F}_j^1$. Each child j_b of j is also in $\mathcal{F}_{i_a}^1$ with all the children i_a of i. Assume that each child intermediate matrix B_{i_a,j_b} has been computed as

$$B_{i_a,j_b} = U_{i_a}^\dagger K(X_{i_a}, X_{j_b})(V_{j_b}^T)^\dagger.$$

Based on the nested forms of U_i and V_j, $B_{i,j}$ can thus be computed as

$$B_{i,j} = U_i^\dagger K(X_i, X_j)(V_j^T)^\dagger$$

$$= R_i^\dagger \begin{pmatrix} U_{i_1}^\dagger & & & \\ & U_{i_2}^\dagger & & \\ & & \ddots & \\ & & & U_{i_s}^\dagger \end{pmatrix} K(X_i, X_j) \begin{pmatrix} (V_{j_1}^T)^\dagger & & & \\ & (V_{j_2}^T)^\dagger & & \\ & & \ddots & \\ & & & (V_{j_s}^T)^\dagger \end{pmatrix} (S_j^T)^\dagger$$

$$= R_i^\dagger \begin{pmatrix} B_{i_1,j_1} & B_{i_1,j_2} & \dots & B_{i_1,j_s} \\ B_{i_2,j_1} & B_{i_2,j_2} & \dots & B_{i_2,j_s} \\ \vdots & \vdots & & \vdots \\ B_{i_s,j_1} & B_{i_s,j_2} & \dots & B_{i_s,j_s} \end{pmatrix} (S_j^T)^\dagger = R_i^\dagger \bar{B}_{i,j} (S_j^T)^\dagger. \tag{A.2}$$

where $\bar{B}_{i,j}$ denotes the matrix made up by all the children intermediate matrices B_{i_a,j_b}. This block $\bar{B}_{i,j}$ is of dimension $2^d r_0 \times 2^d r_0$ and thus computing $R_i^\dagger \bar{B}_{i,j}(S_j^T)^\dagger$ has $O(1)$ cost.

We then consider the case when i is a non-leaf node and j is a leaf node. In this case, $j \in \mathcal{F}_i$ is either at the same level as i or at an upper level. Similar to the above discussion, denote $\bar{B}_{i,j}$ as the vertical concatenation of intermediate matrices $B_{i_a,j}$ with all children i_a of i. Matrix $B_{i,j}$ can then be efficiently computed as

$$B_{i,j} = \begin{cases} R_i^\dagger \bar{B}_{i,j} (V_j^T)^\dagger & j \in \mathcal{F}_i^1 \\ R_i^\dagger \bar{B}_{i,j} & j \in \mathcal{F}_i^2 \end{cases}. \tag{A.3}$$

Similarly, when i is a leaf node and j is a non-leaf node, $i \in \mathcal{F}_j$ is either at the same level as j or at an upper level. Denote $\bar{B}_{i,j}$ as the horizontal concatenation of intermediate matrices B_{i,j_b} with all children j_b of j. Matrix $B_{i,j}$ can then be efficiently computed as

$$B_{i,j} = \begin{cases} U_i^\dagger \bar{B}_{i,j} (S_j^T)^\dagger & i \in \mathcal{F}_j^1 \\ \bar{B}_{i,j} (S_j^T)^\dagger & i \in \mathcal{F}_j^2 \end{cases}. \tag{A.4}$$

The overall construction of an $\mathcal{H}^2$ matrix representation of $K(X,X)$ combines Algorithm 1 and the above recursive construction of $B_{i,j}$ and is shown in Algorithm 13. The recursive construction of $B_{i,j}$ and Algorithm 1 can be merged together to traverse the partition tree $\mathcal{T}$ once. Moreover, using the computed $B_{i,j}$ at a lower level, the approximated matrices in Algorithm 1 for non-leaf nodes can be further replaced by smaller matrices, which helps reduce the low-rank approximation cost. We skip detailed discussion about this cost-reduction technique and interested readers can refer to Ref. [27].

Algorithm 13 Construction of an $\mathcal{H}^2$ matrix representation of $K(X, X)$s

Input: X, $K(x, y)$

Output: $\mathcal{H}^2$ matrix components: $\mathcal{T}$, U_i, V_i, R_i, S_i, and $B_{i,j}$

```
1:  • construct a hierarchical partitioning of $X$ which gives a partition tree $\mathcal{T}$.
2:  • apply Algorithm 1 to construct $U_i$, $V_i$, $R_i$, and $S_i$ for each node $i \in \mathcal{T}$.
3:  for $l = L, L-1, \ldots, 1$ do
4:      for all nodes $i$ and $j$ in level$^+(l)$ do
5:          if $i$ and $j$ are both leaf nodes then
6:              • compute $B_{i,j}$ using eq. (A.1).
7:          else if $i$ and $j$ are both non-leaf nodes then
8:              • compute $B_{i,j}$ using eq. (A.2).
9:          else if $i$ is a non-leaf node and $j$ is a leaf node then
10:             • compute $B_{i,j}$ using eq. (A.3).
11:         else if $i$ is a leaf node and $j$ is a non-leaf node then
12:             • compute $B_{i,j}$ using eq. (A.4).
13:         end if
14:     end for
15: end for
```

REFERENCES

[1] W. Hackbusch, "A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices," *Computing*, vol. 62, no. 2, pp. 89–108, Apr. 1999.

[2] W. Hackbusch and B. N. Khoromskij, "A sparse $\mathcal{H}$-matrix arithmetic. Part II: Application to multi-dimensional problems," *Computing*, vol. 64, no. 1, pp. 21–47, Jan. 2000.

[3] W. Hackbusch and S. Börm, "Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices," *Computing*, vol. 69, no. 1, pp. 1–35, Sep. 2002.

[4] W. Hackbusch, B. Khoromskij, and S. A. Sauter, "On $\mathcal{H}^2$-matrices," *Lectures on Applied Mathematics*, pp. 9–29, 2000.

[5] S. Chandrasekaran, M. Gu, and T. Pals, "A fast ULV decomposition solver for hierarchically semiseparable representations," *SIAM Journal on Matrix Analysis and Applications*, vol. 28, no. 3, pp. 603–622, Jan. 2006.

[6] S. Ambikasaran and E. Darve, "An $O(N \log N)$ fast direct solver for partial hierarchically semi-separable matrices," *Journal of Scientific Computing*, vol. 57, no. 3, pp. 477–501, Dec. 2013.

[7] M. Bebendorf, C. Kuske, and R. Venn, "Wideband nested cross approximation for Helmholtz problems," *Numerische Mathematik*, vol. 130, no. 1, pp. 1–34, 2015.

[8] S. Börm, "Directional $\mathcal{H}^2$-matrix compression for high-frequency problems," *Numerical Linear Algebra with Applications*, vol. 24, no. 6, e2112, Dec. 2017.

[9] Y. Li, H. Yang, E. R. Martin, K. L. Ho, and L. Ying, "Butterfly factorization," *Multiscale Modeling & Simulation*, vol. 13, no. 2, pp. 714–732, 2015.

[10] P. G. Martinsson and V. Rokhlin, "A fast direct solver for boundary integral equations in two dimensions," *Journal of Computational Physics*, vol. 205, no. 1, pp. 1–23, May 2005.

[11] M. Gu and S. Eisenstat, "Efficient algorithms for computing a strong rank-revealing QR factorization," *SIAM Journal on Scientific Computing*, vol. 17, no. 4, pp. 848–869, Jul. 1996.

[12] H. Cheng, Z. Gimbutas, P. G. Martinsson, and V. Rokhlin, "On the compression of low rank matrices," *SIAM Journal on Scientific Computing*, vol. 26, no. 4, pp. 1389–1404, 2005.

[13] A. Gillman and A. Barnett, "A fast direct solver for quasi-periodic scattering problems," *Journal of Computational Physics*, vol. 248, pp. 309–322, 2013.

[14] K. L. Ho and L. Greengard, "A fast direct solver for structured linear systems by recursive skeletonization," *SIAM Journal on Scientific Computing*, vol. 34, no. 5, A2507–A2532, Jan. 2012.

[15] V. Minden, A. Damle, K. L. Ho, and L. Ying, "Fast spatial Gaussian process maximum likelihood estimation via skeletonization factorizations," *Multiscale Modeling & Simulation*, vol. 15, no. 4, pp. 1584–1611, 2017.

[16] E. Corona, P. G. Martinsson, and D. Zorin, "An $O(N)$ direct solver for integral equations on the plane," *Applied and Computational Harmonic Analysis*, vol. 38, no. 2, pp. 284–317, 2015.

[17] L Greengard and V Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, Dec. 1987.

[18] L. Greengard and V. Rokhlin, "A new version of the fast multipole method for the Laplace equation in three dimensions," *Acta Numerica*, vol. 6, pp. 229–269, Jan. 1997.

[19] B. Engquist and L. Ying, "Fast directional multilevel algorithms for oscillatory kernels," *SIAM Journal on Scientific Computing*, vol. 29, no. 4, pp. 1710–1737, 2007.

[20] W. Fong and E. Darve, "The black-box fast multipole method," *Journal of Computational Physics*, vol. 228, no. 23, pp. 8712–8725, 2009.

[21] L. Ying, G. Biros, and D. Zorin, "A kernel-independent adaptive fast multipole algorithm in two and three dimensions," *Journal of Computational Physics*, vol. 196, no. 2, pp. 591–626, 2004.

[22] W. Hackbusch and Z. P. Nowak, "On the fast matrix multiplication in the boundary element method by panel clustering," *Numerische Mathematik*, vol. 54, no. 4, pp. 463–491, 1989.

[23] E. Michielssen and A. Boag, "A multilevel matrix decomposition algorithm for analyzing scattering from large structures," *IEEE Transactions on Antennas and Propagation*, vol. 44, no. 8, pp. 1086–1093, 1996.

[24] Z. Gimbutas and V. Rokhlin, "A generalized fast multipole method for nonoscillatory kernels," *SIAM Journal on Scientific Computing*, vol. 24, no. 3, pp. 796–817, 2003.

[25] L. Ying, "A kernel independent fast multipole algorithm for radial basis functions," *Journal of Computational Physics*, vol. 213, no. 2, pp. 451–457, 2006.

[26] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, "Fast algorithms for hierarchically semiseparable matrices," *Numerical Linear Algebra with Applications*, vol. 17, no. 6, pp. 953–976, Dec. 2010.

[27] W. Hackbusch, *Hierarchical Matrices: Algorithms and Analysis*. Springer, 2015, vol. 49.

[28] S. A. Goreinov, E. E. Tyrtyshnikov, and N. L. Zamarashkin, "A theory of pseudoskeleton approximations," *Linear Algebra and its Applications*, vol. 261, no. 1-3, pp. 1–21, 1997.

[29] M. Bebendorf and S. Rjasanow, "Adaptive low-rank approximation of collocation matrices," *Computing*, vol. 70, no. 1, pp. 1–24, Feb. 2003.

[30] L. Cambier and E. Darve, "Fast low-rank kernel matrix factorization through skeletonized interpolation," *SIAM Journal on Scientific Computing*, to appear.

[31] B. Engquist and L. Ying, "Fast directional algorithms for the Helmholtz kernel," *Journal of Computational and Applied Mathematics*, vol. 234, no. 6, pp. 1851–1859, 2010.

[32] G. Mastroianni and G. Milovanovic, *Interpolation processes: Basic theory and applications*. Springer Science & Business Media, 2008.

[33] L Bos, M Taylor, and B Wingate, "Tensor product Gauss-Lobatto points are Fekete points for the cube," *Mathematics of Computation*, vol. 70, no. 236, pp. 1543–1547, 2001.

[34] L. Bos, S. De Marchi, A. Sommariva, and M. Vianello, "Computing multivariate Fekete and Leja points by numerical linear algebra," *SIAM Journal on Numerical Analysis*, vol. 48, no. 5, pp. 1984–1999, 2010.

[35] N Kishore Kumar and J. Schneider, "Literature survey on low rank approximation of matrices," *Linear and Multilinear Algebra*, vol. 65, no. 11, pp. 2212–2244, 2017.

[36] M. Challacombe and E. Schwegler, "Linear scaling computation of the Fock matrix," *The Journal of Chemical Physics*, vol. 106, no. 13, pp. 5526–5536, 1997.

[37] M. Challacombe, E. Schwegler, and J. Almlöf, "Fast assembly of the Coulomb matrix: A quantum chemical tree code," *The Journal of Chemical Physics*, vol. 104, no. 12, pp. 4685–4698, Mar. 1996.

[38] J. M. Pérez-Jordá and W. Yang, "Fast evaluation of the Coulomb energy for electron densities," *The Journal of Chemical Physics*, vol. 107, no. 4, pp. 1218–1226, 1997.

[39] M. C. Strain, G. E. Scuseria, and M. J. Frisch, "Achieving linear scaling for the electronic quantum Coulomb problem," *Science*, vol. 271, no. 5245, pp. 51–53, 1996.

[40] C. A. White, B. G. Johnson, P. M. W. Gill, and M. Head-Gordon, "The continuous fast multipole method," *Chemical Physics Letters*, vol. 230, no. 1, pp. 8–16, Nov. 1994.

[41] ——, "Linear scaling density functional calculations via the continuous fast multipole method," *Chemical Physics Letters*, vol. 253, no. 3, pp. 268–278, May 1996.

[42] B. I. Dunlap, J. W. D. Connolly, and J. R. Sabin, "On some approximations in applications of $X\alpha$ theory," *The Journal of Chemical Physics*, vol. 71, no. 8, pp. 3396–3402, 1979.

[43] F. Weigend, "A fully direct RI-HF algorithm: Implementation, optimised auxiliary basis sets, demonstration of accuracy and efficiency," *Physical Chemistry Chemical Physics*, vol. 4, no. 18, pp. 4285–4291, 2002.

[44] J. L. Whitten, "Coulombic potential energy integrals and approximations," *The Journal of Chemical Physics*, vol. 58, no. 10, pp. 4496–4501, 1973.

[45] N. H. F. Beebe and J. Linderberg, "Simplifications in the generation and transformation of two-electron integrals in molecular calculations," *International Journal of Quantum Chemistry*, vol. 12, no. 4, pp. 683–705, 1977.

[46] R. A. Friesner, "Solution of self-consistent field electronic structure equations by a pseudospectral method," *Chemical Physics Letters*, vol. 116, no. 1, pp. 39–43, Apr. 1985.

[47] H. Koch, A. Sánchez de Merás, and T. B. Pedersen, "Reduced scaling in electronic structure calculations using Cholesky decompositions," *The Journal of Chemical Physics*, vol. 118, no. 21, pp. 9481–9484, May 2003.

[48] V. Khoromskaia, B. N. Khoromskij, and R. Schneider, "Tensor-structured factorized calculation of two-electron integrals in a general basis," *SIAM Journal on Scientific Computing*, vol. 35, no. 2, A987–A1010, 2013.

[49] B. Peng and K. Kowalski, "Highly efficient and scalable compound decomposition of two-electron integral tensor and its application in coupled cluster calculations," *Journal of Chemical Theory and Computation*, vol. 13, no. 9, pp. 4179–4192, 2017.

[50] E. G. Hohenstein, R. M. Parrish, and T. J. Martínez, "Tensor hypercontraction density fitting. I. Quartic scaling second- and third-order Møller-Plesset perturbation theory," *The Journal of Chemical Physics*, vol. 137, no. 4, p. 044 103, Jul. 2012.

[51] C. A. Lewis, J. A. Calvin, and E. F. Valeev, "Clustered low-rank tensor format: Introduction and application to fast construction of Hartree–Fock exchange," *Journal of Chemical Theory and Computation*, vol. 12, no. 12, pp. 5868–5880, Dec. 2016.

[52] J. Lu and L. Ying, "Compression of the electron repulsion integral tensor in tensor hypercontraction format with cubic scaling cost," *Journal of Computational Physics*, vol. 302, pp. 329–335, 2015.

[53] P. Martinsson, "A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix," *SIAM Journal on Matrix Analysis and Applications*, vol. 32, no. 4, pp. 1251–1274, 2011.

[54] X. Xing and E. Chow, "An efficient method for block low-rank approximations for kernel matrix systems," *arXiv preprint arXiv:1811.04134*, 2018.

[55] ——, "Error analysis of an accelerated interpolative decomposition for 3D Laplace problems," *arXiv:1811.00131*, 2018.

[56] T. Helgaker, P. Jørgensen, and J. Olsen, *Molecular Electronic-Structure Theory*. John Wiley & Sons, 2014.

[57] N. Halko, P. Martinsson, and J. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, Jan. 2011.

[58] E. Chow, X. Liu, S. Misra, M. Dukhan, M. Smelyanskiy, J. R. Hammond, Y. Du, X. Liao, and P. Dubey, "Scaling up Hartree-Fock calculations on Tianhe-2," *The International Journal of High Performance Computing Applications*, vol. 30, no. 1, pp. 85–102, 2016.

[59] M. Häser and R. Ahlrichs, "Improvements on the direct SCF method," *Journal of Computational Chemistry*, vol. 10, no. 1, pp. 104–111, 1989.

[60] D. L. Strout and G. E. Scuseria, "A quantitative study of the scaling properties of the Hartree-Fock method," *The Journal of Chemical Physics*, vol. 102, no. 21, pp. 8448–8452, 1995.

[61] B. P. Pritchard and E. Chow, "Horizontal vectorization of electron repulsion integrals," *Journal of Computational Chemistry*, vol. 37, no. 28, pp. 2537–2546, 2016.

[62] M. Sierka, A. Hogekamp, and R. Ahlrichs, "Fast evaluation of the Coulomb potential for electron densities using multipole accelerated resolution of identity approximation," *The Journal of Chemical Physics*, vol. 118, no. 20, pp. 9136–9148, 2003.

[63] J. C. Burant, M. C. Strain, G. E. Scuseria, and M. J. Frisch, "Analytic energy gradients for the Gaussian very fast multipole method (GvFMM)," *Chemical Physics Letters*, vol. 248, no. 1-2, pp. 43–49, 1996.

[64] Y. Shao, C. A. White, and M. Head-Gordon, "Efficient evaluation of the Coulomb force in density-functional theory calculations," *The Journal of Chemical Physics*, vol. 114, no. 15, pp. 6572–6577, 2001.

[65] F. Weigend and M. Häser, "RI-MP2: First derivatives and global consistency," *Theoretical Chemistry Accounts*, vol. 97, no. 1-4, pp. 331–340, 1997.

[66] R. Łazarski, A. M. Burow, L. Grajciar, and M. Sierka, "Density functional theory for molecular and periodic systems using density fitting and continuous fast multipole method: Analytical gradients," *Journal of Computational Chemistry*, vol. 37, no. 28, pp. 2518–2526, 2016.

[67] R. Łazarski, A. M. Burow, and M. Sierka, "Density functional theory for molecular and periodic systems using density fitting and continuous fast multipole methods," *Journal of Chemical Theory and Computation*, vol. 11, no. 7, pp. 3029–3041, 2015.

[68] M. Bebendorf, "Approximation of boundary element matrices," *Numerische Mathematik*, vol. 86, no. 4, pp. 565–589, Oct. 2000.

[69] D. Malhotra and G. Biros, "PVFMM: A parallel kernel independent fmm for particle and volume potentials," *Communications in Computational Physics*, vol. 18, no. 3, pp. 808–830, 2015.

[70] S. Wang, X. Li, J. Xia, Y. Situ, and M. de Hoop, "Efficient scalable algorithms for solving dense linear systems with hierarchically semiseparable structures," *SIAM Journal on Scientific Computing*, vol. 35, no. 6, pp. C519–C544, Jan. 2013.

[71] J. Xia and M. Gu, "Robust approximate Cholesky factorization of rank-structured symmetric positive definite matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 5, pp. 2899–2920, Jan. 2010.

[72] S. Li, M. Gu, C. Wu, and J. Xia, "New efficient and robust HSS Cholesky factorization of SPD matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 33, no. 3, pp. 886–904, Jan. 2012.

[73] M. Bebendorf and W. Hackbusch, “Stabilized rounded addition of hierarchical matrices,” *Numerical Linear Algebra with Applications*, vol. 14, no. 5, pp. 407–423, 2007.

[74] J. Xia, “A robust inner–outer hierarchically semi-separable preconditioner,” *Numerical Linear Algebra with Applications*, vol. 19, no. 6, pp. 992–1016, 2012.

[75] J. Z. Xia, “Effective and robust preconditioning of general spd matrices via structured incomplete factorization,” *SIAM Journal on Matrix Analysis and Applications*, vol. 38, no. 4, pp. 1298–1322, 2017.

[76] J. Xia, “On the complexity of some hierarchical structured matrix algorithms,” *SIAM Journal on Matrix Analysis and Applications*, vol. 33, no. 2, pp. 388–410, Jan. 2012.

[77] S. Börm, *Efficient Numerical Methods for Non-local Operators: $\mathcal{H}^2$-matrix Compression, Algorithms and Analysis*. European Mathematical Society, 2013.

[78] E. Chow and Y. Saad, “Preconditioned Krylov subspace methods for sampling multivariate gaussian distributions,” *SIAM Journal on Scientific Computing*, vol. 36, no. 2, A588–A608, Jan. 2014.

[79] E. Chow and J. Skolnick, “Effects of confinement on models of intracellular macromolecular dynamics,” *Proceedings of the National Academy of Sciences*, vol. 112, no. 48, pp. 14 846–14 851, 2015.

[80] ——, “DNA internal motion likely accelerates protein target search in a packed nucleoid,” *Biophysical Journal*, vol. 112, no. 11, pp. 2261–2270, 2017.

[81] W. Śmigaj, T. Betcke, S. Arridge, J. Phillips, and M. Schweiger, “Solving boundary integral problems with BEM++,” *ACM Trans. Math. Softw.*, vol. 41, no. 2, 6:1–6:40, 2015.