FAST ALGORITHMS FOR GENERATING MINIMAL RANK H2-MATRIX FOR ELECTRICALLY LARGE SURFACE INTEGRAL OPERATORS

by

Chang Yang

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

School of Electrical and Computer Engineering

West Lafayette, Indiana

December 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Dan Jiao, Chair

School of Electrical and Computer Engineering

Dr. Weng Cho Chew

School of Electrical and Computer Engineering

Dr. Steven D. Pekarek

School of Electrical and Computer Engineering

Dr. Alexander V. Kildishev

School of Electrical and Computer Engineering

Approved by:

Dr. Dimitrios Peroulis

To my family

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor Professor Dan Jiao, who give me the opportunity to work with her and study at this university. She led me into the world of Computational Electromagnetics and steered me in the right direction through the ocean. She will always be there whenever I feel the struggle in my research. Then she would work out every detail to give me a piece of tangible advice. I cannot imagine this work without her help. Her passion, rigor, wisdom, and diligence will surely continue inspiring me no matter where I go. Working with her will be one of my best memories.

I would like to thank other members of my Ph.D. advisory committee: Professor Weng Cho Chew, Professor Steven D. Pekarek, and Professor Alexander V. Kildishev for their precious time, support, and suggestions regarding my research work.

Thanks are also extended to all the fellow labmates who worked with me in the On-Chip Electromagnetics Lab here at Purdue: Dr. Jin Yan, Dr. Kaiyuan Zeng, Dr. Miaomiao Ma, Dr. Ping Li, Dr. Yu Zhao, Dr. Li Xue, Michael R. Hayashi, Shuzhan Sun, Yifan Wang, Vinicius Cabral Do Nascimento for their constant support and the friendly yet professional environment in the lab.

Last but not the least, I would like to thank my family for their love and ultimate support during all these years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ACA	Adaptive Cross Approximation
CEM	Computational Electromagnetics
EFIE	Electric Field Integral Equation
FMM	Fast Multipole Method
IE	Integral Equation
MLFMA	Multi-level Fast Multipole Algorithm
MVM	Matrix Vector Multiplication
NPSA	Nested Pseudo Skeleton Approximation
NRA	Nested Reduction Algorithm
PEC	Perfect Electric Conductor
PSA	Pseudo Skeleton Approximation
RCS	Radar cross-section
rSVD	reduced SVD
RHS	Right Hand Side
SIE	Surface Integral Equation
VIE	Volumn Integral Equation

ABSTRACT

Computational electromagnetics (CEM) plays an important role in many aspects of today's engineering world. Among existing CEM methods, Integral Equation (IE) based solvers are popular because of their versatility, efficiency, and reliability. IE-based methods in general result in a dense system matrix. To solve this system matrix efficiently, a prevailing solution is to use a Fast Multipole Method (FMM) with an iterative solver. Recently, fast $\mathcal{H}^2$-matrix based direct solvers have been developed to directly invert or factorize the dense system matrix. However, how to generate an $\mathcal{H}^2$-representation efficiently for electrically large analysis remains an unsolved problem. Existing methods for generating an $\mathcal{H}^2$-matrix of electrically large integral operators are all expensive, especially for surface IE (SIE) operators.

In this work, we proposed and developed a series of fast algorithms to generate a rank-minimized $\mathcal{H}^2$-matrix for electrically large SIE-based analysis, the best of which has complexity as low as $O(N \log N)$ in time and memory. Using the $\mathcal{H}^2$-matrix generated in this work, we can make the $\mathcal{H}^2$-matrix-based direct solver achieve a total complexity of $O(N^{1.5})$ in time and $O(N \log N)$ in memory for electrically large SIE analysis. In contrast, generating an $\mathcal{H}^2$-matrix or inverting a dense matrix in a brute-force way both will cost $O(N^3)$ in time and $O(N^2)$ in memory. In addition to accelerating direct solvers, we significantly reduce the CPU time of a matrix-vector multiplication as well as the memory consumption because of the rank-minimized $\mathcal{H}^2$-representation. In addition to electromagnetic analysis, the proposed algorithms are applicable to many other disciplines, where a compact representation of dense matrices is the key to the reduction of computational complexity.

1. INTRODUCTION

In the very first chapter, we briefly summarise the challenges posed and the background of our research. We also briefly describe our contribution along with the outline of this work.

1.1 Challenges

Computational Electromagnetics (CEM) is a critical part of modern scientific research and industry design, ranging from Electronic Design Automation (EDA) to the design of stealth airplanes. Nowadays, the computational domain usually either contains very large objects or very fine and complex structures, or both. Solving such complex and large structures in a brutal-force way will be cost-prohibitive. Thus, it is important to develop efficient methods to obtain the solution. Among all the popular methods available now, Surface Integral Equation (SIE) based solvers gain particular interest due to their versatility, efficiency, and reliability. Compared to other mainstream CEM methods that discretize the whole 3-D volume of the computational domain, SIE only needs to discretize the surface of a target object when calculating homogeneous problems, and thus reduce the computational domain by one dimension. SIE solvers result in a dense matrix system $ZI = V$. The current fast method to solve this matrix system is to use Multilevel Fast Multipole Algorithm (MLFMA) together with an iterative solver. By using the MLFMA, the computation of one matrix-vector multiplication can be sped up to as low as $O(N \log N)$ in time [1]–[4]. Thus, the total time needed is $O(NN_{\mathrm{it}}N_{rhs})$, where N_{it} is the number of iterations required for convergence and $O(N_{rhs})$ is the number of right hand side (RHS) vectors.

Despite that great progress has been made in iterative solvers, the direct solver has always lacked although strongly desired. Compared to iterative solvers, a direct solver that directly inverts the impedance matrix Z or directly factorizes it in a decomposition form that can be inverted easily has multiple advantages. First, a direct solver avoids the N_{it} in the time complexity, which is typically very large when N increases. Second, a direct solver can be easily multiplied by multiple RHS, once we get the inverted Z matrix. On the other hand, for an iterative solver, we need to repeat the whole iterative process for each RHS. It is worth mentioning that currently many works have been done to overcome the aforementioned problems inherent in an iterative solver. However, it still is a huge advantage if we can develop a direct solver of optimal time complexity of optimal $O(N \log N)$. This is exactly where the challenges present, especially for electrically large SIE and VIE analysis.

$\mathcal{H}^2$-matrix is a general mathematical framework [5], [6] that has drawn much attention in recent years. It can be used to develop a fast direct solver for electrical large IE operators. For an existing $\mathcal{H}^2$-matrix, we can directly invert it in $O(N^{1.5})$ in time and $O(N)$ in memory for electrically large SIEs and $O(N \log N)$ for VIEs [7]. However, existing methods for generating an $\mathcal{H}^2$-matrix of electrically large integral operators are expensive. Thus, the fast generation of $\mathcal{H}^2$-matrix for large IE operators is in demand.

The dissertation here concerns the research problem of fast generating an $\mathcal{H}^2$-matrix. We propose to start from an MLFMA-based representation and convert it to a minimal rank $\mathcal{H}^2$-matrix. In this way, we can leverage the low complexity of FMM to solve electrically large problems. In addition, we propose purely algebraic methods to generate a rank-minimized $\mathcal{H}^2$ representation.

In the remainder of this chapter, I will go through the mathematical background of $\mathcal{H}^2$-Matrix and how to use the MLFMA to generate an initial $\mathcal{H}^2$-matrix, while introducing the notions I will use throughout the dissertation. Then I will briefly state the contributions of this work.

1.2 Mathematical Background

1.2.1 Electric Field Integral Equations (EFIEs)

The Method of Moments based discretization of the surface electric field integral equation results in a dense system matrix $\mathbf{Z}$, whose mn-th entry can be written as

$$\mathbf{Z}_{m,n} = \int_{S_m} \int_{S_n} (\vec{f_m} \cdot \vec{f_n} - \frac{1}{k^2} \nabla_s \cdot \vec{f_m} \nabla_s \cdot \vec{f_n}) G dS' dS, \tag{1.1}$$

where $G = \frac{\mathrm{e}^{-\mathrm{j}k_0 R}}{4\pi R}$ is the Green's function, k_0 is the wave number, $R = \|\mathbf{r} - \mathbf{r}'\|$ is the distance between a source point $\mathbf{r}'$ and an observer point $\mathbf{r}$, $\vec{f_m}$ and $\vec{f_n}$ are the vector bases on triangular patches S_m, and S_n respectively. Here, triangular elements are used to discretize a surface, and RWG bases are employed to expand unknown currents.

The $\mathbf{Z}$ can be expressed as the sum of the electric scalar and magnetic vector potential based components as follows

$$\mathbf{Z} = -\mathbf{Z}_{\phi} + \mathbf{Z}_{Ax} + \mathbf{Z}_{Ay} + \mathbf{Z}_{Az}, \tag{1.2}$$

where,

$$\mathbf{Z}_{\phi,mn} = \frac{1}{k_0^2} \int_{S_m} \int_{S_n} (\nabla_s \cdot \vec{f_m} \nabla_s \cdot \vec{f_n}) G dS' dS, \tag{1.3}$$

and

$$\mathbf{Z}_{A\xi,mn} = \int_{S_m} \int_{S_n} (f_{m\xi} f_{n\xi}) G dS' dS, \quad \xi = \{x, y, z\}. \tag{1.4}$$

1.2.2 $\mathcal{H}^2$-Matrix

An $\mathcal{H}^2$-matrix [6] represents the interaction between two binary trees. An example of the $\mathcal{H}^2$-matrix is illustrated in Fig. 1.1. We call the binary tree a cluster tree since each node in the tree represents a cluster of unknowns. All the source bases form a column tree, whereas all the observer bases form a row tree. When the testing function is chosen to be the same as the basis function, the resultant matrix is symmetrical, whose row and column trees are identical. In an $\mathcal{H}^2$-matrix, by checking the admissibility condition level by level between a row cluster tree and a column cluster tree, the original matrix is partitioned into multilevel admissible and inadmissible blocks. The admissible block is represented by a green submatrix and the inadmissible one is shown in red in Fig. 1.1. Physically, an admissible block represents the interaction between separated sources (column cluster) and observers (row cluster), which satisfies the following admissibility condition

$$d < \eta D \tag{1.5}$$

where d denotes the maximal diameter of the row and column cluster, D is the distance between two clusters, and η is a positive parameter, which can be used to adjust the matrix partition.

An inadmissible block is stored in its original full matrix format. An admissible block has a compact storage. Take an admissible block formed between a row cluster t and a column cluster s as an example. It is stored as $\mathbf{V}^t\mathbf{S}^{t,s}(\mathbf{V}^s)^H$, where $\mathbf{S}$ is called a coupling matrix, and $\mathbf{V}$ is called a cluster basis. $\mathbf{V}^t$ means the $\mathbf{V}$ matrix specified for cluster t. Similar notation applies for $\mathbf{S}^{t,s}$ and $\mathbf{T}^t$ that will be introduced below. $\mathbf{V}$ is nested in the sense that for a cluster t whose children clusters are t_1 and t_2, their cluster bases have the following relationship

$$\mathbf{V}^t = \begin{bmatrix} \mathbf{V}^{t_1} & \\ & \mathbf{V}^{t_2} \end{bmatrix} \begin{bmatrix} \mathbf{T}^{t_1} \\ \mathbf{T}^{t_2} \end{bmatrix}, \tag{1.6}$$

in which $\mathbf{T}$ matrix is called a transfer matrix. For a non-leaf cluster t, if i is t's child, then we may refer cluster i as t^{i} and i's transfer matrix as $\mathbf{T}^{t\mathrm{i}}$.

A block that is formed by one row cluster and one column cluster sharing the same tree level is called a block cluster and denoted as b. It is easy to see block clusters also form a tree, named block cluster tree. A block cluster can either be admissible, inadmissible or having four children, which are normally denoted b^{i} for the ith child. The root block cluster is the $\mathcal{H}^2$-Matrix itself. It has four children for the $\mathcal{H}^2$ matrix shown in Fig. 1.1, $b^0 = \{7, 7\}$, $b^1 = \{7, 14\}$, $b^2 = \{14, 7\}$, $b^3 = \{14, 14\}$. Each of these four children also has four children. This goes on until one block cluster is either an admissible block or an inadmissible block.

Define t_p as the set of all ancestors of t including t itself. For example, in Fig. 1.1, the set of ancestors of cluster 1 is $\{1, 3, 7, 15\}$. Define $\hat{t}^+ = \{s : \{t_p, s\}\text{is an admissible block}\}$. Let $\boldsymbol{G}^t$ be denoted as the matrix formed by t and $\hat{t}^+$. For example, $\boldsymbol{G}^1$ is the matrix concatenated by three submatrices, which are matrix formed by $\{1, 4\}$, $\{1, 8\}$ and $\{1, 13\}$.

In the algorithms presented in this dissertation, we assume one non-leaf cluster has two children for the sake of simplicity. But the case of multiple children can always be readily generalized.

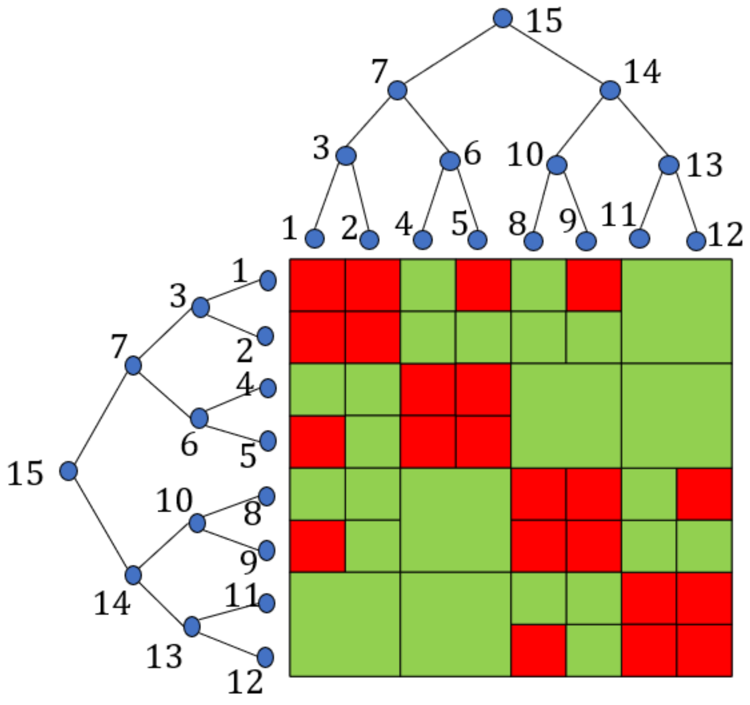


Figure 1.1. Illustration of an $\mathcal{H}^2$-matrix

1.2.3 Method for Generating an Initial $\mathcal{H}^2$-matrix from MLFMA

According to the addition theorem, Green's function G can be written as

$$\frac{\mathrm{e}^{-\mathrm{j}kR}}{4\pi R} \approx \sum_p \omega_p \mathrm{e}^{-\mathrm{j}k\vec{S_p}\cdot\vec{d}} T_{L,X}(\vec{S_p}), \tag{1.7}$$

where p refers to the index of the sampling point defined on a unit spherical surface, $\vec{S_p}$ and ω_p are the position vector, and the quadrature weight of each sampling point, respectively; L is the truncation parameter used in the addition theorem; $\vec{d} = \vec{R} - \vec{X} = \mathbf{r} - \mathbf{r}' - \vec{X}$, $\vec{X} = \vec{O_1} - \vec{O_2}$, with $\vec{O_1}$ being the center of an observer group whose points are denoted by $\mathbf{r}$, and $\vec{O_2}$ the center of a source group whose points are $\mathbf{r}'$, and

$$T_{L,\vec{X}}(\vec{S_p}) = \frac{-\mathrm{j}k_0}{4\pi} \sum_{l=0}^{L} \frac{(-\mathrm{j})^l(2l+1)}{4\pi} h_l^{(1)*}(kX) P_l(\vec{S_p}\cdot\hat{X}), \tag{1.8}$$

where $h_l^{(1)}$ stands for a spherical Hankel function of the first kind, superscript $*$ denotes a complex conjugate, P_l are Legendre polynomials, and $\hat{X}$ stands for a unit vector along $\vec{X}$. Substituting (1.7) into (1.3), we obtain

$$\mathbf{Z}_{\phi,\mathrm{ij}} = \mathbf{V}_{\mathrm{i},p} S_{p,p} \mathbf{W}_{\mathrm{j},p}{}^{H}, \tag{1.9}$$

in which

$$\mathbf{V}_{\mathrm{i},p} = \sum_q \frac{w_{\mathrm{i},q}}{k_0} \mathrm{e}^{-\mathrm{j}k_0 \vec{S_p}\cdot(\vec{r_\mathrm{i}}-\vec{O_1})} \nabla_s \cdot \vec{f_\mathrm{i}}(\vec{r}_{\mathrm{i},q}) \tag{1.10}$$

$$\mathbf{S}_{p,p} = diag(\omega_p T_{l,\vec{O_1}-\vec{O_2}}(\vec{S_p})) \tag{1.11}$$

$$\mathbf{W}_{\mathrm{j},p} = \sum_q \frac{w_{\mathrm{j},q}}{k_0} \mathrm{e}^{-\mathrm{j}k_0 \vec{S_p}\cdot(\vec{r_\mathrm{j}}-\vec{O_2})} \nabla_s \cdot \vec{f_\mathrm{j}}(\vec{r}_{\mathrm{j},q}), \tag{1.12}$$

where w_q are weighting coefficients used for a numerical surface integration on source, and field triangular patch respectively.

Consider a cluster i, whose parent cluster is i$'$, the $\mathbf{V}_{\mathrm{i}'}$ is related to $\mathbf{V}_\mathrm{i}$ by $\mathbf{V}_{\mathrm{i}'} = \mathbf{V}_\mathrm{i}\mathbf{T}$, where $\mathbf{T}$ is called a transfer matrix shown as the following

$$\mathbf{T}_{pp'} = \sum_{m,l\le K} Y_{ml}(\theta'_{p'}, \phi'_{p'}) Y^*_{ml}(\theta_p, \phi_p)\omega_p \mathrm{e}^{-\mathrm{j}k\vec{S'_p}(\vec{O}_1-\vec{O}'_1)},$$

where p' is the index of the sampling points for $\mathbf{V}_{t'}$, K here stands for the number of quadrature points in θ direction of $\mathbf{V}_t$, and Y_{ml} are spherical harmonics. It is clear that the number of the sampling points is the rank of cluster basis $\mathbf{V}$. In θ direction, the sampling points are chosen as Gauss-Legendre points; while in ϕ direction, uniform sampling is used. $\mathbf{T}$ is sparse for a prescribed accuracy, whose number of entries per column is a constant regardless of matrix size. This number can be further reduced using a filter [2]. When the size of $\mathbf{T}$ is large, we can also use Lagrange Polynomial to generate $\mathbf{T}$, which will result in a sparse matrix [1]. This is the key to keep the complexity of FMM low.

There are many ways to determine the truncation number L in (1.8). For example, we can set

$$L = k_0 d + 1.8 d_0^{2/3} (k_0 d)^{\frac{1}{3}} \tag{1.13}$$

where $d_0 = \log_{10}(\frac{1}{\epsilon_F})$ and ϵ_F is the desired accuracy for FMM, k_0 is wavenumber, and d is the diameter of the targeted cluster [1]. Another good criterion for the determination of truncation number L is as the following:

$$L = k_0 d + C \ln(k_0 d + \boldsymbol{\pi}) \tag{1.14}$$

where C is a constant, and $C = 2.25$ is suggested in [2].

This induces the second admissible condition in the construction of FMM generated $\mathcal{H}^2$-matrix, which is shown in the following

$$L(k_0, d) < kD \tag{1.15}$$

where L is the truncation number and is determined by k_0 and d. Otherwise, (1.8) will diverge [3].

1.3 Contributions of This Work

In this work, we propose two classes of algorithms to fast construct rank-minimized $\mathcal{H}^2$-matrices of an electrically large IE operator. Its rank is minimized based on accuracy. In the first class, we propose to start from an MLFMA-based representation whose rank is full, and then develop fast algorithms to convert it to a rank-minimized $\mathcal{H}^2$-matrix. In this way, the low complexity of MLFMA in handling electrically large problems can be utilized to build an efficient $\mathcal{H}^2$-representation for electrically large analysis. In the second class, we propose a method to generate a rank-minimized $\mathcal{H}^2$ algebraically without utilizing kernel-specific information. Using both classes of methods, the real rank of $\mathcal{H}^2$-matrix for EFIE is reduced to the minimal one required by accuracy, which grows as $O(N^{0.5})$ with N the unknown size in an SIE. As a result, using the proposed rank-minimized $\mathcal{H}^2$-matrix can accelerate both iterative and direct solvers. Meanwhile, the complexity of generating a rank-minimized $\mathcal{H}^2$-matrix can be as low as $O(N^{1.5})$ for electrically large SIE operators and $O(N \log N)$ for electrically large VIE operators.

1.4 Dissertation Outline

The remainder of this dissertation is organized as follows.

In Chap. 2, we introduce a method to convert an FMM-based representation to a new rank-minimized $\mathcal{H}^2$-matrix. This method is based on an existing method that can convert an $\mathcal{H}^2$-matrix whose rank is not minimized for accuracy to a new rank-minimized $\mathcal{H}^2$-matrix [8]. However, the conversion method in [8] only has a low computational cost when the original $\mathcal{H}^2$-matrix has a constant rank. This is not the case for the FMM-based representation, which has a full rank. Thus directly applying the method of [8] will result in a very high complexity of $O(N^3)$. Hence, we improved the method to greatly reduce the conversion complexity to $O(N^2)$ for electrically large SIEs.

In Chap. 3, we develop a method that greatly simplifies the method in Chapter 2 while costing much less absolute CPU run time, but with the same accuracy preserved. In this method, we directly obtain new cluster bases from old cluster bases and then use the updated cluster bases to update coupling matrices. As a comparison, in the method described in Chapter 2, we consider the whole matrix information to update cluster bases and coupling matrices. The method developed in this chapter serves as a foundation for the work in the following chapters that further reduce the complexity.

In Chap. 4, we develop a new algorithm to further reduce the complexity of the conversion process. The complexity can be reduced to $O(N^{1.5} \log N)$ in time for electrically large SIE operators and $O(N \log N)$ for electrically large VIE operators. These methods utilize randomness to reduce complexity. But we can show both theoretically and numerically that accuracy is not compromised.

In Chap. 5, we propose an algorithm that is purely algebraic but can generate a rank-minimized $\mathcal{H}^2$-matrix in a similar complexity as in the previous chapter, which is $O(N^{1.5} \log N)$ in time and $O(N \log N)$ in memory. Without using MLFMA, the converting process can be greatly simplified regarding implementation. This new method relies on the Pseudo-Skeleton Approximation, which shows great reliability when used in the SIE.

In Chap. 6, we propose another algebraic algorithm, Nested Pseudo Skeleton Approximation. It cost $O(k^3)$ for each cluster and coupling matrix in time and $O(k^2)$ in memory. This will result in a total complexity of $O(N^{1.5})$ in time and $O(N \log N)$ in memory, which is more advantageous than the previous one, while preserves the advantages of being algebraic.

In Chap. 7, Although PSA is robust most of the time, it still has the potential to fail when we deliberately choose the unfavorable pivots. So, in this chapter, we develop a new algorithm that has the same accuracy as PSA but is unable to fail with IE operators. We call it Analytical Skeleton Approximation because we analytically find a set of real or auxiliary pivots that make PSA robust and accurate.

In Chap. 8, we summarize the work that has been done and present future work.

2. CONVERTING AN $\mathcal{H}^2$-MATRIX TO A MINIMAL RANK $\mathcal{H}^2$-MATRIX

The FMM method and the multi-level fast multipole algorithm (MLFMA) [1] have been widely used to solve electrically large integral operators. The matrix resulting from an FMM-based representation of integral operators (IE) is an $\mathcal{H}^2$-matrix. This $\mathcal{H}^2$-matrix is special in the sense that it has a diagonal coupling matrix, and a sparse transfer matrix. As a result, although the asymptotic rank of an FMM representation is full for electrically large analysis, the complexity of one matrix-vector multiplication for surface IE can still be achieved as $O(N \log N)$, where N is matrix size. The asymptotic behavior of the FMM's rank has led to a belief that the rank of an electrically large IE operator is full, i.e., it is not of low rank. In this work, we present fast algorithms to convert the FMM-based $\mathcal{H}^2$-matrix to a rank-minimized $\mathcal{H}^2$-matrix based on prescribed accuracy.

There is an existing method that can convert an $\mathcal{H}^2$-matrix whose rank is not minimized for accuracy to a new rank-minimized $\mathcal{H}^2$-matrix [8]. If the original $\mathcal{H}^2$-matrix is of constant rank, then the converting process can be done in $O(N)$. But the $\mathcal{H}^2$-matrix derived from MLFMA is full rank, meaning that the rank grows quadratically with the electrical size and thus linear with the number of unknowns in one cluster. Thus, directly applying the converting method in [8] will be costly. The procedure described in this chapter is an improvement of [8] to accommodate the scenario of a full rank $\mathcal{H}^2$-matrix. Some other changes are also done because the $\mathcal{H}^2$-matrix generated from FMM is in the form of $\mathbf{VSV}^H$ instead of $\mathbf{VSV}^T$.

To speed up the converting process, for a low-rank matrix $\mathbf{B}$ associated with a cluster basis or a block cluster basis, we choose to store its decomposed form, like $\mathbf{B} = \mathbf{B}_a\mathbf{B}_b^T$. Here subscript a and b indicate the left and right matrix. Then all the operations carried on that matrix $\mathbf{B}$ in the original algorithm is carried out on the factorized form of matrix $\mathbf{B}_a\mathbf{B}_b^T$. For the matrix $\mathbf{B}$ in higher level, instead of directly factorizing $\mathbf{B}$, we use the nested property of $\mathcal{H}^2$-matrix, meaning using children level's $\mathbf{B}_a$ and $\mathbf{B}_b^T$ to compute parent level's $\mathbf{B}_a$ and $\mathbf{B}_b^T$. Let $\tilde{\mathbf{V}}^t$, $\tilde{\mathbf{T}}$, $\tilde{\mathbf{S}}^{t,s}$ denote cluster basis, transfer matrix, coupling matrix for the converted low rank new $\mathcal{H}^2$-matrix.

We apply these algorithms to a surface-based electric-field integral equation (EFIE) for solving scattering problems. Our numerical experiments show that after the conversion, the original full rank of the FMM representation can be significantly reduced to a much smaller rank. The resulting minimal-rank $\mathcal{H}^2$-matrix can be used for both fast iterative and direct solutions of the integral equations. As an example, we directly factorize and solve the minimal-rank $\mathcal{H}^2$-matrix obtained from an initial FMM representation of the EFIE using [7]. The resultant RCS of a sphere shows excellent agreement with analytical Mie series solutions.

2.1 Proposed Algorithm

2.1.1 Fast Computation Using the Factorized Form of AB^T

First I will introduce the procedure of an efficient addition of two AB^T form matrices while applying rSVD, which can be found in [6]. This procedure includes an efficient rSVD of an AB^T form matrix which can also be found in [6]. The algorithms are shown in Algorithm 1 and 2. These two algorithms are essential for the development of the new algorithm in this chapter.

Algorithm 1 Adding two lowrank matrices

```
1: procedure Addition_low_ranks($\mathbf{M}_1 = \mathbf{A}\mathbf{B}^T, \mathbf{M}_2 = \mathbf{C}\mathbf{D}^T$)
2:     $\mathbf{M} = \mathbf{M}_1 + \mathbf{M}_2 = \begin{bmatrix}\mathbf{A} & \mathbf{C}\end{bmatrix}\begin{bmatrix}\mathbf{B} & \mathbf{D}\end{bmatrix}^T$
3: end procedure
```

Algorithm 2 reduced SVD on a low rank matrix of AB^T form

```
1: procedure rSVD_low_rank($\mathbf{M} = \mathbf{A}\mathbf{B}^T \in \mathcal{R}^{m\times n}$)
2:     Compute (reduced) QR-factorisations of $\mathbf{A}$,$\mathbf{B}$: $\mathbf{A} = \mathbf{Q}_A\mathbf{R}_A$, $\mathbf{B} = \mathbf{Q}_B\mathbf{R}_B$
3:     Compute a rSVD of $\mathbf{R}_A\mathbf{R}_B^T = \mathbf{U}'\mathbf{\Sigma}\mathbf{V}'^T$
4:     Compute $\mathbf{U} := \mathbf{Q}_A\mathbf{U}'$, $\mathbf{V} := \mathbf{Q}_B\mathbf{V}'$
5: end procedure
```

In algorithm 2, assuming that the input matrix $\mathbf{M}$ is stored in the form of $\mathbf{M} = \mathbf{A}\mathbf{B}^T$ and is of size $m \times n$. And the size of $\mathbf{A}$ and $\mathbf{B}$ is $m \times k$ and $n \times k$, respectively. Note here k is not the minimal numerical rank for $\mathbf{M}$ and we are going to find the real minimal numerical rank in the algorithm of this chapter. In line 2, we compute reduced QR factorization of $\mathbf{A}$, $\mathbf{B}$ such that $\mathbf{A} = \mathbf{Q}_A\mathbf{R}_A$, $\mathbf{B} = \mathbf{Q}_B\mathbf{R}_B$, respectively. This procedure only costs $O((m+n)k^2)$. Then in line 3, we first compute the multiplication of $\mathbf{R}_A\mathbf{R}_B^T$. Then we do reduced SVD based on a prescribed criterion ϵ, on the resultant matrix to get $\mathbf{R}_A\mathbf{R}_B^T = \mathbf{U}'\mathbf{\Sigma}\mathbf{V}'^T$. This step of rSVD can reveal the true rank $\tilde{k}$ for input matrix $\mathbf{M}$. This procedure only costs $O(k^3)$ since $\mathbf{R}_A$ and $\mathbf{R}_B$ are both of size $k \times k$. Finally, we do two multiplications $\mathbf{U} := \mathbf{Q}_A\mathbf{U}'$ and $\mathbf{V} := \mathbf{Q}_B\mathbf{V}'$. This final procedure costs $O((m+n)k\tilde{k})$. After these three steps, the input $\mathbf{M}$ can be expressed as $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. Its true rank is revealed based on accuracy criterion ϵ compared to the original form of $\mathbf{A}\mathbf{B}^T$. Compared to brute-force decomposition of $\mathbf{M}$, which will cost $O(min(m^2n, mn^2))$, this procedure clearly has a complexity advantage if k is small compared to m and n, which is true in the following algorithm. We can multiply $\mathbf{\Sigma}$ to either $\mathbf{U}$ or $\mathbf{V}$ if $\mathbf{M} = \mathbf{A}\mathbf{B}^T$ form is more desired.

2.1.2 Proposed Algorithm

In [8], a linear-complexity algorithm is developed for converting a constant-rank $\mathcal{H}^2$-matrix whose rank is not minimized for accuracy to a new rank-minimized $\mathcal{H}^2$-matrix. Here, since the $\mathcal{H}^2$-matrix obtained from the FMM has a rank increasing with tree level from the leaf level up to the root level, as the electrical size increases, we have to accelerate the algorithm in [8] to make the conversion efficient. In addition, the cluster bases of the FMM-based $\mathcal{H}^2$ are not orthogonal. They are also complex-valued. In this chapter, we address these difficulties and successfully develop fast algorithms to compress the FMM-based representation which is full rank to a minimal-rank $\mathcal{H}^2$ based on prescribed accuracy. The detailed procedure is as follows.

We start from an initial $\mathcal{H}^2$-matrix generated from the FMM. Beginning from leaf level, for every cluster t, we assemble all the admissible blocks formed by t, and t's parents at all non-leaf levels, into a matrix called $\mathbf{G}^t$. We then obtain $\mathbf{G}_2^t = \mathbf{G}^t\mathbf{G}^{tH}$. An SVD based on prescribed accuracy performed on $\mathbf{G}_2^t$ provides a new cluster basis of leaf cluster t, $\tilde{\mathbf{V}}^t$.

A naive brute-force computation of $\mathbf{G}_2^t$ is computationally expensive. Here, we use the nested property of the FMM-based $\mathcal{H}^2$-matrix to compute $\mathbf{G}_2^t$ efficiently. The expression of $\mathbf{G}_2^t$ can be found from Eqn. (19) in [8]. However, here, since $\mathbf{V}$ is complex-valued, different from (20) in [8], we have $\mathbf{B}^s = \mathbf{V}^{sH}\mathbf{V}^s$ for cluster s. $\mathbf{V}^s$ is of size $\#s \times k_{F,l}$, where $\#s$ denotes the cardinality of cluster s, and $k_{F,l}$ denotes an FMM-rank at tree level l. Since $k_{F,l}$ is large, we cannot afford to computing $\mathbf{B}^s$ as it is. Instead, we first obtain $\mathbf{B}^s$ for each leaf cluster, then we apply a reduced SVD (rSVD) [6] based on prescribed accuracy ϵ to $\mathbf{B}^s$, obtaining $\mathbf{B}^s = \mathbf{B}_a^s\mathbf{B}_b^{sT}$, where the size of $\mathbf{B}_a^s$ and $\mathbf{B}_b^s$ is both $k_{F,l} \times k_{m,l}$, and $k_{m,l}$ is the minimal rank at level l. Then at a non-leaf level, using nested property, $\mathbf{B}^s$ is obtained from $\mathbf{T}_1^H\mathbf{B}_a^{s1}\mathbf{B}_b^{s1^T}\mathbf{T}_1 + \mathbf{E}_2^H\mathbf{B}_a^{s2}\mathbf{B}_b^{s2^T}\mathbf{T}_2$. Since $\mathbf{T}$ is sparse, the computation of $\mathbf{T}_1^H\mathbf{B}_a^{s1}$ costs only $O(k_{F,l} \times k_{m,l})$ instead of $O(k_{F,l}^2 \times k_{m,l})$. We then add the two low rank matrices, and perform another rSVD based on accuracy ϵ to obtain $\mathbf{B}^s = \mathbf{B}_a^s\mathbf{B}_b^{sT}$, the overall cost of which is only $O(k_{F,l} \times k_{m,l}^2)$ as compared to a brute-force $O(k_{F,l}^3)$ cost. The aforementioned procedure is performed level by level from leaf level up to the minimum level where there are admissible blocks. The pseudo-code of this step is shown in Algorithm 3. We feed this algorithm with root cluster basis.

Algorithm 3 Computing_$\mathbf{B}^s$

```
1:  procedure Computing_$\mathbf{B}^s(s)$
2:      if t is leaf cluster then
3:          $\mathbf{B}^s \leftarrow \mathbf{V}^{sH}\mathbf{V}^s$
4:          Do SVD on $\mathbf{B}^s$, store it as $\mathbf{B}_a^s$ and $\mathbf{B}_b^s$ so that $\mathbf{B}^s = \mathbf{B}_a^s\mathbf{B}_b^{sT}$
5:      else
6:          for i = 0;i <$children$;i++ do
7:              Computing_$\mathbf{B}^s(s_\mathrm{i})$
8:              $a_{temp} \leftarrow \mathbf{T}^{\mathrm{si}^H}\mathbf{B}_a^{\mathrm{si}}$
9:              $b_{temp}^T \leftarrow \mathbf{B}_b^{\mathrm{si}^T}\mathbf{T}^{\mathrm{si}}$
10:             $A \leftarrow$ Addition_low_ranks($\mathbf{B}^s, a_{temp}b_{temp}^T$)
11:             $(\mathbf{B}_a^s, \mathbf{B}_b^s) \leftarrow$ rSVD_low_rank(A)
12:         end for
13:     end if
14: end procedure
```

After computing $\mathbf{B}$ for each cluster, next we compute $\bar{\mathbf{S}}$ for each cluster. This matrix is shown in (20) in [8], which sums up all the coupling matrices of the admissible blocks formed by a cluster. Since $\mathbf{B}$ has been obtained in its minimal-rank factorized form, we can compute $\bar{\mathbf{S}}^t$ efficiently also in its minimal-rank form, which is $\bar{\mathbf{S}}^t = \bar{\mathbf{S}}_a \bar{\mathbf{S}}_b^T$. Obtaining $\bar{\mathbf{S}}_{a(b)}$ for each admissible block costs only $O(k_{F,l} \times k_{m,l})$ since $\bar{\mathbf{S}}_{a(b)} = \mathbf{S}^{t,s}\mathbf{B}^s_{a(b)}$ and $\mathbf{S}^{t,s}$ is diagonal. We then perform the addition of C_{sp} low-rank matrices formed by t, and apply an rSVD to the resultant to obtain $\bar{\mathbf{S}}^t$ in its minimal-rank form, the cost of which is also $O(k_{F,l} \times k^2_{m,l})$. The pseudo-code of this step is shown in Algorithm 4. In this pseudo-code, b denote the corresponding block cluster basis. And we need to feed the root block cluster basis to this algorithm. The root block cluster basis is just the original $\mathcal{H}^2$-matrix.

Algorithm 4 Computing_$\bar{\mathbf{S}}^t$

```
1:  procedure Computing_$\bar{\mathbf{S}}^t(b)$
2:      if $b$ is non-leaf block cluster then
3:          for i = 0; i <children; i++ do
4:              Computing_$\bar{\mathbf{S}}(b_{\mathrm{i}})$
5:          end for
6:      else if $b$ is admissible block then
7:          $a_{temp} \leftarrow \mathbf{S}^{t,s}\mathbf{B}^s_a$
8:          $b^T_{temp} \leftarrow \mathbf{B}^{sT}_b \mathbf{S}^{t,s^H}$
9:          $A \leftarrow$ Addition_low_ranks$(\bar{\mathbf{S}}^t, a_{temp} b^T_{temp})$
10:         $(\bar{\mathbf{S}}^t_a, \bar{\mathbf{S}}^t_b) \leftarrow$ rSVD_low_rank(A)
11:     end if
12: end procedure
```

After $\bar{\mathbf{S}}$ is obtained for each cluster, we compute $\mathbf{S}_{sum}$ by a top-down tree traversal, since the transfer matrices are sparse, the cost of obtaining the minimal-rank form of $\mathbf{S}^t_{sum}$ for each cluster t also costs $O(k_{F,l} \times k^2_{m,l})$. The pseudo-code of this step is shown in Algorithm 5. Note here we still use $\bar{\mathbf{S}}^t$ in the pseudo-code. However, after running this procedure, $\bar{\mathbf{S}}^t$ becomes $\mathbf{S}_{sum}$. We feed this algorithm with the root cluster basis.

After $\mathbf{S}^t_{sum}$ is computed, we perform an SVD on $\mathbf{G}^t_2 = \mathbf{V}^t \mathbf{S}^t_{sum} \mathbf{V}^{tH}$ for leaf cluster t, the cost of which is constant. The resultant singular vectors truncated based on accuracy ϵ are the new cluster bases for leaf cluster t. At a non-leaf level, we first project the $\mathbf{G}^t_2$ of a non-leaf cluster t onto the new cluster bases of its children, as shown in (24) of [8]. We obtain $\mathbf{G}^t_{2,proj}$, the size of which is $k_{m,l} \times k_{m,l}$, on which an SVD is performed to find the new transfer matrix of t. Again, computing $\mathbf{G}^t_{2,proj}$ costs $O(k_{F,l} \times k^2_{m,l})$ only, and the subsequent

Algorithm 5 Updating_$\bar{\mathbf{S}}^t$

```
1: procedure Updating_$\bar{\mathbf{S}}^t(t)$
2:     if $t$ is non-leaf cluster then
3:         for i = 0; i <children; i++ do
4:             $a_{temp} \leftarrow T^{t\mathrm{i}}\bar{\mathbf{S}}^t_a$
5:             $b^T_{temp} \leftarrow (\bar{\mathbf{S}}^t_b)^T(T^{t\mathrm{i}})^H$
6:             $A \leftarrow$ Addition_low_ranks($\bar{\mathbf{S}}^{t\mathrm{i}}, a_{temp}b^T_{temp}$)
7:             $(\bar{\mathbf{S}}^{t\mathrm{i}}_a, \bar{\mathbf{S}}^{t\mathrm{i}}_b) \leftarrow$ rSVD_low_rank(A)
8:             Updating_$\bar{\mathbf{S}}^t(t_\mathrm{i})$
9:         end for
10:    end if
11: end procedure
```

SVD costs $O(k_{m,l}^3)$. The pseudo-code of this part is shown at Algorithm 6. Here line 13 and line 21 are added if we want final $\mathcal{H}^2$-matrix possess the form of $\mathbf{Z} = \mathbf{VSW}^T$. If we just want final $\mathcal{H}^2$-matrix possess the form of $\mathbf{Z} = \mathbf{VSW}^H$, then we do not need these two lines. And we also do not need to compute or store $\tilde{\mathbf{B}}^s$.

Algorithm 6 Mini_cluster_rank

```
1: procedure Mini_cluster_rank($(t, \epsilon)$)
2:     if $t$ is non-leaf cluster then
3:         for i = 0; i <children; i++ do
4:             Mini_cluster_rank($t_\mathrm{i}, \epsilon$)
5:             $\boldsymbol{B}_T^{t\mathrm{i}} \leftarrow \tilde{\mathbf{B}}^{ti}\mathbf{T}^{ti}$
6:         end for
7:         $a_{temp} \leftarrow \begin{bmatrix} \boldsymbol{B}_T^{t1} \\ \boldsymbol{B}_T^{t2} \end{bmatrix} \mathbf{S}_{sum,a}^t$
8:         $b_{temp}^T \leftarrow (\mathbf{S}_{sum,b}^t)^T \begin{bmatrix} \boldsymbol{B}_T^{t1} \\ \boldsymbol{B}_T^{t2} \end{bmatrix}^H$
9:         $\mathbf{G}_{2,proj}^t \leftarrow a_{temp} b_{temp}^T$
10:        do SVD on $\mathbf{G}_{2,proj}^t = pdp^H$
11:        $\tilde{\mathbf{T}}^{t1} \leftarrow p^{t1}$, $\tilde{\mathbf{T}}^{t2} \leftarrow p^{t2}$
12:        $\tilde{\mathbf{B}}^t \leftarrow \tilde{\mathbf{T}}^{t1}\boldsymbol{B}_T^{t1} + \tilde{\mathbf{T}}^{t2}\boldsymbol{B}_T^{t2}$
13:        $\tilde{\mathbf{B}}^s \leftarrow (\tilde{\mathbf{T}}^{t1})^H \tilde{\mathbf{B}}^{s1} conj(\mathbf{T}^{t1}) + (\tilde{\mathbf{T}}^{t2})^H \tilde{\mathbf{B}}^{s2} conj(\mathbf{T}^{t2})$
14:    else
15:        $a_{temp} = \mathbf{V}^t \mathbf{S}_{sum,a}^t$
16:        $b_{temp}^T = (\mathbf{S}_{sum,b}^t)^T \mathbf{V}^{tH}$
17:        $\mathbf{G}_2 = a_{temp} b_{temp}^T$
18:        do SVD on $\mathbf{G}_2$, $\mathbf{G}_2 = pdp^H$
19:        $\tilde{\mathbf{V}}^t \leftarrow p^t$
20:        $\tilde{\mathbf{B}}^t \leftarrow (\tilde{\mathbf{V}}^t)^H \mathbf{V}^t$
21:        $\tilde{\mathbf{B}}^s \leftarrow (\tilde{\mathbf{V}}^t)^H conj(\mathbf{V}^t)$
22:    end if
23: end procedure
```

Finally, we can update the coupling matrix, as shown in 7. Remember here $\mathbf{S}^{t,s}$ is original FMM $\mathcal{H}^2$-matrix's coupling matrix.

Carrying out the computation described in Algorithms 3, 4, 5, 6 and 7 in sequence will convert the original full rank FMM $\mathcal{H}^2$-matrix to a new minimal rank $\mathcal{H}^2$-matrix. The accuracy is obviously well-controlled since only QR and SVD involve approximations whose accuracy is also controlled. The complexity is also low as we will analyze below.

Algorithm 7 Mini_block_rank

```
1: procedure Mini_block_rank((b))
2:     if $b$ is non-leaf block cluster then
3:         for i = 0; i <children; i++ do
4:             Mini_block_rank($b_{\mathrm{i}}$)
5:         end for
6:     else if $b$ is admissible block then
7:         $\tilde{\mathbf{S}}^{t,s}_{new} \leftarrow \tilde{\mathbf{B}}^t \mathbf{S}^{t,s} (\tilde{\mathbf{B}}^s)^T$
8:     end if
9: end procedure
```

2.2 Accuracy and Complexity Analysis

Doing reduced SVD on a matrix of $\mathbf{M} = \mathbf{A}\mathbf{B}^T$ form costs $O((m+n)k^2)$ operations, where k is the rank of $\mathbf{M}$. m and n is the row dimension and column dimension of $\mathbf{M}$. Thus, $\mathbf{A}$ is of is size (m, k), and $\mathbf{B}$ is of size (n, k). Having this knowledge, we first analyze the complexity of algorithm 3. Since we want to store $\mathbf{B}$ matrix in its low rank form ($\mathbf{B} = \mathbf{B}_a\mathbf{B}_b^T$). Thus, for a non-leaf cluster t, we need to first know $\mathbf{B}_a$ and $\mathbf{B}_b$. Here we have $\mathbf{B}$ matrix of its son cluster t_{i}. We first do $\mathbf{B}_a^t = \mathbf{T}^H\mathbf{B}_a^{t_{\mathrm{i}}}$. Since $\mathbf{T}$ is a sparse matrix with constant number of entries per line, the complexity of this procedure is $O(kn)$. Similarly, computing $\mathbf{B}_b^t = \mathbf{B}_b\mathbf{T}$ also takes $O(kn)$ operations. Cluster t normally has two children t_1, t_2. Thus, we need to add two low-rank matrices. This procedure can be done using algorithm 1, followed with algorithm 2 to minimize the rank. This procedure can be done in $O((m+n)k^2)$ as well. Consider we need to do traverse through the tree structure. For each cluster, the cost is $O(nk^2)$, where n is the size of this cluster, and k is the rank of this cluster.

Next we analyze the complexity of algorithm 4. Similarly to last one, we first compute $\bar{\mathbf{S}}_a^t = \mathbf{S}_{t,s}\mathbf{B}_a^s$. Then compute $\bar{\mathbf{S}}_b^t = \mathbf{B}_b^s\mathbf{S}_{t,s}^H$. Since $\mathbf{S}^{t,s}$ is diagonal matrix. This process need $O(kn)$ operations. For a cluster t, $\bar{\mathbf{S}}^t$ needed to be added C_{sp} times for different cluster s. Each addition uses the process mentioned above and takes $O(nk^2)$ operations. Thus, the total number of operations for one cluster is $O(C_{sp}nk^2) = O(nk^2)$.

Next is the analysis of algorithm 5. The analysis of this process is almost the same as that of algorithm 3 but reversed. Thus for each cluster, the cost is $O(nk^2)$, where n is the size of this cluster, k is the rank of this cluster.

Finally, the analysis of algorithm 6. For a non-leaf level, we need to compute $\mathbf{G}_{2,proj} = \boldsymbol{B}_T \cdot \bar{\mathbf{S}} \cdot \boldsymbol{B}_T^H$. $\boldsymbol{B}_T$ involved here is of size $k \times n$, where k is minimal rank revealed by SVD of $\mathbf{G}_2$ matrix. Thus, similarly, we first compute $\boldsymbol{B}_T \cdot \bar{\mathbf{S}}_a^t$, then compute $\bar{\mathbf{S}}_b^t \cdot \boldsymbol{B}_T^H$. Finally multiply these two resulting matrices. Time and memory complexity is of $O(k^3)$.

To summarize, in each of the aforementioned algorithms, the complexity of each cluster is $O(nk^2)$. Thus, if for each cluster, rank k is a constant, the total complexity is $O(N \log N)$, where N is the total number of degrees of freedom. If for each cluster, rank k has order

$k \sim O(\log(n))$, the total complexity is $O(N \log^3 N)$. If the rank k grows as $O(n^{0.5})$, as in the SIE case [9], the total complexity is $O(N^2)$.

2.3 Numerical Result

The scattering of a conducting sphere at 300 MHz is considered. The setup is shown in Fig. 2.1. The PEC sphere is centered in the origin of the coordinate. The direction of incident plane wave is $\hat{k} = (0, 0, -1)$ with E field of $\hat{E} = (1, 0, 0)$. The scattering field is measured as bistatic RCS with $\phi = 0$ and θ ranging from 0 to 180. In this thesis, when other geometries are considered, the bRCS are all computed in this fashion with the geometry is centered at origin. The mesh density is chosen to be about $\frac{\lambda}{10}$, and the radius of the sphere is increased to examine the electrical size dependence of the proposed algorithm. We use MLFMA to generate an initial $\mathcal{H}^2$-matrix of an EFIE. The number of Gaussian quadrature points is three on each triangular patch. $Leafsize$ is set to be 40, and $\eta = 1.2$. For coupling matrices, Lagrange interpolation is used for larger sizes. For small transfer matrices, spherical harmonics are employed for global interpolation; for large ones that are sparse, six points are used for Lagrange interpolation along both θ and ϕ direction. We set $L = kd + 1.8 d_0^{\frac{2}{3}} (kd)^{\frac{1}{3}}$ as the truncation number used in the addition theorem, where k is wave number, d is the diameter of a cluster. We first validate the proposed algorithm using a sphere whose $N = 73,728$, with a radius of 4.42 m. After converting the FMM $\mathcal{H}^2$-matrix to a new minimal-rank $\mathcal{H}^2$-matrix, we apply the direct solver of [7] to directly solve it. The resultant bistatic RCS along θ direction is plotted in Fig. 2.2 in comparison with Mie series solution. Excellent agreement is observed. In this example, $d_0 = \log(100)$, ϵ for the rSVD used in the algorithm is 10^{-4}. The sparsity constant used in the transfer matrix is 10.

We then examine the performance of the proposed algorithm by increasing the electrical size of the sphere, yielding N from 4,608, 18,432, 73,728 to 294,912. For a quick test, $\epsilon = 10^{-2}$ is used for rSVD, and $d_0 = \log(10)$. In Fig. 2.4, we plot the rank of FMM and that of the minimal-rank $\mathcal{H}^2$-matrix as a function of N, clearly, the rank of the new $\mathcal{H}^2$-matrix is significantly reduced, and its growth rate is lower. If a higher accuracy setting is used, the growth rate of the new rank is expected to be even lower. In Fig. 2.3, we plot the memory

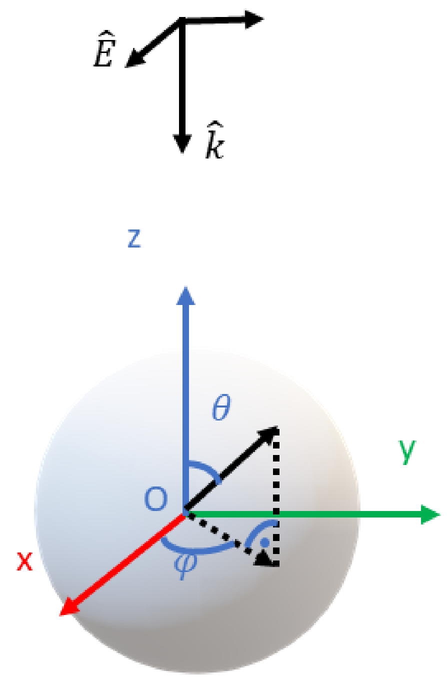


Figure 2.1. Setup for computing bistatic RCS

usage of the FMM and that of the new $\mathcal{H}^2$-matrix. A clear reduction in memory is observed. The accuracy is checked by evaluating the largest admissible block error of the new $\mathcal{H}^2$-matrix. It is found to be 0.227%, 0.229%, 0.325%, and 0.335% respectively. Corresponding figure is shown at Fig. 2.5. Hence, the proposed algorithm is able to significantly compress the original FMM-representation while maintaining good accuracy. These results are also shown in Table 2.1. In this table, N is the number of unknowns, FMM rank is the maximum rank in FMM $\mathcal{H}^2$-matrix, and New rank is the maximum rank in New $\mathcal{H}^2$-matrix. FMM time is the time used to generate FMM $\mathcal{H}^2$-matrix, while New time is the time used to generate New $\mathcal{H}^2$-matrix. FMM mem is the memory needed to generate FMM $\mathcal{H}^2$-matrix, while New mem is the memory needed to generate New $\mathcal{H}^2$-matrix. FMM error is the relative error of the largest admissible block of FMM $\mathcal{H}^2$-matrix compared with that of MoM matrix, and New error is the counterpart of New $\mathcal{H}^2$-matrix.

Table 2.1. Data of converting process for a suite of spheres

N	4608	18432	73728	294912
FMM rank	242	882	3872	12168
New rank	6	14	39	134
FMM time (s)	77.12	310.34	1247.32	4966.76
New time (s)	1.78e3	1.77e4	1.18e5	8.66e5
FMM mem (Mbs)	121.95	878.64	4.12e3	1.85e4
New mem (Mbs)	3.21	30.39	185.13	1239.47
FMM error	1.49e-6	5.18e-6	1.93e-5	3.07e-5
New error	2.27e-3	2.29e-3	3.25e-3	3.35e-3

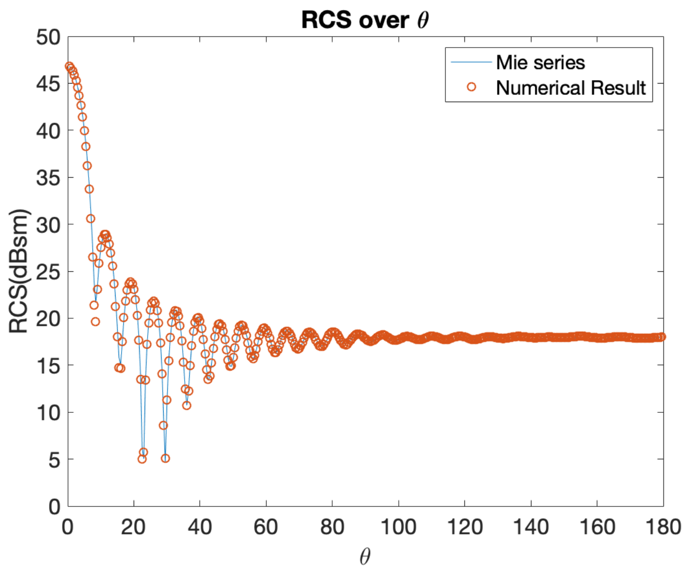


Figure 2.2. number of unknowns to be 73728, electrical size to be 4.42

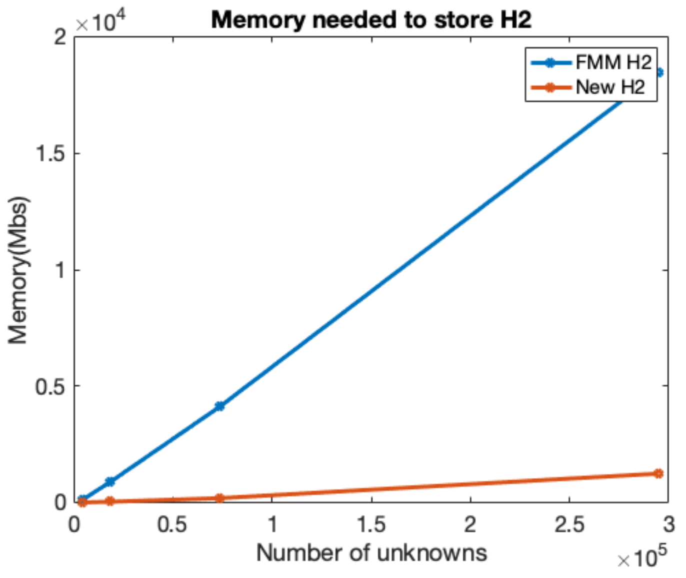


Figure 2.3. Compare of memory used to generate $\mathcal{H}^2$-matrices, PEC-sphere, $\epsilon_{rSVD} = 10^{-2}$

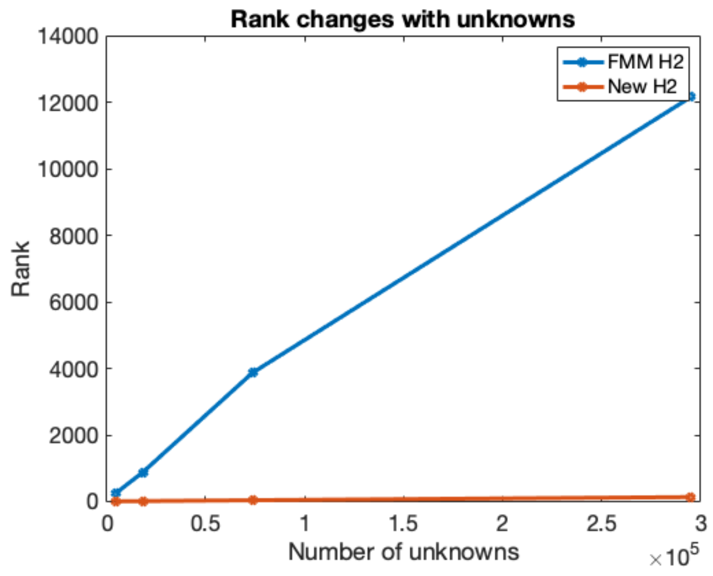


Figure 2.4. Compare of rank of two $\mathcal{H}^2$-matrices, PEC-sphere, $\epsilon_{rSVD} = 10^{-2}$

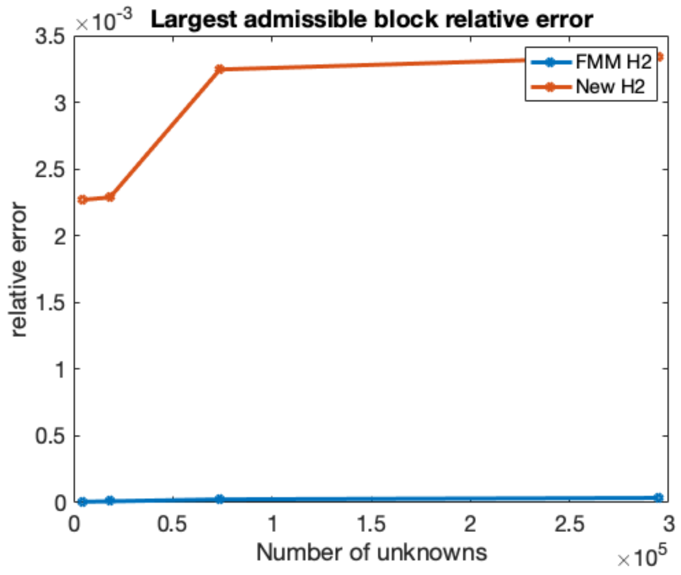


Figure 2.5. Compare of relative error of two $\mathcal{H}^2$-matrices, PEC-sphere, $\epsilon_{rSVD} = 10^{-2}$

3. NESTED REDUCTION ALGORITHM (NRA)

In the last chapter, we show how to leverage the MLFMA [2]–[4] to generate an $\mathcal{H}^2$-matrix. But the algorithm is lengthy and complicated. The CPU run time is also long. Thus, in this chapter, we aim to develop a method that is much simplified and takes much less CPU run time, without sacrificing accuracy. We use the resultant minimal rank $\mathcal{H}^2$-matrix together with a direct solver [7] to solve an electrically large surface IE problem and compute the RCS of it. The resultant RCS shows great agreement with reference results.

3.1 Simplifying the conversion algorithm

In chapter 2, we do the conversion based on the construction and decomposition of $\mathbf{G}_2$ matrix. However, later we find out that only manipulating cluster basis $\mathbf{V}$ and $\mathbf{T}$ can also do the conversion. Based on this observation, we propose the following algorithm to convert an FMM $\mathcal{H}^2$-matrix to a rank-minimized $\mathcal{H}^2$-matrix.

3.1.1 New Scheme to Convert FMM H2 to New H2

In this new scheme, for leaf cluster, we first compute $\mathbf{B}^t = (\mathbf{V}^t)^H\mathbf{V}^t$, where $\mathbf{V}^t$ is FMM $\mathcal{H}^2$ cluster basis. Then do rSVD on $\mathbf{B}^t$ to get $\mathbf{B}^t = \mathbf{P}^t\mathbf{D}^t(\mathbf{P}^t)^H$. Since $\mathbf{V}^t$ can represent $\mathbf{G}^t$, $\mathbf{P}^t$ can also represent $\mathbf{G}^t$. Remember $\mathbf{G}^t$ is the matrix concatenated by all the admissible submatrices formed by t. Thus, $\tilde{\mathbf{V}}^t = \mathbf{V}^t\mathbf{P}^t(\mathbf{D}^t)^{-\frac{1}{2}}$ can also accurately represent $\mathbf{G}^t$. This can be justified as follows

$$\begin{aligned}(\tilde{\mathbf{V}}^t)^H\tilde{\mathbf{V}}^t\mathbf{G}^t &= (\mathbf{D}^t)^{-1/2}(\mathbf{P}^t)^H(\mathbf{V}^t)^H\mathbf{V}^t\mathbf{P}^t(\mathbf{D}^t)^{1/2}\mathbf{G}^t \\ &= (\mathbf{D}^t)^{-1/2}(\mathbf{P}^t)^H\mathbf{P}^t\mathbf{D}^t(\mathbf{P}^t)^H\mathbf{P}^t(\mathbf{D}^t)^{1/2}\mathbf{G}^t \\ &= \mathbf{G}^t\end{aligned} \tag{3.1}$$

Here we use the fact that $\mathbf{D}^t$ is real diagonal. Then we compute $\tilde{\mathbf{B}} = (\tilde{\mathbf{V}}^t)^H\mathbf{V}^t$ and store it for further use. If we want final result to be of form $\mathbf{V}\mathbf{S}\mathbf{V}^T$ instead of $\mathbf{V}\mathbf{S}\mathbf{V}^H$, we need to compute one more matrix $\hat{\mathbf{B}} = (\tilde{\mathbf{V}}^t)^H\bar{\mathbf{V}}^t$ and store it for further use.

For non-leaf cluster, we use projected $\mathbf{V}^t$ matrix to $\hat{\mathbf{V}}^t$. In another word, we use $\hat{\mathbf{V}}^t$ to substitute $\mathbf{V}^t$ in non-leaf level, which can be expressed as

$$\begin{aligned}\hat{\mathbf{V}}^t &= \begin{bmatrix}\tilde{\mathbf{V}}^{t1}(\tilde{\mathbf{V}}^{t1})^H & 0 \\ 0 & \tilde{\mathbf{V}}^{t2}(\tilde{\mathbf{V}}^{t2})^H\end{bmatrix}\mathbf{V}^t \\ &= \begin{bmatrix}\tilde{\mathbf{V}}^{t1}(\tilde{\mathbf{V}}^{t1})^H\mathbf{V}^{t1}\mathbf{T}^{t1} \\ \tilde{\mathbf{V}}^{t2}(\tilde{\mathbf{V}}^{t2})^H\mathbf{V}^{t2}\mathbf{T}^{t2}\end{bmatrix}\end{aligned} \tag{3.2}$$

Thus, as in the leaf cluster case, we construct $\tilde{\mathbf{V}}^t$ as,

$$\begin{aligned}\tilde{\mathbf{V}}^t &= \hat{\mathbf{V}}^t\mathbf{P}^t(\mathbf{D}^t)^{-1/2} \\ &= \begin{bmatrix}\tilde{\mathbf{V}}^{t1}(\tilde{\mathbf{V}}^{t1})^H\mathbf{V}^{t1}\mathbf{T}^{t1}\mathbf{P}^t(\mathbf{D}^t)^{-1/2} \\ \tilde{\mathbf{V}}^{t2}(\tilde{\mathbf{V}}^{t2})^H\mathbf{V}^{t2}\mathbf{T}^{t2}\mathbf{P}^t(\mathbf{D}^t)^{-1/2}\end{bmatrix}\end{aligned} \tag{3.3}$$

Thus, we have $\tilde{\mathbf{T}}^{ti} = (\tilde{\mathbf{V}}^{ti})^H\mathbf{V}^{ti}\mathbf{T}^{ti}\mathbf{P}^t(\mathbf{D}^t)^{-\frac{1}{2}} = \tilde{\mathbf{B}}^{ti}\mathbf{T}^{ti}\mathbf{P}^t(\mathbf{D}^t)^{-\frac{1}{2}}$.

Finally, we can do $\tilde{\mathbf{S}}^{t,s} = \tilde{\mathbf{B}}^t\mathbf{S}^{t,s}(\tilde{\mathbf{B}}^s)^H$ to get new coupling matrix. We briefly summarise this algorithm in the following.

1. For leaf cluster,

   (a) Compute $\mathbf{B}^t = (\mathbf{V}^t)^H\mathbf{V}^t$. Then do rSVD on $\mathbf{B}$ such that $\mathbf{B} = \mathbf{P}^t\mathbf{D}^t(\mathbf{P}^t)^H$, while truncating $\mathbf{D}^t$ based on accuracy ϵ.

   $$\mathbf{B}^t = (\mathbf{V}^t)^H\mathbf{V}^t = \mathbf{P}^t\mathbf{D}^t(\mathbf{P}^t)^H \tag{3.4}$$

   Then we store $\mathbf{B}^t$ in decomposition form

   $$\mathbf{B}^t = \mathbf{B}_a^t(\mathbf{B}_a^t)^H \tag{3.5}$$

   by letting $\mathbf{B}_a^t = \mathbf{P}^t(\mathbf{D}^t)^{1/2}$.

(b) Then, update $\mathbf{V}^t$ matrix to $\tilde{\mathbf{V}}^t$ using follow equation

$$\tilde{\mathbf{V}}^t = \mathbf{V}^t\mathbf{P}^t(\mathbf{D}^t)^{-\frac{1}{2}} \tag{3.6}$$

(c) finally, compute $\tilde{\mathbf{B}}$ and store it for further use,

$$\tilde{\mathbf{B}} = (\tilde{\mathbf{V}}^t)^H\mathbf{V}^t \tag{3.7}$$

(d) Additionally, if want final result to be of form $\mathbf{VSV}^T$ instead of $\mathbf{VSV}^H$. One more computation is needed, which is

$$\hat{\mathbf{B}} = (\tilde{\mathbf{V}}^t)^H\bar{\mathbf{V}}^t \tag{3.8}$$

2. For non-leaf cluster,

(a) first compute $\mathbf{B}^t = (\mathbf{V}^t)^H\mathbf{V}^t$ following

$$\begin{aligned}
\mathbf{B}^t &= (\mathbf{V}^t)^H\mathbf{V}^t \\
&= \begin{bmatrix} \mathbf{V}^{t1}\mathbf{T}^{t1} \\ \mathbf{V}^{t2}\mathbf{T}^{t2} \end{bmatrix}^H \begin{bmatrix} \mathbf{V}^{t1}\mathbf{T}^{t1} \\ \mathbf{V}^{t2}\mathbf{T}^{t2} \end{bmatrix} \\
&= (\mathbf{T}^{t1})^H\mathbf{B}^{t1}\mathbf{T}^{t1} + (\mathbf{T}^{t2})^H\mathbf{B}^{t2}\mathbf{T}^{t2} \\
&= \sum_{\mathrm{i}}(\mathbf{T}^{t\mathrm{i}})^H\mathbf{B}_a^{t\mathrm{i}}(\mathbf{B}_a^{t\mathrm{i}})^H\mathbf{T}^{t\mathrm{i}} \\
&= \mathbf{P}^t\mathbf{D}^t(\mathbf{P}^t)^H
\end{aligned} \tag{3.9}$$

Note here Algorithm 1 and 2 are used in above computation to reduce the complexity. Alternatively, we can compute $\mathbf{B}^t = (\hat{\mathbf{V}}^t)^H\hat{\mathbf{V}}^t$ as

$$\begin{aligned}
\mathbf{B}^t &= (\hat{\mathbf{V}}^t)^H\hat{\mathbf{V}}^t \\
&= \sum_{\mathrm{i}}(\mathbf{T}^{t\mathrm{i}})^H(\tilde{\mathbf{G}}^{t\mathrm{i}})^H\tilde{\mathbf{G}}^{t\mathrm{i}}\mathbf{T}^{t\mathrm{i}} \\
&= \mathbf{P}^t\mathbf{D}^t(\mathbf{P}^t)^H
\end{aligned} \tag{3.10}$$

in which Algorithm 1 and 2 should also be used.

(b) Then compute transfer matrix as

$$\begin{aligned}\tilde{\mathbf{T}}^{ti} &= (\tilde{\mathbf{V}}^{ti})^H \mathbf{V}^{ti}\mathbf{T}^{ti}\mathbf{P}^t(\mathbf{D}^t)^{-\frac{1}{2}} \\ &= \tilde{\mathbf{B}}^{ti}\mathbf{T}^{ti}\mathbf{P}^t(\mathbf{D}^t)^{-\frac{1}{2}}\end{aligned} \tag{3.11}$$

(c) finally compute $\tilde{\mathbf{B}}$ for further use

$$\begin{aligned}\tilde{\mathbf{B}}^t &= (\tilde{\mathbf{V}}^t)^H \mathbf{V}^t \\ &= \begin{bmatrix}\tilde{\mathbf{V}}^{t1}\tilde{\mathbf{T}}^{t1} \\ \tilde{\mathbf{V}}^{t2}\tilde{\mathbf{T}}^{t2}\end{bmatrix}^H \begin{bmatrix}\mathbf{V}^{t1}\mathbf{T}^{t1} \\ \mathbf{V}^{t2}\mathbf{T}^{t2}\end{bmatrix} \\ &= (\tilde{\mathbf{T}}^{t1})^H\tilde{\mathbf{B}}^{t1}\mathbf{T}^{t1} + (\tilde{\mathbf{T}}^{t2})^H\tilde{\mathbf{B}}^{t2}\mathbf{T}^{t2}\end{aligned} \tag{3.12}$$

(d) Additionally, if want final result to be of form $\mathbf{VSV}^T$ instead of $\mathbf{VSV}^H$. One more computation is needed, which is

$$\begin{aligned}\hat{\mathbf{B}} &= (\tilde{\mathbf{V}}^t)^H \bar{\mathbf{V}}^t \\ &= \begin{bmatrix}\tilde{\mathbf{V}}^{t1}\tilde{\mathbf{T}}^{t1} \\ \tilde{\mathbf{V}}^{t2}\tilde{\mathbf{T}}^{t2}\end{bmatrix}^H \begin{bmatrix}\bar{\mathbf{V}}^{t1}\bar{\mathbf{T}}^{t1} \\ \bar{\mathbf{V}}^{t2}\bar{\mathbf{T}}^{t2}\end{bmatrix} \\ &= \sum_{\mathrm{i}} (\tilde{\mathbf{T}}^{t\mathrm{i}})^H(\tilde{\mathbf{V}}^{t\mathrm{i}})^H\bar{\mathbf{V}}^{t\mathrm{i}}\bar{\mathbf{T}}^{t\mathrm{i}} \\ &= \sum_{\mathrm{i}} (\tilde{\mathbf{T}}^{t\mathrm{i}})^H\hat{\mathbf{B}}^{t\mathrm{i}}\bar{\mathbf{T}}^{t\mathrm{i}}\end{aligned} \tag{3.13}$$

3. Finally, update coupling matrix for admissible block

$$\tilde{\mathbf{S}}^{t,s} = \tilde{\mathbf{B}}^t\mathbf{S}^{t,s}(\tilde{\mathbf{B}}^s)^H \tag{3.14}$$

If want final result to be of form $\mathbf{VSV}^T$ instead of $\mathbf{VSV}^H$. Should update coupling using below equation instead of using (3.14)

$$\tilde{\mathbf{S}}^{t,s} = \tilde{\mathbf{B}}^t\mathbf{S}^{t,s}(\hat{\mathbf{B}}^s)^T \tag{3.15}$$

The correctness of above procedure can be easily verified by substitute the corresponding expression for $\tilde{\mathbf{V}}^t$, $\tilde{\mathbf{S}}^{t,s}$ and $\tilde{\mathbf{V}}^s$ into $\tilde{\mathbf{V}}^t\tilde{\mathbf{S}}^{t,s}\tilde{\mathbf{V}}^s$. It is easy to find the $\tilde{\mathbf{V}}^t\tilde{\mathbf{S}}^{t,s}\tilde{\mathbf{V}}^s \approx \mathbf{VSV}^H$ with some derivation.

The method in chapter 2 takes 5 tree traversals to update the FMM $\mathcal{H}^2$-matrix to rank-minimized $\mathcal{H}^2$-matrix, while it only this procedure 2. One for all cluster basis, another for all block cluster basis.

3.1.2 Complexity Analysis

For leaf level, complexity is $O(N)$.

For non-leaf level, the complexity of computing $\mathbf{B}^t = \sum_{\mathrm{i}}(\mathbf{T}^{t\mathrm{i}})^H\mathbf{B}_a^{t\mathrm{i}}(\mathbf{B}_a^{t\mathrm{i}})^H\mathbf{T}^{t\mathrm{i}} = \mathbf{P}^t\mathbf{D}^t(\mathbf{P}^t)^H$ is $O(k_S^2k_F)$. Here k_S is the rank revealed by SVD and k_F is FMM rank.

The complexity of computing $\tilde{\mathbf{T}}^{t\mathrm{i}} = \tilde{\mathbf{B}}^{t\mathrm{i}}\mathbf{T}^{t\mathrm{i}}\mathbf{P}^t(\mathbf{D}^t)^{-\frac{1}{2}}$ is $O(k_Fk_{S\mathrm{i}})+O(k_{s\mathrm{i}}k_Sk_F) = O(k_S^2k_F)$.

The complexity of computing $\tilde{\mathbf{B}}^t = (\tilde{\mathbf{T}}^{t1})^H\tilde{\mathbf{B}}^{t1}\mathbf{T}^{t1} + (\tilde{\mathbf{T}}^{t2})^H\tilde{\mathbf{B}}^{t2}\mathbf{T}^{t2}$ is $O(k_{S\mathrm{i}}k_Sk_{F\mathrm{i}}) + O(k_Sk_F) = O(k_S^2k_F)$.

For coupling matrix, complexity is $O(k_S^2k_F)$

3.1.3 Converting from $\mathbf{VSV}^H$ $\mathcal{H}^2$-matrix to $\mathbf{VSV}^T$ $\mathcal{H}^2$-matrix

In the above procedure, we incorporate the process of converting a $\mathcal{H}^2$-matrix from $\mathbf{VSV}^H$to $\mathbf{VSV}^T$ in the rank-minimization process. Here we summarise this process. Let original cluster basis and coupling matrix be $\mathbf{V}^t, \mathbf{S}, \mathbf{V}^s$, and updated ones be $\tilde{\mathbf{V}}^t, \tilde{\mathbf{S}}, \tilde{\mathbf{V}}^s$. Here $\mathbf{V}^t$ should be orthonormal. We want to have

$$\tilde{\mathbf{V}}^t\tilde{\mathbf{S}}^{t,s}(\tilde{\mathbf{S}}^s)^T = \mathbf{V}^t\mathbf{S}^{t,s}(\mathbf{V}^s)^H \tag{3.16}$$

Let $\tilde{\mathbf{V}}^t = \mathbf{V}^t$, We have

$$\tilde{\mathbf{S}}^{t,s} = \mathbf{S}^{t,s}(V^s)^H\bar{\mathbf{V}}^s \tag{3.17}$$

Thus, we can do one tree traversal on cluster tree to compute $(\mathbf{V}^s)^H\bar{\mathbf{V}}^s$ for every cluster. Then do one tree traversal on block cluster tree to update coupling matrix using (3.17).

3.2 Nested Reduction Algorithm (NRA)

In this section, we still start from MLFMA, whose complexity is low for electrically large IEs as compared to interpolation or ACA-based methods, to build $\mathcal{H}^2$-matrix whose rank is minimized based on accuracy. But different from the method shown in Chapter 2 and the method shown in the previous chapter, the proposed new algorithm is much simplified and takes much less CPU run time while retaining the same accuracy. The resultant rank-minimized $\mathcal{H}^2$-matrix facilitates both fast iterative and direct solutions. Comparisons with analytical Mie series solutions and reference solutions from a commercial tool have validated the accuracy and efficiency of the proposed method for solving electrically large surface IEs.

3.2.1 Proposed Work

In [8], a linear-complexity algorithm is developed for converting a constant-rank $\mathcal{H}^2$-matrix whose rank is not minimized for accuracy to a new rank-minimized $\mathcal{H}^2$-matrix. FMM will naturally generate an $\mathcal{H}^2$-matrix, in which the rank of each cluster grows linear with the electrical size of this cluster, coupling matrix is diagonal, and transfer matrix is sparse in the sense that the number of non-zero elements in each column is a constant. Details are already discussed above and can be found in [10], [11]. In this section, we further simplify the procedure introduced in section 3.1. We name this method as Nested Reduction Algorithm (NRA) since we reduce the rank of each cluster t in a nested fashion.

For an arbitrary cluster t in the cluster tree, let its original cluster basis obtained from the FMM be $\mathbf{V}^t$. Such a $\mathbf{V}^t$ is of size $\#t$ by k_F, where $\#t$ denotes the number of unknowns contained in cluster t, and k_F is the rank resulting from the FMM. Since k_F is large, the algorithm presented here is to generate a new cluster basis $\tilde{\mathbf{V}}^t$ such that its rank k is the minimal one required by accuracy, and hence being much smaller than k_F. Certainly, such a rank reduction algorithm must retain the original nested property of the $\mathbf{V}$ across the cluster tree, while keeping the computational complexity low. In addition, for an electrically large analysis, both k and k_F are electrical size-dependent, and hence tree level dependent. For the simplicity of the notation, we would not add a tree-level dependence for k and k_F. But it should be noted that they are different at different tree levels. The k_F scales quadratically with the electrical size, whereas k scales linearly with the electrical size [9].

We start from the leaf level $l = \mathcal{L}$ (root level is at $l = 0$) of the cluster tree. For a leaf cluster t, we perform an SVD on $\mathbf{V}^t$. Based on prescribed accuracy ϵ, we keep the singular vectors whose singular values normalized by the maximum one are no less than ϵ. These singular vectors make the new leaf cluster basis $\tilde{\mathbf{V}}^t$. Thus, we obtain

$$\mathbf{V}^t_{\#t \times k_F} \overset{\epsilon}{\approx} \tilde{\mathbf{V}}^t_{\#t \times k} (\tilde{\mathbf{U}}^t)^H_{k \times k_F}, \tag{3.18}$$

where the subscripts denote the matrix dimension, and $(\tilde{\mathbf{U}}^t)^H$ is the other factor resulting from the SVD. Since $\#t$ is bounded by $leafsize$ which is a constant, the cost of (3.18) is constant, which is small.

We then proceed to the non-leaf level. For each non-leaf cluster t, its cluster basis is related to its children clusters' bases as shown in (1.6). Now the children clusters' bases have been changed. Hence, the transfer matrices must be updated for nonleaf cluster t. To see how to update them, we can substitute (3.18) into (1.6) obtaining

$$\mathbf{V}^t = \begin{bmatrix} (\tilde{\mathbf{V}}^{t_1}) & \\ & (\tilde{\mathbf{V}}^{t_2}) \end{bmatrix} \begin{bmatrix} (\tilde{\mathbf{U}}^{t_1})^H \mathbf{T}^{t_1} \\ (\tilde{\mathbf{U}}^{t_2})^H \mathbf{T}^{t_2} \end{bmatrix}, \tag{3.19}$$

from which it can be seen that

$$\tilde{\mathbf{G}}^t = \begin{bmatrix} (\tilde{\mathbf{U}}^{t_1})^H \mathbf{T}^{t_1} \\ (\tilde{\mathbf{U}}^{t_2})^H \mathbf{T}^{t_2} \end{bmatrix} \tag{3.20}$$

makes the new transfer matrix of cluster t. However, its rank may not be the minimal one required by accuracy. Therefore, we perform another SVD on (3.20) and truncate singular vectors based on accuracy ϵ to obtain the new transfer matrix $\tilde{\mathbf{T}}^t$. As a result, we have

$$\tilde{\mathbf{G}}^t_{(k_1+k_2) \times k_F} \overset{\epsilon}{\approx} \tilde{\mathbf{T}}^t_{(k_1+k_2) \times k} (\tilde{\mathbf{U}}^t)^H_{k \times k_F}, \tag{3.21}$$

where the rank $k_1 + k_2$, which is the sum of the rank of the two children's new cluster bases, is further reduced to k based on prescribed accuracy.

The aforementioned procedure at a non-leaf level is then repeated level by level up, until we reach the minimal level that has admissible blocks. At that level, we finish generating the new cluster bases. After that, we update the orginal FMM-based coupling matrix, $\mathbf{S}^{t,s}$, as follows for each admissible block

$$\tilde{\mathbf{S}}^{t,s} = (\tilde{\mathbf{U}}^t)^H \mathbf{S}^{t,s} \tilde{\mathbf{U}}^s. \tag{3.22}$$

The overall procedure is a bottom-up tree traversal procedure summarized as follows, which is termed the Nested Reduction Algorithm (NRA).

At each tree level of the cluster tree, we do the following computation:

1. For a leaf cluster t, do (3.18) to obtain new cluster basis $\tilde{\mathbf{V}}^t$ based on required accuracy ϵ.

2. For a non-leaf cluster t, compute (3.20) to obtain $\tilde{\mathbf{G}}^t$, then factorize it to obtain (3.21), and hence new transfer matrix $\tilde{\mathbf{T}}^t$ as well as $(\tilde{\mathbf{U}}^t)^H$, based on accuracy ϵ.

If column cluster bases are different from row cluster bases such as those in an unsymmetrical matrix, Steps 1 and 2 are repeated for column cluster bases. After cluster basis update, for each admissible block, we update the coupling matrix based on (3.22).

At a non-leaf level, the cost of computing (3.20) is only of $O(kk_F)$, which is $O(k^3)$, since transfer matrix $\mathbf{T}$ is sparse. However, the subsequent step of computing the SVD of $\hat{\mathbf{T}}^t$ can be costly, which is $O(k^2 k_F)$, and hence scaling as $O(k^4)$.

We apply these steps recursively to every cluster in cluster tree and every admissible block in $\mathcal{H}^2$-matrix, and formalized it into the form of an algorithm as shown in algorithm 8. In line 2 of algorithm 8 we call algorithm 9 on the root of cluster tree, which recursively reduces the rank of all clusters. In line 3 we apply algorithm 10 to the root of $\mathcal{H}^2$-matrix to compute new coupling matrix $\tilde{\mathbf{S}}$ for all admissible blocks recursively.

After these steps, the new cluster basis's rank is minimized based on accuracy ϵ. Furthermore, they become orthonormal matrices. In this algorithm, the action is either exact or approximate using SVD under a prescribed criterion, and thus is accurate. As for the complexity, at the leaf level, it is $O(N)$, where N is matrix size. For non-leaf levels, the

Algorithm 8 nested_reduction_algorithm

This algorithm shows Nested Reduction Algorithm

```
1: procedure nested_reduction_algorithm($t, b$)
2:     nested_update_cluster_basis($t$)
3:     nested_update_coupling_matrix($b$)
4: end procedure
```

Algorithm 9 nested_update_cluster_basis

This algorithm shows how to update cluster basis in a nested way

```
1: procedure nested_update_cluster_basis($t$)
2:     if $t$ is non-leaf cluster then
3:         for i is t's child do
4:             nested_update_cluster_basis($t^{\mathrm{i}}$)
5:         end for
6:         compute $\tilde{\mathbf{G}} = \begin{bmatrix} (\tilde{\mathbf{U}}^{t1})^H \tilde{\mathbf{T}}^{t1} \\ (\tilde{\mathbf{U}}^{t2})^H \tilde{\mathbf{T}}^{t2} \end{bmatrix}$
7:         do SVD on $\tilde{\mathbf{G}} = \hat{T}^t (\tilde{\mathbf{U}}^t)^H$
8:         Splite $\hat{\mathbf{T}}$ to get new transfer matrix: $\begin{bmatrix} \tilde{\mathbf{T}}^{t1} \\ \tilde{\mathbf{T}}^{t2} \end{bmatrix} = \hat{\mathbf{T}}$
9:     else
10:        do SVD on $\mathbf{V}^t$ such that $\mathbf{V}^t = \tilde{\mathbf{V}}^t (\tilde{\mathbf{U}})^H$
11:    end if
12: end procedure
```

Algorithm 10 nested_update_coupling_matrix

This algorithm updates a coupling matrix in a nested fashion.

```
1: procedure nested_update_coupling_matrix($b$)
2:     if $b$ is admissible block then
3:         $\tilde{\mathbf{S}}^{t,s} = (\tilde{\mathbf{U}}^t)^H \mathbf{S}^{t,s} \tilde{\mathbf{U}}^s$
4:     else if i is $b$'s sons then
5:         nested_update_coupling_matrix($b^{\mathrm{i}}$)
6:     end if
7: end procedure
```

complexity of computing $\tilde{\mathbf{G}}^t$ is $O(k_S k_F)$, since $\mathbf{T}^{ti}$ is a sparse matrix in the sense that every column only has a constant number of non-zero elements. Doing SVD on $\tilde{\mathbf{G}}^t$ takes $O(k_S^2 k_F)$ operations. Storing $\tilde{\mathbf{U}}^t$ for each cluster needs $O(k_S k_F)$ memory. Thus, for each non-leaf cluster, the time complexity is $O(k_S^2 k_F)$ and the memory cost is $O(k_S k_F)$. The complexity of computing coupling matrices $\tilde{\mathbf{S}}^{t,s} = (\tilde{\mathbf{U}}^t)^H \mathbf{S}^{t,s} \tilde{\mathbf{U}}^s$ is $O(k_S^2 k_F)$.

3.2.2 Numerical Result

The scattering of a conducting sphere at 300 MHz is considered. The mesh density is chosen to be about $\frac{\lambda}{10}$, and the radius of the sphere is increased to examine the electrical size dependence of the proposed algorithm. We use FMM to generate an initial $\mathcal{H}^2$-matrix of an EFIE. The number of Gaussian quadrature points is three on each triangular patch. $Leafsize$ is set to be 40, and $\eta = 1.2$. We set $L = kd + 1.8 d_0^{\frac{2}{3}} (kd)^{\frac{1}{3}}$ as the truncation number used in the addition theorem, where k is wave number, d is the diameter of a cluster, and $d_0 = \log_{10}(1e4)$. For the criterion of an admissible block, we use $d \leq \eta D$ and $L \leq kD$ at the same time, where d is the largest diameter of two clusters and D is the distance between them. For coupling matrices, Lagrange interpolation is used for larger sizes. For small transfer matrices, spherical harmonics are employed for global interpolation; for large ones that are sparse, six points are used for Lagrange interpolation along both θ and ϕ direction. The ϵ in the converting process is set to be 1e-3. We validate the proposed algorithm using a sphere whose diameter is 17.68m, in which N=294,912. We compared the new converted $\mathcal{H}^2$-matrix solved by direct solver [7] with the MIE series. The result is shown in Fig. 3.1

Next, we validate the proposed algorithm together with the direct solver for a PEC cube and compare the RCS with HFSS, the condition is set identical with the above example. This cube is of size $12.8\lambda \times 12.8\lambda \times 12.8\lambda$. The number of unknowns involved is 294,912. The result is shown in Fig. 4.1

Next, we validate the NRA together with the direct solver for a complicated structure, a coil. A coil is shown as Fig. 3.3. This coil is of size 14.156 m. After discretization, there are 121,914 unknowns. As always, the new $\mathcal{H}^2$-matrix generated from the Fast NRA is directly

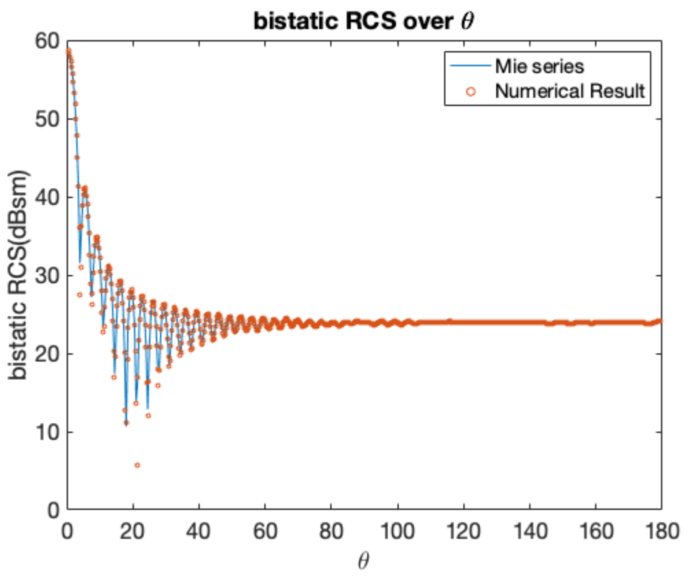


Figure 3.1. Compare RCS of conducting sphere

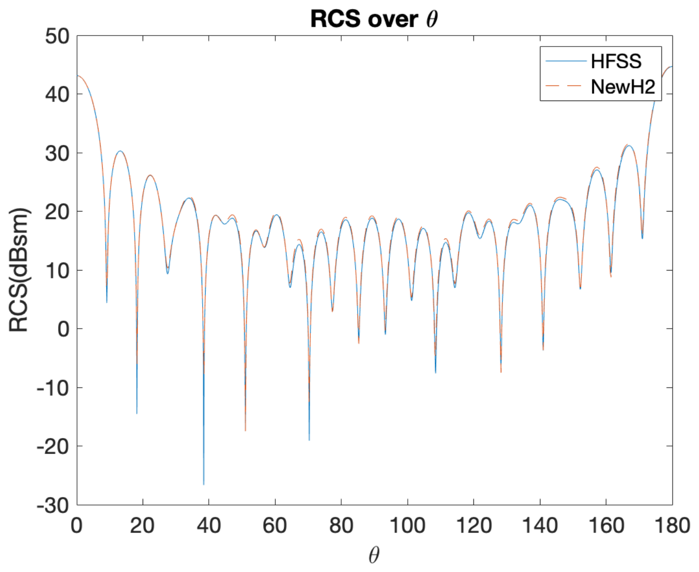


Figure 3.2. Compare RCS of conducting Cube

solved using the solver in [7], and the bistatic RCS is extracted and compared with HFSS. Result is shown at Fig. 3.4.

Another complicated structure is a joint, which is shown in 3.5. The conditions are the same as the previous one. This joint has a diameter of 17.302m. After discretization, there

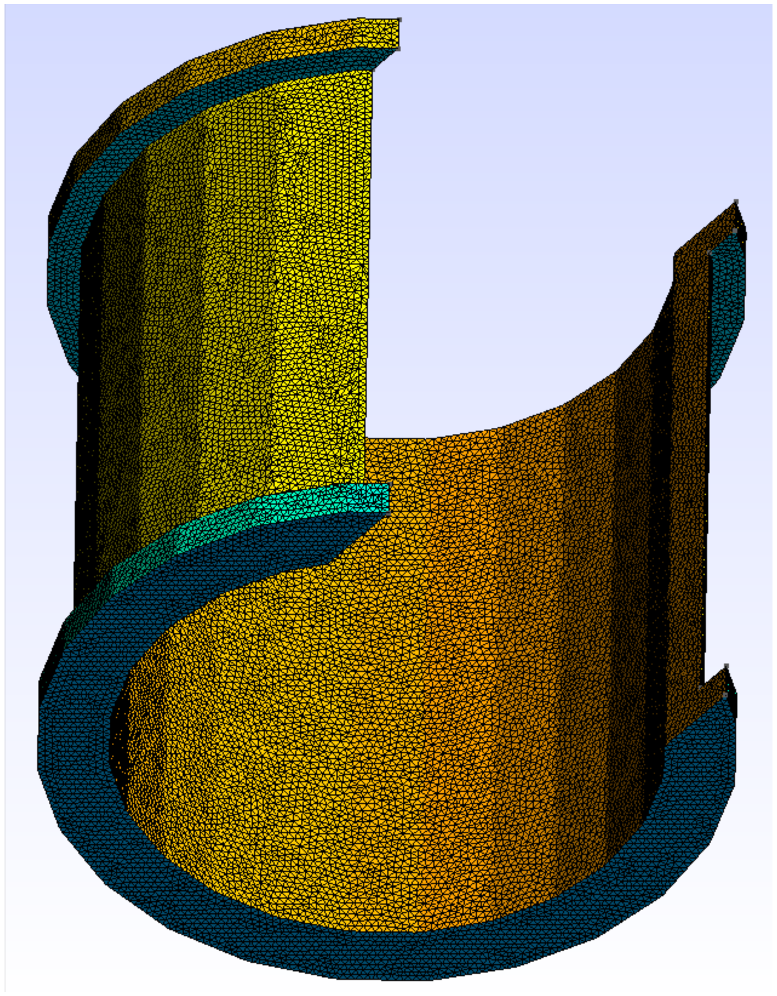


Figure 3.3. A coil.

are 172,077 unknowns. The comparison between results of HFSS and Fast NRA solved by the direct solver is shown in Fig. 3.6.

Next in Table 3.1, we show the rank of New $\mathcal{H}^2$-matrix with original FMM $\mathcal{H}^2$-matrix'rank. In this table, e$_s$ is the largest electrical size of all clusters in this level, N is the largest unknowns of all clusters in a tree level, r_F is the FMM's rank (FMM rank is the same for all this level's clusters regardless the size of cluster), r_N is the rank of New $\mathcal{H}^2$-matrix after we merge $\mathbf{Z}_\phi$, $\mathbf{Z}_{A_x}$, $\mathbf{Z}_{A_y}$, $\mathbf{Z}_{A_z}$ together, as we shown in [10]. $r_{N,\phi}$ is the rank of New $\mathcal{H}^2$-matrix for only $\mathbf{Z}_\phi$ using the method introduced in this article. This shows that our new $\mathcal{H}^2$-matrix's rank is much reduced compared to the original FMM's rank.

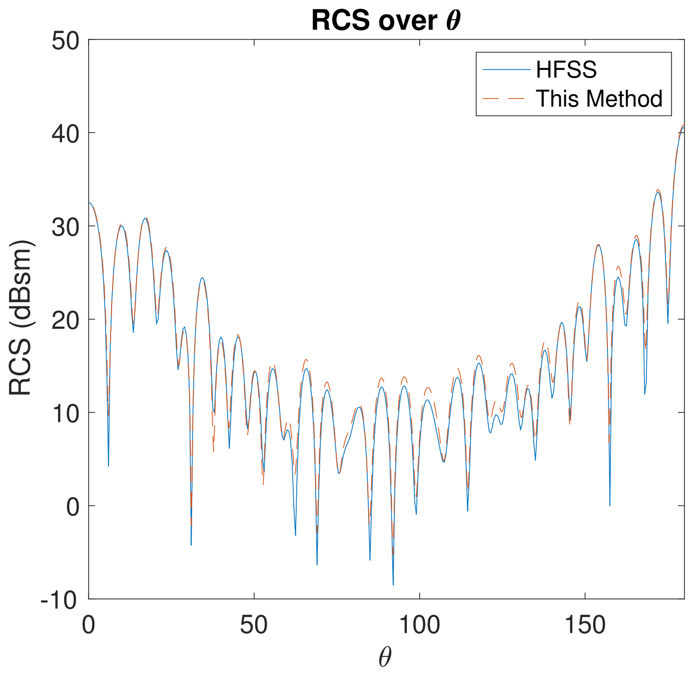


Figure 3.4. RCS of a coil simulated using Fast NRA.

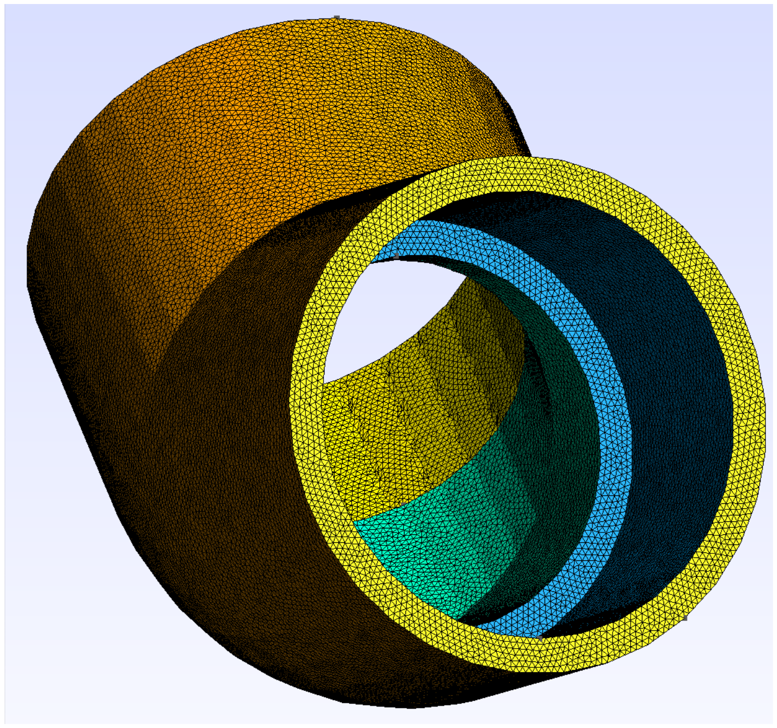


Figure 3.5. A joint.

Table 3.1. Rank versus tree level using Nested Reduction for $\epsilon = 10^{-3}$

tree level	e_s	N	r_F	r_N	$r_{N,\phi}$
3	20.61	49152	42050	967	954
4	15.74	24576	25538	721	552
5	10.26	12288	11552	509	323
6	8.29	6144	7938	324	204
7	5.26	3072	3698	207	125
8	4.48	1536	2738	134	86
9	2.79	768	1250	90	57
10	2.46	384	1058	63	43
11	1.50	192	512	44	31
12	1.39	96	450	32	25
13	0.88	48	242	25	18
14	0.84	24	242	18	15

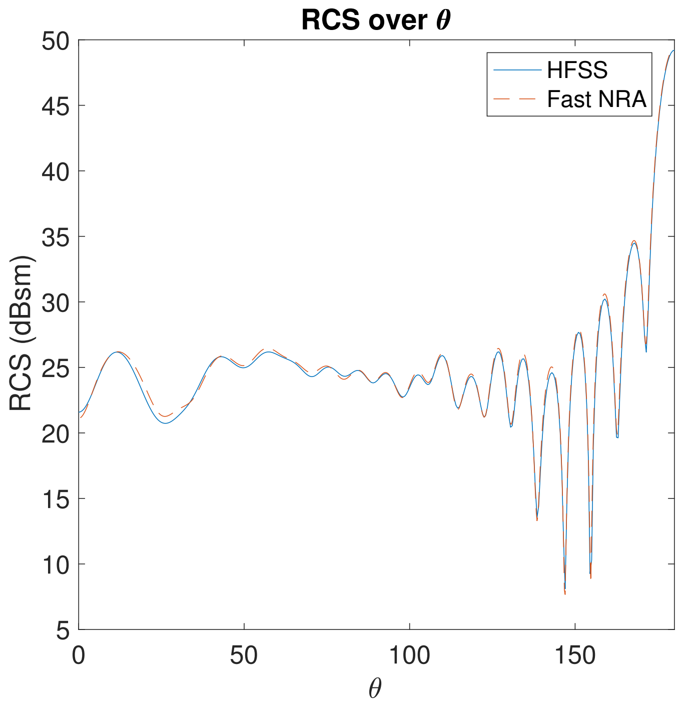


Figure 3.6. RCS of a joint simulated using Fast NRA.

4. NRA WITH FURTHER REDUCED COMPLEXITY

In the above mentioned algorithm NRA, at a non-leaf level, the cost of computing ($\tilde{\mathbf{G}}^t = \begin{bmatrix} (\tilde{\mathbf{U}}^{t1})^H \mathbf{T}^{t1} \\ (\tilde{\mathbf{U}}^{t2})^H \mathbf{T}^{t2} \end{bmatrix}$) is only of $O(kk_F)$, which is $O(k^3)$, since transfer matrix $\mathbf{T}$ is sparse. However, the subsequent step of computing the SVD of $\tilde{\mathbf{G}}^t$ can be costly, which is $O(k^2 k_F)$, and hence scaling as $O(k^4)$. To circumvent this cost, we propose a fast algorithm as the following.

4.1 Fast NRA

For a non-leaf cluster t, in ($\tilde{\mathbf{G}}^t = \begin{bmatrix} (\tilde{\mathbf{U}}^{t1})^H \mathbf{T}^{t1} \\ (\tilde{\mathbf{U}}^{t2})^H \mathbf{T}^{t2} \end{bmatrix}$), we randomly choose $O(k_1 + k_2)$ columns to compute its low-rank factorization ($\tilde{\mathbf{G}}^t = \hat{\mathbf{T}}^t (\tilde{\mathbf{U}}^t)^H$) instead of operating on the full k_F columns. This can be done because the rank of $\tilde{\mathbf{G}}^t$ is no greater than its row rank, which is $k_1 + k_2$. We then compute a full cross approximation (FCA) [6] with prescribed accuracy on the randomly selected $O(k_1 + k_2)$ columns of $\tilde{\mathbf{G}}^t$, obtaining row pivots τ and column pivots σ. As a result, $\tilde{\mathbf{G}}^t$ is factorized to

$$\tilde{\mathbf{G}}^t \overset{\epsilon}{\approx} \tilde{\mathbf{G}}^t_{:,\sigma} (\tilde{\mathbf{G}}^t_{\tau,\sigma})^{-1} \tilde{\mathbf{G}}^t_{\tau,:} \tag{4.1}$$

whose new rank is k. Since the full cross approximation is performed on the $O(k_1 + k_2)$ columns of $\tilde{\mathbf{G}}^t$, the entire computational cost of obtaining (4.1) is reduced to $O(k^3)$ as compared to $O(k^4)$ from a brute-force SVD-based low-rank compression of $\tilde{\mathbf{G}}^t$. In (4.1), $\tilde{\mathbf{G}}^t_{:,\sigma}$ denotes the selected k columns of $\tilde{\mathbf{G}}^t$, whose indexes are contained in σ. The $\tilde{\mathbf{G}}^t_{:,\sigma}$ multiplied by the following small k by k matrix, $(\tilde{\mathbf{G}}^t_{\tau,\sigma})^{-1}$, is nothing but the new transfer matrix of t, thus

$$\hat{\mathbf{T}}^t = \tilde{\mathbf{G}}^t_{:,\sigma} (\tilde{\mathbf{G}}^t_{\tau,\sigma})^{-1}, \tag{4.2}$$

whereas the

$$(\tilde{\mathbf{U}}^t)^H = \tilde{\mathbf{G}}^t_{\tau,:}, \tag{4.3}$$

is the k rows of $\tilde{\mathbf{G}}^t$, whose row indexes are contained in τ. In this way, the $(\tilde{\mathbf{U}}^t)^H$ can be obtained in $O(k^3)$ cost because it involves k-rows of $(\tilde{\mathbf{U}})^H$ multiplied by a sparse $\mathbf{T}$.

It is worth mentioning the full cross approximation instead of adaptive cross approximation is employed here because the accuracy of the former is better than the latter. Furthermore, for a low-rank matrix, the accuracy of an FCA is guaranteed. Certainly, the SVD can be directly used on the selected $O(k)$ columns to obtain new bases, which has a reduced cost of $O(k^3)$ as well. However, if we do that, the subsequent step of obtaining $(\tilde{\mathbf{U}}^t)^H$ would cost more than $O(k^3)$.

The pseudo-code of the aforementioned fast NRA is shown in **Algorithm** 11. In line 2 of this algorithm, **Algorithm** 12 is called starting from the leaf level of the cluster tree, which factorizes $\mathbf{V} = \tilde{\mathbf{V}}(\tilde{\mathbf{U}})^H$ for each leaf cluster, and then factorizes $\tilde{\mathbf{G}}^t = \hat{\mathbf{T}}(\tilde{\mathbf{U}})^H$ for each non-leaf cluster. **Algorithm** 12 yields new rank-minimized cluster bases based on accuracy. In line 3 we apply **Algorithm** 13 to the root of the block cluster tree of the $\mathcal{H}^2$-matrix, which updates the coupling matrix of each admissible block.

Algorithm 11 Nested_Reduction_Algorithm

```
 1: procedure nested_reduction_algorithm($t, b$)
 2:     update_cluster_basis($t$)
 3:     update_coupling_matrix($b$)
 4: end procedure
```

Algorithm 12 Update_Cluster_Basis

This algorithm efficiently obtains new nested cluster bases with rank minimized based on accuracy

```
 1: procedure randomized_update_cluster_basis($t$)
 2:     if $t$ is a non-leaf cluster then
 3:         Compute $\tilde{\mathbf{G}} = \begin{bmatrix} (\tilde{\mathbf{U}}^{t1})^H \mathbf{T}^{t1} \\ (\tilde{\mathbf{U}}^{t2})^H \mathbf{T}^{t2} \end{bmatrix}$
 4:         Randomly select $\|c^t\|$ columns from $\tilde{\mathbf{G}}$ to form $\tilde{\mathbf{G}}_w$.
 5:         Do FCA based on $\epsilon$ on $\tilde{\mathbf{G}}_w$ to get $\tau$ and $\sigma$
 6:         Obtain $(\tilde{\mathbf{U}}^t)^H = \tilde{\mathbf{G}}^t_{\tau,:}$, $\hat{\mathbf{T}} = \tilde{\mathbf{G}}^t_{:,\sigma}(\tilde{\mathbf{G}}^t_{\tau,\sigma})^{-1}$
 7:     else
 8:         Do SVD on $\mathbf{V}^t$ such that $\mathbf{V}^t = \tilde{\mathbf{V}}^t(\tilde{\mathbf{U}}^t)^H$
 9:     end if
10: end procedure
```

The procedure is also outlined as follow,

Algorithm 13 Update_Coupling_Matrix

This algorithm updates coupling matrices

```
1: procedure UPDATE_COUPLING_MATRIX($b$)
2:     if $b$ is an admissible block then
3:         $\tilde{\mathbf{S}}^{t,s} = (\tilde{\mathbf{U}}^t)^H \mathbf{S}^{t,s} \tilde{\mathbf{U}}^s$
4:     else if i is $b$'s child then
5:         update_coupling_matrix($b^{\mathrm{i}}$)
6:     end if
7: end procedure
```

1. For leaf cluster t, do ACA or rSVD to reveal the rank of cluster basis $\mathbf{V}^t = \tilde{\mathbf{V}}^t(\tilde{\mathbf{U}}^t)^H$. Denote this rank by k_S^t.

2. For non-leaf cluster t, randomly choose k column pivot set c^t. the size of this set (denoted by $\|c^t\|$) is decided by the rank of t's two sons, $t1$ and $t2$. Essentially, $\|c^t\| = c(k_S^{t1} + k_S^{t2})$, where c is enlarging coefficient. Denote $\tilde{\mathbf{G}}^t = \begin{bmatrix} (\tilde{\mathbf{U}}^{t1})^H \mathbf{T}^{t1} \\ (\tilde{\mathbf{U}}^{t2})^H \mathbf{T}^{t2} \end{bmatrix}$. We just compute c^t columns of $\tilde{\mathbf{G}}^t$. Denote this abbreviated version of $\tilde{\mathbf{G}}^t$ matrix by $\tilde{\mathbf{G}}_w$. Do Cross Approximation with full pivoting on $\tilde{\mathbf{G}}_w$ to determine the row pivot set and column pivot set τ, σ. Then we can compute $\hat{\mathbf{T}} = \tilde{\mathbf{G}}^t_{:,\sigma}(\tilde{\mathbf{G}}^t_{\tau,\sigma})^{-1}$. Then split $\hat{\mathbf{T}} = \begin{bmatrix} \tilde{\mathbf{T}}^{t1} \\ \tilde{\mathbf{T}}^{t2} \end{bmatrix}$ to get new transfer matrix. Also, form $(\tilde{\mathbf{U}}^t)^H = \tilde{\mathbf{G}}^t_{\tau,:}$ for this cluster's parent to use.

3. For coupling matrix, we have $\tilde{\mathbf{S}}^{t,s} = (\tilde{\mathbf{U}}^t)^H \mathbf{S}^{t,s} \tilde{\mathbf{U}}^s$

From the aforementioned cost analysis given along with the description of the algorithm, it can be seen that the time cost of obtaining new cluster bases $\tilde{\mathbf{V}}^t$ is low, which is $O(k^3)$ for each cluster. Here, notice that k is the minimal rank required by accuracy instead of the original FMM's rank. However, the storage of $\tilde{\mathbf{U}}^t$ would cost $O(kk_F)$ units for each cluster, thus being $O(k^3)$. In the next section, we propose another NRA to reduce the memory costs.

4.2 Memory Efficient NRA (Double Recursive Algorithm)

We further improve the algorithm in the previous section. In this section, we introduce an algorithm that only needs $O(nk \log n)$ in time and $O(k^2)$ in memory for each cluster and each coupling matrix.

In this new algorithm, there are also two steps. One is to generate new cluster basis whose rank is minimized, and the other is to update coupling matrices, similar to the algorithm presented in previous section. However, we do not explicitly compute or store $\tilde{\mathbf{U}}$ matrix for each cluster. As can be seen from (3.18) and (3.21), if the new cluster basis $\tilde{\mathbf{V}}^t$ is made unitary, then $(\tilde{\mathbf{U}}^t)^H$ is nothing but the projection of the original basis onto the new basis, thus

$$(\tilde{\mathbf{U}}^t)^H = (\tilde{\mathbf{V}}^t)^H \mathbf{V}^t. \tag{4.4}$$

Hence, whenever $(\tilde{\mathbf{U}}^t)^H$ is involved in computation, we can utilize the nested property of both the original basis $\mathbf{V}^t$ and the new basis $(\tilde{\mathbf{V}}^t)$ to compute it efficiently. Storage wise, we only need to store the new cluster basis $(\tilde{\mathbf{V}}^t)$ whose rank is minimized, and the original cluster basis $\mathbf{V}^t$ which is sparse, thus bypassing the storage of $(\tilde{\mathbf{U}}^t)^H$. Next, we elaborate this algorithm.

At the leaf level, the computation is the same as that in the previous algorithm, where SVD is used to obtain $\tilde{\mathbf{V}}^t$ so that each leaf cluster basis is unitary. At a non-leaf level, for each cluster t, we randomly select $O(k_1 + k_2)$ columns of $\mathbf{T}^t$ to compute (3.20). Let this set be c^t. Computing c^t-columns of $\hat{\mathbf{T}}^t$ is the same as computing c^t-columns of $(\tilde{\mathbf{V}}^t)^H_{ch}\mathbf{V}^t$, where $(\tilde{\mathbf{V}}^t)_{ch}$ is a block diagonal matrix containing t's two children's new cluster bases. To see this more clearly, we can rewrite (3.20) as

$$\tilde{\mathbf{G}}^t = \begin{bmatrix} (\tilde{\mathbf{V}}^{t_1})^H & \\ & (\tilde{\mathbf{V}}^{t_2})^H \end{bmatrix} \begin{bmatrix} \mathbf{V}^{t_1}\mathbf{T}^{t_1} \\ \mathbf{V}^{t_2}\mathbf{T}^{t_2} \end{bmatrix}, \tag{4.5}$$

which is nothing but

$$\tilde{\mathbf{G}}^t = (\tilde{\mathbf{V}}^t)^H_{ch}\mathbf{V}^t. \tag{4.6}$$

For each index c in the set c^t, we form a cardinal vector $\mathrm{e}^{\mathbf{c}}$, which has only one non-zero element at the c-th entry. Multiplying $(\tilde{\mathbf{V}}^t)^H_{ch}\mathbf{V}^t$ by $\mathrm{e}^{\mathbf{c}}$ is the same as computing $\mathbf{V}^t\mathrm{e}^{\mathbf{c}}$ first, and then multiplying the resultant vector by $(\tilde{\mathbf{V}}^t)^H_{ch}$, each of which costs $O(n \log n)$ complexity using the nested property of both bases, where n is the size of cluster t. After obtaining the c^t columns of $\tilde{\mathbf{G}}^t$, we perform an SVD on it based on prescribed accuracy ϵ to obtain new transfer matrix $\tilde{\mathbf{T}}^t$ whose rank is reduced to k. The cost of this step is $O(k^3)$. In addition, such a new transfer matrix is unitary. The aforementioned procedure of computing new transfer matrix $\tilde{\mathbf{T}}$ at a non-leaf level continues level by level up, until the highest level having admissible block is reached.

The pseudo-code of the new algorithm is shown in **Algorithm** 14, in which the fast matrix-vector multiplication algorithm for $\mathbf{V}^t$ and $(\tilde{\mathbf{V}}^t)^H$ are shown in **Algorithm** 15, and **Algorithm** 16 respectively.

Algorithm 14 new_update_cluster_basis

This algorithm is for new cluster basis generation.

```
 1: procedure New Update Cluster Basis($t$)
 2:     if $t$ is a non-leaf cluster then
 3:         Randomly select $c^t$ pivots from $k_F$ columns of
            $t$'s original transfer matrix $\mathbf{T}^t$
 4:         for $c \in c^t$ do
 5:             $\boldsymbol{\omega}^t = \mathrm{e}^{\boldsymbol{c}}$
 6:             recursively_multi_old_trans($t, c$)
 7:             recursively_multi_new_trans($t, c$)
 8:         end for
 9:         Use the resultant $c^t$ columns of $\hat{\mathbf{T}}^t$ to form $\hat{\mathbf{T}}^t_w$
10:         Do SVD on $\hat{\mathbf{T}}^t_w$ based on $\epsilon$ to get $\tilde{\mathbf{T}}^t$
11:     else
12:         Do SVD on $\mathbf{V}^t$ to obtain $\tilde{\mathbf{V}}^t$
13:     end if
14: end procedure
```

Algorithm 15 recursively_multi_old_trans

This algorithm performs a top-down tree travserval to compute $\mathbf{V}\omega$

```
 1: procedure recursively_multi_old_trans($t, \boldsymbol{\omega}$)
 2:     if $t$ is a non-leaf cluster then
 3:         for i is $t$'s child do
 4:             $\boldsymbol{\omega}^{\mathrm{i}} = \mathbf{T}^{\mathrm{i}}\boldsymbol{\omega}$
 5:             recursively_multi_old_trans($t_{\mathrm{i}}, \boldsymbol{\omega}^{\mathrm{i}}$)
 6:         end for
 7:     else
 8:         $\boldsymbol{\omega}^t = \mathbf{V}^t\boldsymbol{\omega}^t$
 9:     end if
10: end procedure
```

Algorithm 16 recursively_multi_new_trans

This Algorithm performs a bottom-up tree travserval to compute $\tilde{\mathbf{V}}^H\omega$

```
1: procedure recursively_multi_new_trans($t, \boldsymbol{\omega}$)
2:     if $t$ is a non-leaf cluster then
3:         for i is $t$'s child do
4:             recursively_multi_new_trans($t_{\mathrm{i}}, \boldsymbol{\omega}^{\mathrm{i}}$)
5:         end for
6:         $\boldsymbol{\omega}^t \leftarrow \begin{bmatrix} (\tilde{\mathbf{T}}^{t_1})^H \boldsymbol{\omega}^1 \\ (\tilde{\mathbf{T}}^{t_2})^H \boldsymbol{\omega}^2 \end{bmatrix}$
7:     else
8:         $\boldsymbol{\omega}^t \leftarrow (\tilde{\mathbf{V}}^t)^H \boldsymbol{\omega}^t$
9:     end if
10: end procedure
```

After generating new cluster bases, next we update the coupling matrix of each admissible block. Similarly, utilizing (4.4), (3.22) becomes the computation of

$$\tilde{\mathbf{S}}^{t,s} = (\tilde{\mathbf{V}}^t)^H \mathbf{V}^t \mathbf{S}^{t,s} (\mathbf{V}^s)^H \tilde{\mathbf{V}}^s. \tag{4.7}$$

Since $\tilde{\mathbf{V}}$ basis is of rank k, $\tilde{\mathbf{V}}^s$ has only k columns. The computation of $(\mathbf{V}^s)^H\tilde{\mathbf{V}}^s$ is the computation of k matrix-vector multiplications of $(\mathbf{V}^s)^H$ multiplying the k columns in $\tilde{\mathbf{V}}^s$. This can be done using **Algorithm** 16 where the $\tilde{}$ on top of the symbols is removed. This step costs $O(n \log n)$ for one vector. Hence the total cost of $(\mathbf{V}^s)^H\tilde{\mathbf{V}}^s$ is $O(kn \log n)$, which is $O(k^3 \log k)$ since n scales as k^2. Next, we multiply $\mathbf{S}^{t,s}$ by the computed $(\mathbf{V}^s)^H\tilde{\mathbf{V}}^s$. Since $\mathbf{S}^{t,s}$ is diagonal, the cost of this step is $O(k_F k)$, which is $O(k^3)$. After that, we multiply $(\tilde{\mathbf{V}}^t)^H\mathbf{V}^t$ by the k vectors resulting from $\mathbf{S}^{t,s}(\mathbf{V}^s)^H\tilde{\mathbf{V}}^s$, which again can be computed by k matrix-vector multiplications using $\mathbf{V}^t$, followed by the other k matrix-vector multiplications of $(\tilde{\mathbf{V}}^t)^H$. The pseudo-code of the new coupling matrix update algorithm is shown in **Algorithm** 17. The cost of this step is also $O(kn \log n)$ utilizing the nest property of the cluster bases.

The memory requirement of the new algorithm for each cluster is $O(k^2)$. This is because we do not store $\tilde{\mathbf{U}}$ for each cluster. Instead, we store $\tilde{\mathbf{V}}$ and $\mathbf{V}$. At the leaf level, the storage is a constant for each cluster. At a non-leaf level, the storage is a new transfer matrix of size $k \times k$ for each cluster. As for the original cluster basis $\mathbf{V}$, the storage is a transfer matrix of size $k_F \times k_F$ for each cluster. However, this transfer matrix is sparse, thus costing $O(k_F) \approx O(k^2)$ units to store also.

4.3 NRA taking advantage of sparsity

In this section, we take advantage of the sparsity of $\tilde{\mathbf{U}}^H$ to facilitate the NRA process. Table 4.1 shows the largest percentage of Number of Non-Zero (NNZ) elements in $\tilde{\mathbf{U}}^H$ of all the $\tilde{\mathbf{U}}^H$ in one case for different size of unknowns. We can see $\tilde{\mathbf{U}}^H$ becomes sparser and sparser when the number of unknowns goes up. Thus, taking advantage of this knowledge, we can construct algorithms that are complexity low.

Algorithm 17 double_recursive_update_coupling

This algorithm updates a coupling matrix at l in converting process. This algorithm computes $\tilde{S}^{t,s} = (\tilde{U}^t)^H S^{t,s} \tilde{U}^s$

```
1: procedure DOUBLE_RECURSIVE_UPDATE_COUPLING($b$)
2:     if $b$ is admissible block then
3:         for i $= 1, 2, ..., k_S^t$ do
4:             if t is non-leaf cluster then
5:                 for i is $t$'s sons do
6:                     $\boldsymbol{\omega}^{\mathrm{i}} \leftarrow (\tilde{T})_{(\mathrm{i},:)}$
7:                     recursively_multi_new_trans($t^{\mathrm{i}}, \boldsymbol{\omega}^{\mathrm{i}}$)
8:                 end for
9:                 recursively_multi_old_trans($t, \boldsymbol{\omega}$)
10:            else
11:                $row \leftarrow ((\tilde{U}^t)^H)_{(\mathrm{i},:)}$
12:            end if
13:            get $row$ by multiply $row$ with $S^{t,s}$
14:            $\omega^s = conj(row)$
15:            recursively_multi_old_trans($s, \boldsymbol{\omega}^s$)
16:            recursively_multi_new_trans($s, \boldsymbol{\omega}^s$)
17:            $\tilde{S}^{t,s}_{(\mathrm{i},:)} \leftarrow conj(row)$
18:        end for
19:    else if i is $b$'s sons then
20:        double_recursive_update_coupling($b$)
21:    end if
22: end procedure
```

Table 4.1. Sparsity of $\tilde{\mathbf{U}}^H$ against the number of unknowns for cubes.

N	4608	18432	73728	294912	1179648
sparsity $\tilde{\mathbf{U}}^H$	0.8351	0.8382	0.7584	0.6213	0.4582

4.3.1 NRA Using Sparsity

Below outlines the algorithm that takes advantages of the sparsity of $(\tilde{\mathbf{U}}^t)^H$. Note in below $\tilde{\mathbf{V}}$ and $\tilde{\mathbf{T}}$ are orthonormal matrix, and $\tilde{\mathbf{U}}$ is orthogonal matrix.

1. For leaf cluster, directly do SVD to get orthogonalized cluster basis such that $\mathbf{V} = \tilde{\mathbf{V}}\tilde{\mathbf{U}}^H$.

2. For non-leaf cluster t, randomly choose k column pivot set c^t. the size of this set (denoted by $\|c^t\|$) is decided by the rank of t's two sons, $t1$ and $t2$. Essentially, $\|c^t\| = c(k_S^{t1} + k_S^{t2})$, where c is enlarging coefficient. Denote $\tilde{\mathbf{G}}^t = \begin{bmatrix} (\tilde{\mathbf{U}}^{t1})^H \mathbf{T}^{t1} \\ (\tilde{\mathbf{U}}^{t2})^H \mathbf{T}^{t2} \end{bmatrix}$. We just compute c^t columns of $\tilde{\mathbf{G}}^t$. Denote this abbreviated version of $\tilde{\mathbf{G}}^t$ matrix by $\hat{\mathbf{G}}^t$. Then do full SVD on $\hat{\mathbf{G}}$. We can get left-singular vectors $\hat{\mathbf{T}} = \begin{bmatrix} \tilde{\mathbf{T}}^{t1} \\ \tilde{\mathbf{T}}^{t2} \end{bmatrix}$, which is orthogonalized transfer matrix. Such $\hat{\mathbf{T}}$ can represent the column span of $\hat{\mathbf{G}}^t$ and $\tilde{\mathbf{G}}^t$. Then use the relationship $\tilde{\mathbf{U}}^H = \tilde{\mathbf{T}}^{t1}(\tilde{\mathbf{U}}^{t1})^H \mathbf{T}^{t1} + \tilde{\mathbf{T}}^{t2}(\tilde{\mathbf{U}}^{t2})^H \mathbf{T}^{t2}$ to get this cluster's $\tilde{\mathbf{U}}^H$.

3. $\tilde{\mathbf{S}}^{t,s} = (\tilde{\mathbf{U}}^t)^H \mathbf{S}^{t,s} \tilde{\mathbf{U}}^s$

In this algorithm, first as always we do SVD on $\mathbf{V}$ for leaf cluster. For non-leaf cluster, we also pre-select k columns and assemble a shrunk submatrix $\hat{\mathbf{G}}^t$ from $\tilde{\mathbf{G}}^t$. Then wo do SVD on $\hat{\mathbf{G}}^t$ to get left-singular vectors $\hat{\mathbf{T}}^t = \begin{bmatrix} \tilde{\mathbf{T}}^{t1} \\ \tilde{\mathbf{T}}^{t2} \end{bmatrix}$. The difference here is how we compute $(\tilde{\mathbf{U}}^t)^H$. Since we want $(\tilde{\mathbf{U}}^t)^H$ to satisfy $\tilde{\mathbf{G}}^t = \hat{\mathbf{T}}^t(\tilde{\mathbf{U}}^t)^H$. We have

$$(\tilde{\mathbf{U}}^t)^H = (\hat{\mathbf{T}}^t)^H \tilde{\mathbf{G}}^t \tag{4.8}$$

This is

$$(\tilde{\mathbf{U}}^t)^H = \tilde{\mathbf{T}}^{t1}(\tilde{\mathbf{U}}^{t1})^H\mathbf{T}^{t1} + \tilde{\mathbf{T}}^{t2}(\tilde{\mathbf{U}}^{t2})^H\mathbf{T}^{t2} \tag{4.9}$$

given the expression of $\hat{\mathbf{T}}^t$ and $\tilde{\mathbf{G}}^t$. We can do (4.9) fast because $(\tilde{\mathbf{U}}^t)^H$, $(\tilde{\mathbf{U}}^{t1})^H$ and $(\tilde{\mathbf{U}}^{t2})^H$ are all sparse, so do $\mathbf{T}^{t1}$ and $\mathbf{T}^{t2}$. In this way, wo circumvent the need to do SVD on $\tilde{\mathbf{G}}^t$, which is costly. Finally, we do $\tilde{\mathbf{S}}^{t,s} = (\tilde{\mathbf{U}}^t)^H\mathbf{S}^{t,s}\tilde{\mathbf{U}}^s$ to update coupling matrix as always.

There are several ways to do (4.9) efficiently. First method is that we can compute each row of $(\tilde{\mathbf{U}}^t)^H$ at one time. In this way, we only need to deal with two sparse matrix-vector multiplications for each row of $\tilde{\mathbf{T}}^{t1}$ and $\tilde{\mathbf{T}}^{t2}$. Then for each row of $(\tilde{\mathbf{U}}^t)^H$, find the maximum. After finding the maximum for one row, free this row. Then find the maximum of the next row. Finally, we can find the maximum of the entire matrix. Then we use this maximum and prescribed accuracy ϵ to determine the threshold. Then we do the same procedure again to find which element is above the threshold to decide whether or not to keep this element. The second method is similar to the first one, only different in that each row's maximum is used to truncate that row. Thus, resultant $\tilde{\mathbf{U}}^H$ will be less sparse and takes more memory but is more accurate.

4.3.2 Eliminating Randomness

The above method is efficient but still relies on randomness when choosing randomly column pivot set c^t from $\tilde{\mathbf{G}}^t$. In the below algorithm, we eliminate the randomness while still preserving the complexity. In below, $\tilde{\mathbf{V}}^t$ and $\tilde{\mathbf{T}}^t$ are orthonormal matrix, and $\tilde{\mathbf{U}}^t$ is orthogonal matrix.

1. For leaf cluster, directly do SVD to get orthogonalized cluster basis such that $\mathbf{V} = \tilde{\mathbf{V}}\tilde{\mathbf{U}}^H$.

2. For non-leaf cluster, first compute $\tilde{\mathbf{G}}^t = \begin{bmatrix}(\tilde{\mathbf{U}}^{t1})^H\mathbf{T}^{t1}\\(\tilde{\mathbf{U}}^{t2})^H\mathbf{T}^{t2}\end{bmatrix}$, then compute $\tilde{\mathbf{G}}^t(\tilde{\mathbf{G}}^t)^H$. Then do SVD on $\tilde{\mathbf{G}}^t(\tilde{\mathbf{G}}^t)^H$ to get orthogonalized transfer matrix $\hat{\mathbf{T}}$. Then split $\hat{\mathbf{T}} = \begin{bmatrix}\tilde{\mathbf{T}}_1\\\tilde{\mathbf{T}}_2\end{bmatrix}$. Then use the relationship $(\tilde{\mathbf{U}}^t)^H = \tilde{\mathbf{T}}^{t1}(\tilde{\mathbf{U}}^{t1})^H\mathbf{T}^{t1} + \tilde{\mathbf{T}}^{t2}(\tilde{\mathbf{U}}^{t2})^H\mathbf{T}^{t2}$ to get this cluster's

$(\tilde{\mathbf{U}}^t)^H$. Note we store both $(\tilde{\mathbf{U}}^t)^H$ and $(\tilde{\mathbf{G}}^t)^H$ in sparse matrix format, and corresponding matrix operations are sparse ones.

3. $\tilde{\mathbf{S}}^{t,s} = (\tilde{\mathbf{U}}^t)^H \mathbf{S}^{t,s} \tilde{\mathbf{U}}^s$

In this new algorithm, we first get $\tilde{\mathbf{G}}^t = \begin{bmatrix} (\tilde{\mathbf{U}}^{t1})^H \mathbf{T}^{t1} \\ (\tilde{\mathbf{U}}^{t2})^H \mathbf{T}^{t2} \end{bmatrix}$, then compute $\tilde{\mathbf{G}}^t(\tilde{\mathbf{G}}^t)^H$. Then compute $\tilde{\mathbf{G}}^t(\tilde{\mathbf{G}}^t)^H$ and do SVD on it. Here the complexity is still low because $\tilde{\mathbf{G}}^t$ is also a sparse matrix. Doing SVD on $\tilde{\mathbf{G}}^t(\tilde{\mathbf{G}}^t)^H$ will give us the same $\hat{\mathbf{T}}$ as above algorithm. Following procedure is the same as above procedure.

4.4 Accuracy and Complexity

4.4.1 Accuracy

In all the algorithms in this chapter, almost all the actions are either exact or approximate using SVD or CA under a prescribed criterion and thus are accurate. The only exception is the action of randomly selecting pivots in Fast NRA, Double Recursive Procedure, and NRA using sparsity. We will later show the behavior of randomly selecting is accurate in Section 4.5.

In all the algorithms proposed in this chapter, only the steps of SVD or CA involve approximations. However, they are performed subject to a prescribed accuracy. Hence, the overall procedure is error controlled. In the CA part, we randomly select $O(k_1 + k_2) = c(k_1 + k_2)$ columns to perform CA, where c is a constant coefficient greater than 1. Since the matrix upon which CA is performed is known to be bounded by $k_1 + k_2$ in rank, and c is chosen to be greater than 1, the resultant cross approximation is ensured to produce an accurate rank-k representation. As can be seen from [12], the choice of k columns is not unique in a CA or ACA algorithm. As long as the k columns are linearly independent, they yield an accurate rank-k model. Different from ACA, in a CA, at every step, the maximum entry in the residual matrix is identified, whose row and column pivots are chosen to generate a rank-1 model. Hence, the accuracy of CA is guaranteed [13]. If it happens that the randomly selected $c(k_1 + k_2)$ columns do not contain k linearly independent columns, it can be identified in the CA process, and more columns can then be selected. The accuracy of randomly selecting k columns will also be numerically shown in Section 4.5.

4.4.2 Time and Memory Complexity

The complexity analysis is dependent on the rank's behavior. Consider a rank that grows linearly with the electrical size. This is also shown to be the minimal rank required by accuracy in an electrically large IE analysis [9]. Let n be the size of a cluster, then in an SIE, the rank k scales as

$$k = O(\sqrt{n}). \tag{4.10}$$

As for the rank of FMM, it scales quadratically with the electrical size, thus

$$k_F = O(n). \tag{4.11}$$

In the proposed fast NRA algorithm, every cluster basis needs $O(k^3 + k_F k) \approx O(k^3)$ operations to be computed. Based on (4.10) and (4.11), the total time complexity for new cluster basis generation can be computed as:

$$\begin{aligned} C_t &= \sum_{l=0}^{L} 2^l O(k k_F + k^3) \\ &= \sum_{l=0}^{L} 2^l O\left(\sqrt{\frac{N}{2^l}}\frac{N}{2^l} + \left(\sqrt{\frac{N}{2^l}}\right)^3\right) = O(N^{1.5}). \end{aligned} \tag{4.12}$$

The time complexity for coupling matrix updates can be computed as:

$$\begin{aligned} C_{t,S} &= \sum_{l=0}^{L} 2^l C_{sp} O(k^2 k_F) \\ &= \sum_{l=0}^{L} 2^l C_{sp} O\left(\sqrt{\frac{N}{2^l}}^2 \frac{N}{2^l}\right) = O(N^2), \end{aligned} \tag{4.13}$$

where C_{sp} denotes the maximal number of blocks formed by a single cluster, which is a constant [6]. The total memory complexity can be computed by adding the memory cost of each cluster basis with that of each admissible block as the following

$$\begin{aligned} C_m &= \sum_{l=0}^{L}(2^l O(k^3) + C_{sp} 2^l O(k^2)) \\ &\approx \sum_{l=0}^{L} 2^l O\left(\frac{N}{2^l}\right)^{1.5} = O(N^{1.5}). \end{aligned} \tag{4.14}$$

Hence, $O(N^{1.5})$ memory is required during the rank reduction. After the rank reduction is finished, the memory of storing the new rank-minimized $\mathcal{H}^2$-matrix would be just $O(N \log N)$ for SIE, since only $O(k^2)$ is required for storing each cluster basis, and each admissible block.

As for DRA shown in Section 4.2, the complexity for every cluster is $O(kn \log n)$ in time and $O(k^2)$ in memory. Adding the cost of every cluster across all tree levels, we obtain the total time cost as

$$\begin{aligned} C_t &= \sum_{l=0}^{L} 2^l O(kn \log n) \\ &= \sum_{l=0}^{L} 2^l O\left(\sqrt{\frac{N}{2^l}} \frac{N}{2^l} l\right) = O(N^{1.5} \log N). \end{aligned} \tag{4.15}$$

Similarly, updating all the admissible blocks has also $O(N^{1.5} \log N)$ complexity, since at each tree level, there are $2^l O(C_{sp})$ admissible blocks, each of which costs $O(kn \log n)$ operations to update. The total memory consumption including the memory required for both cluster basis generation and coupling matrix updates scales as

$$\begin{aligned} C_m &= \sum_{l=0}^{L}(2^l O(k^2) + C_{sp} 2^l O(k^2)) \\ &= \sum_{l=0}^{L} 2^l O\left(\frac{N}{2^l}\right) = O(N \log N) \end{aligned} \tag{4.16}$$

for electrically large SIE analyses.

It is worth mentioning that although the new algorithm of Section 4.2 has the same time complexity in cluster basis generation as compared to that in Section 3.2, the constant in front of the $N^{1.5}\log N$ is larger. As for the coupling matrix update, the DRA in Section 4.2 has a reduced complexity of $O(N^{1.5}\log N)$, but the constant is also larger. We notice that the absolute run time of the DAR in Section 4.2 can exceed that of the fast NRA algorithm in Section 3.2 when simulating medium-sized problems. However, memory and its scaling rate are reduced. In addition, from the aforementioned complexity analysis, it can be seen that for applications where the rank is a constant, the total complexity of the proposed algorithms is $O(N)$.

4.4.3 Further Rank Reduction

In the proposed NRA algorithms, the original FMM-based cluster bases are reduced in rank based on accuracy. To further explore the redundancy in the matrix content, we employ the algorithm in [14] on top of the new $\mathcal{H}^2$-matrix generated from the proposed NRA algorithms to further reduce its rank. The algorithm in [14] can be used to convert an $\mathcal{H}^2$-matrix whose rank is not minimized to a new one that is minimized based on accuracy. However, if the algorithm in [14] is directly applied to an FMM-based matrix, the conversion cost would be too high since the starting rank is high. In contrast, when using the $\mathcal{H}^2$-matrix generated from the proposed NRA algorithms to do further compression using [14], the cost is as low as $O(k^3)$ for each cluster and admissible block in time, and $O(k^2)$ in memory, thus not increasing the complexity of the proposed NRA algorithms.

4.5 Numerical Results

A common set of simulation parameters are used for all examples simulated in this section. Specifically, in the FMM, we set the truncation criterion of the addition theorem as $L = k_0 d + 1.8 d_0^{2/3}(k_0 d)^{\frac{1}{3}}$, where $d_0 = \log_{10}(\frac{1}{\epsilon_F})$ and $\epsilon_F = 10^{-2}$, k_0 is wavenumber, and d is the diameter of the targeted cluster. When generating an $\mathcal{H}^2$-representation of the FMM, we use 6 points along each of the θ and ϕ directions in the Lagrange polynomial based interpolation to obtain transfer matrices. In addition, we choose $\eta = 0.8$ in the admissibility condition and $leafsize$ to be 40. The accuracy criterion is set to be ϵ, which is a user-defined parameter, for FCA and SVD in the NRA algorithm. When randomly choosing $\#c^t = c(k_1 + k_2)$ columns from $\hat{\mathbf{T}}^t$, we choose c to be 4. In the case that the direct solver of [7] is used to solve the $\mathcal{H}^2$-matrix generated from the proposed algorithm, the accuracy is set to be 10^{-3}.

4.5.1 Accuracy

We first validate the accuracy of the proposed algorithms before examining their complexity in time and memory. A common set of simulation parameters are used for all examples simulated in this section. Specifically, in the FMM, we set the truncation criterion of the addition theorem as $L = k_0 d + 1.8 d_0^2 (k_0 d)^{\frac{1}{3}}$, where $d_0 = \log_{10}(\frac{1}{\epsilon_F})$ and $\epsilon_F = 10^{-2}$, k_0 is wavenumber and d is the diameter of the targeted cluster. When generating an $\mathcal{H}^2$-representation of the FMM, we use 12 points along each of the θ and ϕ directions to do the Lagrange Polynomial based interpolation to obtain transfer matrices. In addition, we choose $\eta = 1.2$ in the admissibility condition. When using the proposed NRA to do rank reduction, the accuracy criterion set for all ACA or SVD is ϵ. When choosing $\#c^t = c(k_1 + k_2)$ columns from $\hat{\mathbf{T}}^t$, we choose c to be 4.

Accuracy Comparison Between the Proposed Fast NRA Algorithm and the DRA

In **Algorithm** 12, a fast algorithm is developed to bypass the cost of the SVD in the proposed NRA algorithm. In this algorithm, $O(k)$ columns are randomly selected to perform CA, out of the k_F columns of the original FMM-based transfer matrix. Here, we examine the accuracy of this approach as compared to the brute-force NRA. We take a random vector x, and perform a matrix-vector multiplication using the new $\mathcal{H}^2$-matrix generated by the brute-force NRA to obtain $\mathbf{Z}_{\mathcal{H}^2} x$. We also use the fast NRA to compute $\mathbf{Z}_{fast\mathcal{H}^2} x$. The error of the resultant vector is then assessed by comparing with the original FMM-based matrix-vector multiplication, $\mathbf{Z}_{FMM} x$. The results are shown in Table 4.2 for $\epsilon = 10^{-2}$, and $\epsilon = 10^{-3}$ respectively. In this table, $err_0 = \|\mathbf{Z}_{\mathcal{H}^2} x - \mathbf{Z}_{FMM} x\| / \|\mathbf{Z}_{FMM} x\|$ represents the relative error of the new $\mathcal{H}^2$ generated by the brute-force NRA, while $err_1 = \|\mathbf{Z}_{Fast\mathcal{H}^2} x - \mathbf{Z}_{FMM} x\| / \|\mathbf{Z}_{FMM} x\|$ represents that of the fast NRA. As can be seen, the fast NRA is accurate, and its accuracy is also controllable like NRA. Similar results are also shown for DRA. In Table 4.3, err_0 remains the same, while $err_1 = \|\mathbf{Z}_{DRA\mathcal{H}^2} x - \mathbf{Z}_{FMM} x\| / \|\mathbf{Z}_{FMM} x\|$ represents the relative

Table 4.2. Accuracy Comparison Between NRA and Fast NRA

N	4608	18432	73728	294912	1179648
$err_0(\epsilon = 10^{-2})$	7.44e-3	1.07e-2	1.66e-2	2.07e-2	2.42e-2
$err_1(\epsilon = 10^{-2})$	6.92e-3	1.18e-2	1.83e-2	2.41e-2	4.51e-2
$err_0(\epsilon = 10^{-3})$	1.03e-3	1.13e-3	1.79e-3	2.28e-3	2.73e-3
$err_1(\epsilon = 10^{-3})$	9.42e-4	1.17e-3	1.95e-3	2.71e-3	

Table 4.3. Accuracy Comparison Between DRA and Fast NRA

N	4608	18432	73728	294912	1179648
$err_0(\epsilon = 10^{-2})$	7.417e-3	9.451e-3	1.609e-2	2.060e-2	2.289e-2
$err_1(\epsilon = 10^{-2})$	7.435e-3	9.607e-3	1.623e-2	2.072e-2	2.283e-2

error of the new $\mathcal{H}^2$ generated by the DRA. As we can see, DRA is also accurate. This also verifies our assumption before that the windowing technique will not affect the accuracy.

Scattering from a Conducting Cube Using Fast NRA

In this example, we compute the bistatic RCS of a conducting cube of size $12.8\lambda \times 12.8\lambda \times 12.8\lambda$ at 300 MHz, which has 294,912 unknowns. The new $\mathcal{H}^2$-matrix generated from the Fast NRA is solved using the direct solver of [7]. The resultant bistatic RCS is compared with that from HFSS, which reveals good agreement as can be seen from 4.1. In this example, the accuracy criterion used in the fast NRA is set to be $\epsilon = 10^{-3}$.

Scattering from a Conducting Plate Using Fast NRA

In this example, we compute the bistatic RCS of a conducting plate of size $60.8\lambda \times 60.8\lambda$ at 300 MHz, which has 1,107,776 unknowns. A BiCGStab iterative solver with a diagonal preconditioner is employed to solve the new $\mathcal{H}^2$-matrix generated from the proposed method. The result is then compared with HFSS and shown in Fig. 4.2. Again, very good agreement is observed, which validates the accuracy of the proposed algorithm. The simulation parameters are chosen the same as the previous cube example.

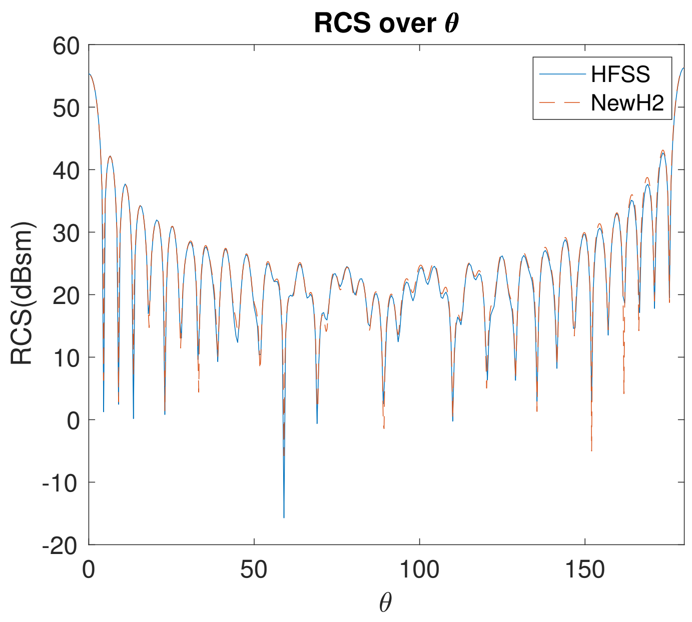


Figure 4.1. Compare RCS of conducting Cube Using Fast NRA

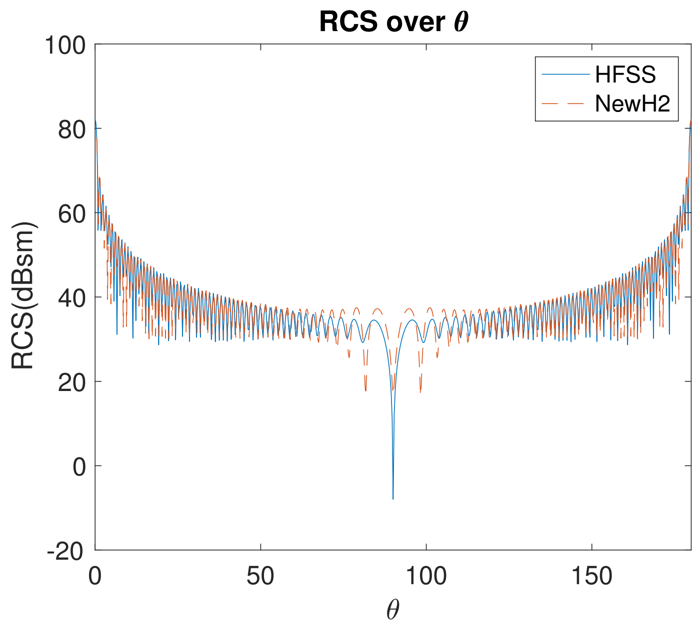


Figure 4.2. Compare RCS of conducting Plate Using Fast NRA

Scattering from an Array of Spheres Using Fast NRA

In this example, we simulate an array of spheres, having $4 \times 4 \times 4$ spheres, each of which has a diameter of 0.5525 m and is discretized with 188 unknowns at 300 MHz. The distance

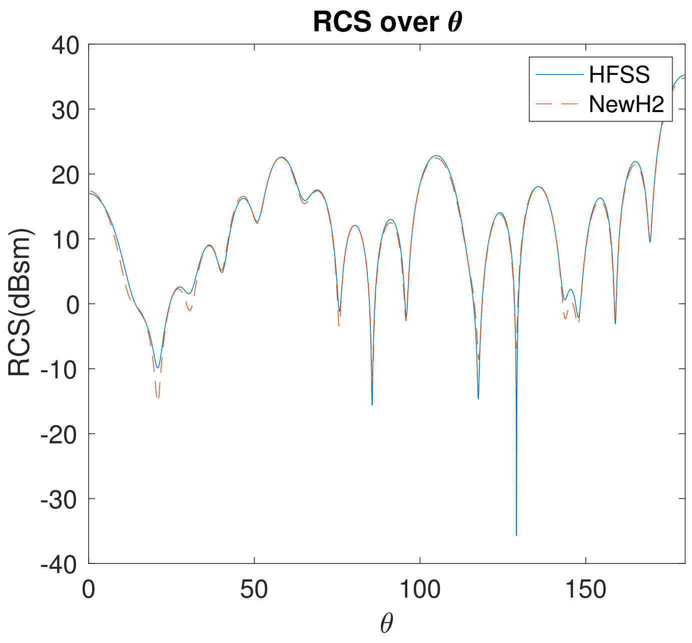


Figure 4.3. Simulated RCS of an array of conducting spheres $(4 \times 4 \times 4)$ Using Fast NRA.

between two adjacent spheres is 1.3820 m. The bistatic RCS is computed with a direct solution of the new $\mathcal{H}^2$-matrix by the direct solver in [7] and compared with HFSS. Good agreement is observed as can be seen from Fig. 4.3. The accuracy criterion used in the fast NRA is set to be $\epsilon = 10^{-3}$.

We also simulate another array having $6 \times 6 \times 6$ spheres, each of which is of diameter 0.8288 m and is discretized with 648 unknowns, yielding 139,968 unknowns in total at 300 MHz. The distance between two adjacent spheres is 2.0730 m. Again, the new $\mathcal{H}^2$ matrix generated from the proposed method is directly solved using the solver in [7], and the bistatic RCS is extracted and compared with HFSS. The comparison is shown in Fig. 4.4. which further validates the accuracy of the proposed algorithm. The simulation parameters are chosen the same as those in the previous example.

Scattering from two Complex Structures Using Fast NRA

We also simulate two complex structures 3.3and 3.5. This coil is of size 14.156 m. After discretization, there are 121,914 unknowns. This joint has a diameter of 17.302m.

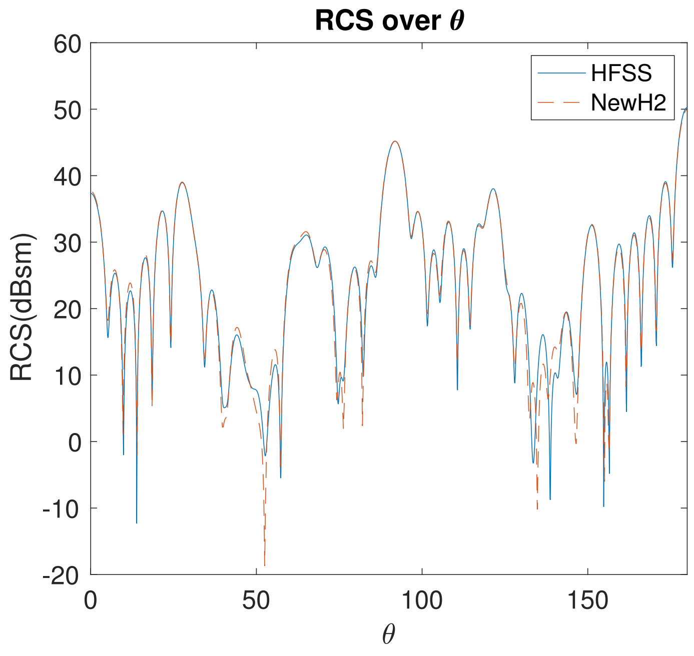


Figure 4.4. Simulated RCS of an array of conducting spheres ($6 \times 6 \times 6$) Using Fast NRA.

After discretization, there are 172,077 unknowns. The comparison between results of HFSS and DRA solved by the direct solver is shown at Fig. 4.5 and Fig. 4.6, for coil and joint, respectively.

Scattering from a Conducting Cube Using DRA

In this example, we compute the bistatic RCS of a conducting cube of size $12.8\lambda \times 12.8\lambda \times 12.8\lambda$ at 300 MHz, which has 294,912 unknowns. The new $\mathcal{H}^2$-matrix generated from the proposed method is solved using the direct solver of [7]. The resultant bistatic RCS is compared with that from HFSS, which shows very good agreement, as can be seen from Fig. 4.7. In this example, the accuracy criterion used in the DRA is set to be $\epsilon = 10^{-3}$. As we can see, the algorithm shows good accuracy as well.

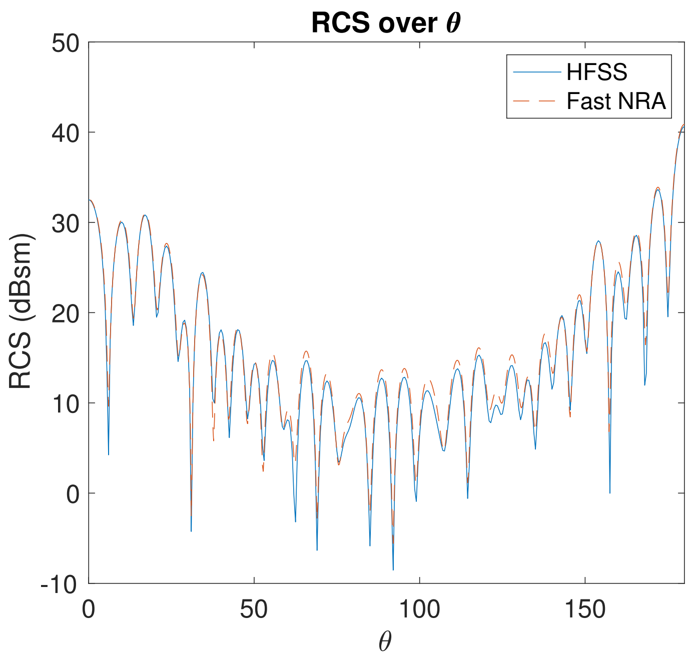


Figure 4.5. RCS of a coil simulated using Fast NRA.

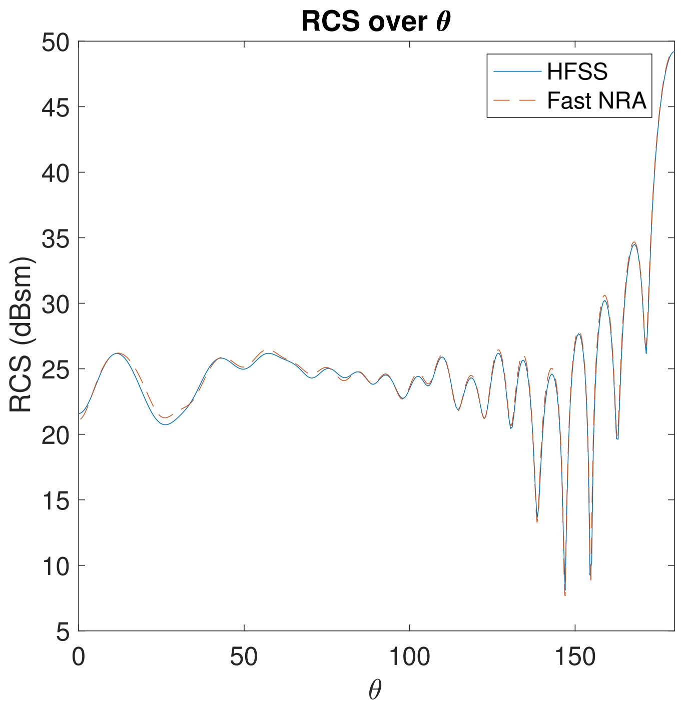


Figure 4.6. RCS of a joint simulated using Fast NRA.

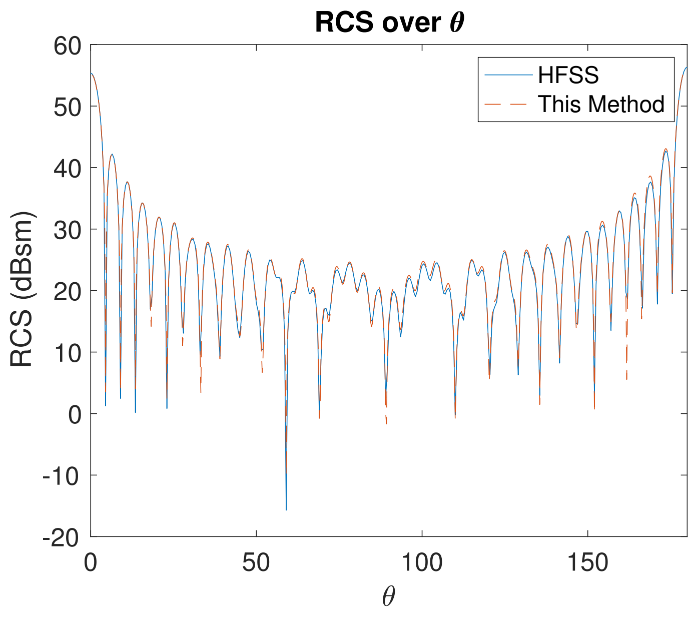


Figure 4.7. Compare RCS of conducting Cube Using DRA

4.5.2 Time and Memory Complexity

With the accuracy of the proposed algorithms validated, next, we examine the complexity of the proposed algorithms for generating a rank-minimized $\mathcal{H}^2$-matrix for electrically large analysis.

The Growth Rate of the Rank

First, we examine the growth rate of the rank with electrical size since it is one of the key parameters in the complexity analysis. We use a conducting sphere as an example, and find its rank level by level using the Fast NRA with $\epsilon = 10^{-3}$. The results are listed in Table 4.4. In this table, e_s denotes the largest electrical size of all clusters at a tree level, n is the largest unknown number of all clusters, k_F is the FMM's rank (note that in FMM, all cluster bases at the same tree level share the same rank in common), k is the rank of the new $\mathcal{H}^2$-matrix of the entire SIE obtained from the proposed reduction algorithm, and k_ϕ is the rank of the new $\mathcal{H}^2$-matrix for $\mathbf{Z}_\phi$ part only. We also plot the new rank as a function of

Table 4.4. Rank versus tree level using NRA with $\epsilon = 10^{-3}$

tree level	e_s	n	k_F	k	k_ϕ
3	20.61	49152.00	42050	967	954
4	15.74	24576.00	25538	721	552
5	10.26	12288.00	11552	509	323
6	8.29	6144.00	7938	324	204
7	5.26	3072.00	3698	207	125
8	4.48	1536.00	2738	134	86
9	2.79	768.00	1250	90	57
10	2.46	384.00	1058	63	43
11	1.50	192.00	512	44	31
12	1.39	96.00	450	32	25
13	0.88	48.00	242	25	18
14	0.84	24.00	242	18	15

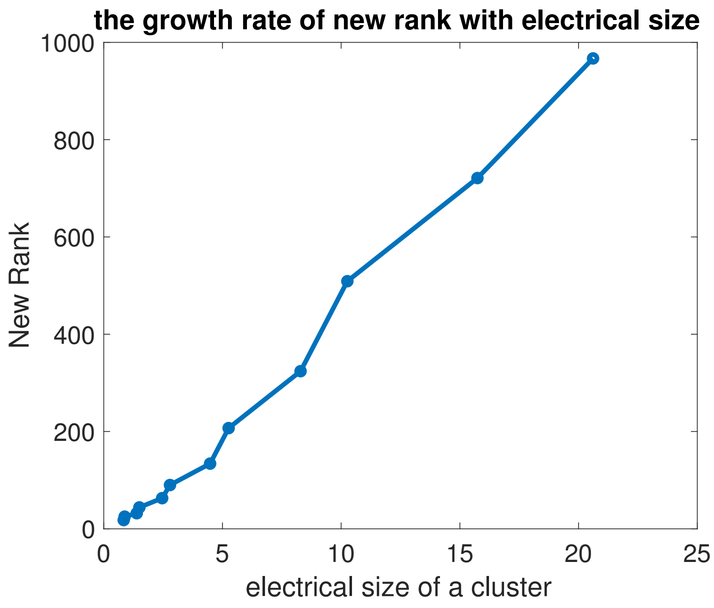


Figure 4.8. New rank's growth rate with electrical size in a sphere example.

electrical size in Fig. 4.8. From this figure, it can also be seen clearly that the rank scales linearly with electrical size, which agrees with the one used in our complexity analysis.

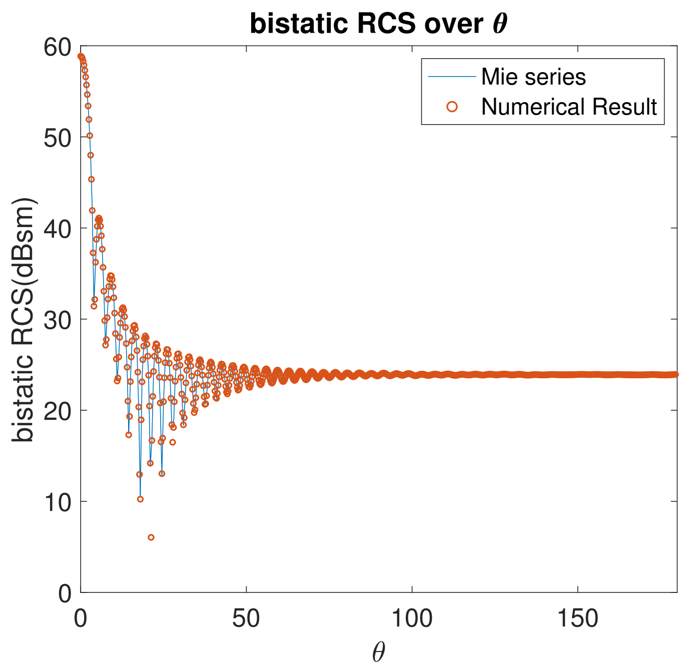


Figure 4.9. Simulated RCS of a conducting sphere using Fast NRA in comparison with Mie series solution.

Complexity Analysis

A suite of conducting spheres of various diameters at 300 MHz is then simulated to examine the time and memory complexity of the proposed fast NRA algorithm. First, as a sanity check of the accuracy, we simulate one case, which is a sphere of 17.68 m diameter, whose number of unknowns is 294,912. The accuracy criterion is chosen to be $\epsilon = 10^{-3}$ in the fast NRA algorithm. We then use the direct solver in [7] to solve the $\mathcal{H}^2$ matrix generated from the proposed algorithm. In Fig. 4.9, the simulated bistatic RCS is plotted as a function of θ, which reveals an excellent agreement with MIE series solution [15].

We then vary the size of the sphere and examine the time and memory scaling of the proposed fast NRA algorithm. The scaling data are listed in Table 4.5. In this Table, N is the number of unknowns, D is the diameter of the conducting sphere, e_{MV} denotes the relative error between the result of an FMM-based matrix multiplied by a vector and the new $\mathcal{H}^2$-matrix multiplied by the same vector, $t_{C,\phi}$ is the time for converting the ϕ part of $\mathbf{Z}$ matrix, t_C is the time for converting the whole $\mathbf{Z}$ matrix and $t_{FMM}(s)$ is the assembly time

Table 4.5. Time, rank, and memory scaling of the proposed fast NRA with $\epsilon = 10^{-2}$ and comparison with the FMM-based representation.

N	4608	18432	73728	294912	1179648
D	2.21λ	4.42λ	8.84λ	17.68λ	35.38λ
e$_{MV}$	6.92e-3	1.18e-2	1.83e-2	2.41e-2	4.51e-2
$t_C(s)$	114.75	1180.56	7570.54	46893.61	295287.38
$t_{C,\phi}(s)$	30.29	287.69	1874.19	10269.44	70441.52
$t_{FMM}(s)$	72.96	283.85	1188.37	4798.54	18731.23
$k_{F,\phi}$	288	1152	3698	11858	42050
k_ϕ	21	50	115	302	898
$t_{F,MV}(s)$	5.53	29.00	75.26	209.48	719.94
$t_{MV}(s)$	1.27	1.89	5.61	19.90	65.12
$m_F(Mb)$	278.36	1429.50	6633.92	27883.57	115991.14
$m_{gF}(Mb)$	318.25	1622.83	7481.95	31392.16	130411.31
$m(Mb)$	176.05	776.78	3416.73	15623.0	80965.49
$m_C(Mb)$	312.78	1793.66	8844.36	42692.75	231295.43

of FMM. The $k_{F,\phi}$ is the rank of the FMM $\mathcal{H}^2$-matrix's ϕ part, k_ϕ is the rank of the new $\mathcal{H}^2$-matrix's ϕ part, $t_{F,MV}(s)$ is the time of the FMM $\mathcal{H}^2$-matrix multiplied by a vector, $t_{N,MV}(s)$ is the time of the new $\mathcal{H}^2$-matrix multiplied by the same vector, $m_F(Mb)$ is the memory to store the FMM-based $\mathcal{H}^2$-matrix, $m_{gF}(Mb)$ is the memory used to assemble the FMM-based $\mathcal{H}^2$-matrix, $m(Mb)$ is the memory to store the new $\mathcal{H}^2$-matrix, $m_C(Mb)$ is the maximal memory used in the converting process. It can be seen clearly the new $\mathcal{H}^2$-matrix generated from the proposed work has a much reduced rank, memory, and matrix-vector multiplication time as compared to the FMM-based representation. We also plot the time and memory usage in log scale in Fig. 4.10 and Fig. 4.11 for the minimal-rank $\mathcal{H}^2$-generation time. They are shown to agree very well with our theoretical complexity analysis. We also plot the comparison between memory needed to store an FMM $\mathcal{H}^2$-matrix and a new $\mathcal{H}^2$-matrix generated by Fast NRA in Fig. 4.12. We also plot the comparison between the time needed for FMM representation and a new $\mathcal{H}^2$-matrix generated by Fast NRA to do Matrix-Vector Multiplication with the same vector in Fig. 4.13. We can see Matrix-Vector Multiplication

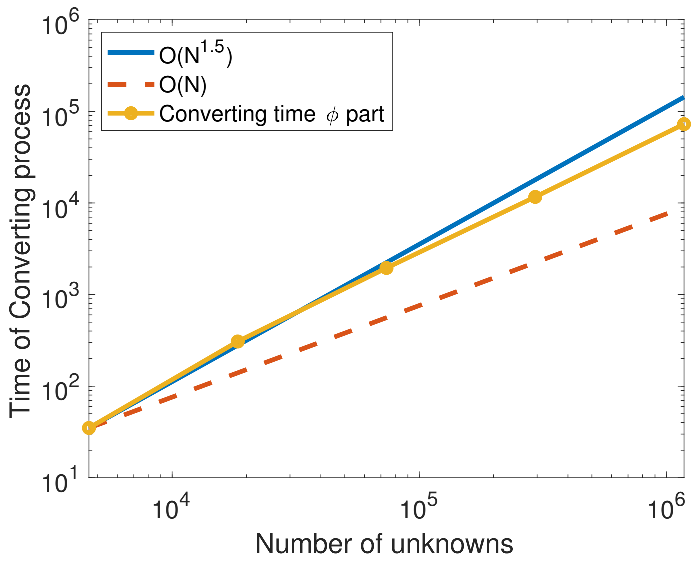


Figure 4.10. Time Complexity of the proposed fast NRA.

time is much reduced. This is not only because new $\mathcal{H}^2$-matrix needs less memory, but also because new $\mathcal{H}^2$-matrix contains only dense matrices, which is implemented well in modern library and is much faster when doing any computation. This shows we indeed get a more compact representation of the IE operator, which can be advantageous in the subsequent procedure for not only the direct solver but also the iterative solver, especially in the case of multiple right-hand sides. In addition, we simulated this example using the NRA described in Section 4.2. The CPU run time does not show advantages compared to the Fast NRA algorithm of Section 3.2. However, the memory is reduced. In Fig. 4.14, we plot its memory scaling of DRA in comparison with the Fast NRA algorithm of Section 3.2. As can be seen clearly, the algorithm incurs less memory and shows a reduced scaling rate with N.

Finally, the time and memory scaling is also examined for NRA using sparsity. The scaling data are listed in Table. 4.6, Table. 4.9, 4.7 and 4.8. Table. 4.6 shows the results of only hard-thresholding $(\tilde{\mathbf{U}}^t)^H$ in two ways. First is to threshold $(\tilde{\mathbf{U}}^t)^H$'s each row based on its row's own maximum. The second is to threshold $(\tilde{\mathbf{U}}^t)^H$ based on its entire matrix maximum. e$_{ad}$ is the relative error of all the admissible blocks between converted H2-matrix and original FMM matrix. e$_{all}$ is the whole matrix relative error between converted H2-matrix and original FMM matrix. e$_{MV}$ is the matrix-vector multiplication relative error

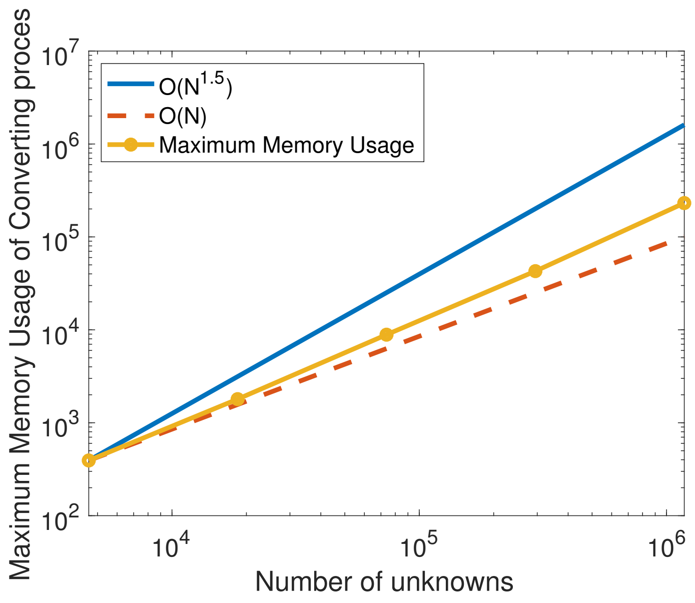


Figure 4.11. Memory Complexity of the proposed fast NRA.

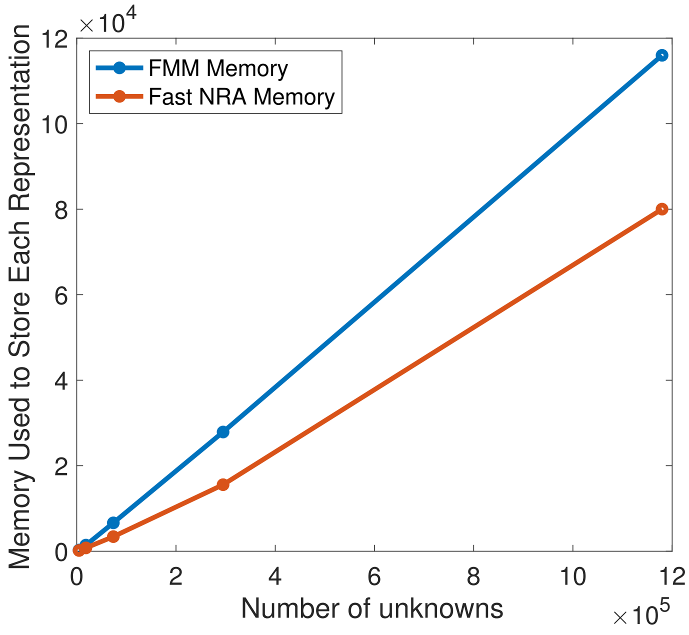


Figure 4.12. comparison of memory needed to store FMM representation and Fast NRA representation

obtained by using converted H2-matrix and original FMM matrix multiplied by the same vector. t(s) means the time consumed by the converting process, mem(Mb) means the memory needed to do the converting process. Table 4.7 corresponds the method in 4.3.2,

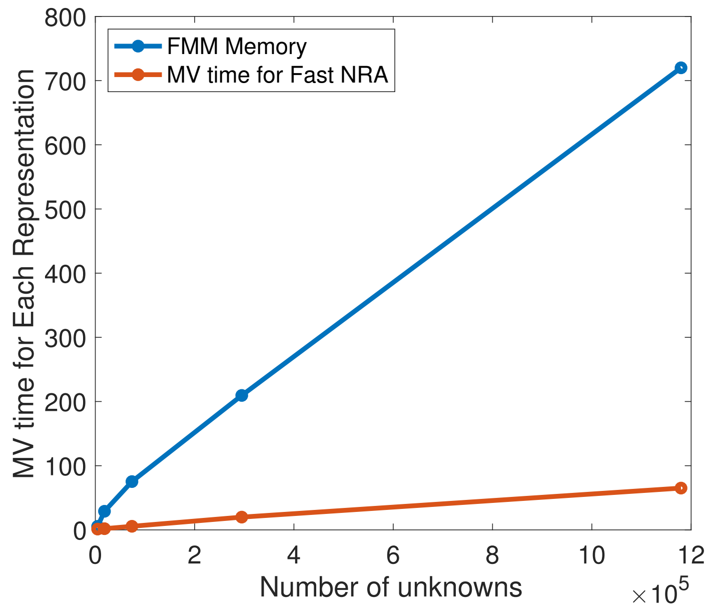


Figure 4.13. comparison of time needed for FMM representation and Fast NRA representation to do MV with the same vector

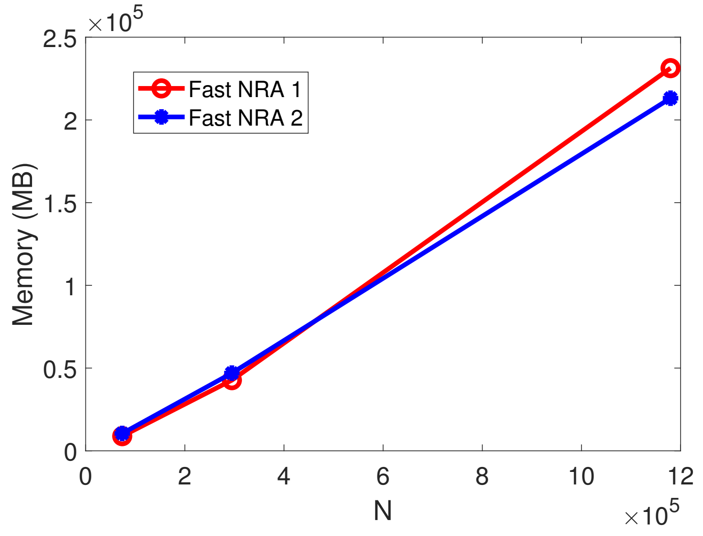


Figure 4.14. Memory comparison between Fast NRA 1 and Fast NRA 2 during the conversion stage.

which eliminates the randomness in the converting process. I also tested the method that hard-threshold $\tilde{\mathbf{G}} = \begin{bmatrix} \tilde{\mathbf{U}}_1^H \mathbf{T}_1 \\ \tilde{\mathbf{U}}_2^H \mathbf{T}_2 \end{bmatrix}$. In this method, we first hard-threshold $(\tilde{\mathbf{U}}^t)^H$ based on entire matrix's max for every cluster. Then we hard-threshold $(\tilde{\mathbf{U}}_1^t)^H \mathbf{T}_1$ and $(\tilde{\mathbf{U}}_2^t)^H \mathbf{T}_2$. Finally we

Table 4.6. Accuracy of NRA using sparsity which hard-threshold $(\tilde{\mathbf{U}}^t)^H$ for every cluster. Cube, $\epsilon = 10^{-2}$

N	4608	18432	73728	294912	1179648
e_{ad} (row max)	3.912e-2	3.915e-2	5.332e-2		
e_{all} (row max)	1.116e-3	2.084e-3	3.757e-3		
e_{MV} (row max)	4.648e-3	8.155e-3	1.174e-2	1.844e-2	2.260e-2
t(s) (row max)	607.25	5847.63	33407.47	175119.79	836089.41
mem(Mb) (row max)	475.90	2792.72	12115.56	54473.56	256710.23
e_{ad} (entire max)	3.926e-2	4.026e-2	5.573e-2		
e_{all} (entire max)	1.120e-3	2.144e-3	3.927e-3		
e_{MV} (entire max)	4.675e-3	8.377e-3	1.228e-2	2.049e-2	3.071e-2
t(s) (entire max)	608.09	6389.36	30649.30	150935.80	756467.80
mem(Mb) (entire max)	396.14	2480.02	11686.10	52775.24	240388.59

use SVD on $\tilde{\mathbf{G}}\tilde{\mathbf{G}}^H$ (Note $\tilde{\mathbf{G}} = \begin{bmatrix} \tilde{\mathbf{U}}_1^H \mathbf{T}_1 \\ \tilde{\mathbf{U}}_2^H \mathbf{T}_2 \end{bmatrix}$). Although $\tilde{\mathbf{G}}$ is more sparse, it takes more time than not doing so. Data corresponds to this method is shown in Tab. 4.8 for comparison. Tab. 4.9 shows data that does not do any hard-thresholding at all, including $\tilde{U}^H$ and $\tilde{\mathbf{G}}$. In these table, t_{conv} is the total converting time, $t_{conv,\phi}$ is the time of converting the $\mathbf{Z}_\phi$ part of the impedance $\mathbf{Z}$ matrix. $r_{V,\phi}$ is the new rank of $\mathbf{Z}_\phi$, while $r_{V,S}$ is the new rank of $\mathbf{Z}$, $t_{FMM,MV}(s)$ is the time needed to do one matrix vector multiplication of FMM $\mathcal{H}^2$-matrix, $t_{\text{New},MV}(s)$ is the time needed to do one matrix vector multiplication of New $\mathcal{H}^2$-matrix. $mem_{FMM}(Mb)$ is the memory needed to store FMM H2-matrix, $mem_{\text{New}}(Mb)$ is the memory need to store new $\mathcal{H}^2$-matrix, mem_{max} is the maximum memory needed to convert an FMM $\mathcal{H}^2$-matrix ot a New $\mathcal{H}^2$-matrix.

4.6 Conclusion

We present new algorithms to generate a rank-minimized $\mathcal{H}^2$-matrix to represent electrically large surface IE operators. First, the FMM is leveraged to obtain an initial $\mathcal{H}^2$-matrix in low complexity. Fast nested reduction algorithms are then developed to convert the FMM-based $\mathcal{H}^2$-representation to a new $\mathcal{H}^2$-matrix whose rank is minimized based on accuracy. Then we propose the Double Recursive Algorithm and NRA using sparsity to further reduce the complexity of converting process. The resultant new $\mathcal{H}^2$-matrix is found to have a

Table 4.7. Data of NRA using sparsity, hard-thresholding $(\tilde{\mathbf{U}}^t)^H$ based on entire matrix's max for every cluster, using SVD on $\tilde{\mathbf{G}}\tilde{\mathbf{G}}^H$, Sphere, $\epsilon = 10^{-2}$

N	4608	18432	73728	294912	1179648
e_{MV}	7.529e-3	1.150e-2	1.783e-2	2.415e-2	3.331e-2
t_{conv}	475.90	3997.77	19918.75	93580.29	686575.04
$t_{conv,\phi}$	127.34	1108.64	5081.38	23848.25	161832.38
$r_{V,\phi}$	18	43	100	273	818
$r_{V,S}$	12	27	73	224	1194
$t_{FMM,MV}(s)$	3.76	21.74	68.62	189.03	685.95
$t_{New,MV}(s)$	0.54	3.33	5.21	39.23	122.92
$mem_{FMM}(Mb)$	278.36	1429.50	6633.92	27890.01	116086.91
$mem_{New}(Mb)$	176.10	772.82	3342.13	14887.16	75322.47
mem_{max}	384.53	1839.41	8060.90	37521.27	190776.79

Table 4.8. Data of NRA using sparsity, hard-thresholding $(\tilde{\mathbf{U}}^t)^H$ based on entire matrix's max for every cluster, and hard-thresholding $(\tilde{\mathbf{U}}_1^t)^H T_1$ and $(\tilde{\mathbf{U}}_2^t)^H T_2$, using SVD on $\tilde{\mathbf{G}}\tilde{\mathbf{G}}^H$ (Note $\tilde{\mathbf{G}} = \begin{bmatrix} \tilde{\mathbf{U}}_1^H \mathbf{T}_1 \\ \tilde{\mathbf{U}}_2^H \mathbf{T}_2 \end{bmatrix}$), Sphere, $\epsilon = 10^{-2}$

N	4608	18432	73728	294912	1179648
e_{MV}	7.479e-3	1.167e-2	1.821e-2	2.511e-2	3.504e-2
t_{conv}	595.86	4014.23	25439.88	135229.79	797184.03
$t_{conv,\phi}$	162.97	1133.15	7060.59	36015.48	196844.77
$r_{V,\phi}$	18	43	103	278	813
$r_{V,S}$	12	27	74	245	1272
$t_{FMM,MV}(s)$	3.61	21.33	73.87	197.99	677.45
$t_{New,MV}(s)$	0.79	2.98	3.71	38.87	123.97
$mem_{FMM}(Mb)$	278.36	1429.50	6633.92	27890.01	116086.91
$mem_{New}(Mb)$	176.12	772.44	3337.91	14896.65	75909.89
mem_{max}	384.42	1842.91	8062.64	37618.62	191730.48

Table 4.9. Data of NRA using sparsity, without thresholding $(\tilde{\mathbf{U}}^t)^H$, using SVD on $\tilde{\mathbf{G}}\tilde{\mathbf{G}}^H$, Sphere, $\epsilon = 10^{-2}$

N	4608	18432	73728	294912	1179648
e_{MV}	7.443e-3	1.075e-2	1.660e-2	2.059e-2	2.322e-2
t_{conv}	79.83	624.33	4508.70	27730.21	216170.56
$t_{conv,\phi}$	19.07	131.79	805.71	5310.30	42569.79
$r_{V,\phi}$	18	45	102	270	816
$r_{V,S}$	12	27	71	199	578
$t_{FMM,MV}(s)$	3.62	20.89	70.78	195.51	685.08
$t_{New,MV}(s)$	0.63	2.67	2.51	31.32	151.85
$mem_{FMM}(Mb)$	278.36	1429.50	6633.92	27890.01	116086.91
$mem_{New}(Mb)$	175.79	773.42	3354.14	14872.06	70623.54
mem_{max}	383.31	2012.29	8107.30	37939.30	197145.83

much reduced rank without sacrificing prescribed accuracy, which accelerates both iterative and direct solutions. The proposed work has been applied to solve electrically large SIE equations for scattering analysis. Its accuracy and efficiency are demonstrated by numerical experiments. In addition to surface IEs, it is also applicable to volume IEs, and other IE operators.

5. NESTED CONSTRUCTION METHOD

In chapter 4.2, the MLFMA is leveraged to accelerate the generation of $\mathcal{H}^2$-matrices for SIEs. The method can achieve a time complexity of $O(N^{1.5}\log N)$. However, it is kernel-dependent, involving the implementation of MLFMA.

In this chapter, we develop a kernel-independent and purely algebraic method, Nested Construction Method, which can construct a rank-minimized $\mathcal{H}^2$-matrix to represent electrically large surface IE operators with low complexity based on prescribed accuracy. In this method, for each cluster in the $\mathcal{H}^2$-tree, we consider the interaction between the cluster (row cluster) and other admissible clusters (column clusters) at the same tree level, as well as all ancestor levels. To find the low-rank representation of such an interaction efficiently, we employ Pseudo-Skeleton Approximation [16], [17] (PSA), and also randomly choose $O(k)$ rows from the row cluster, and $O(k)$ columns from the column clusters to build a low-rank representation. The time cost of this method in generating each cluster basis and coupling matrix is of $O(kn\log n)$, while the memory consumption scales as $O(k^2)$, where k is the rank of the cluster basis, and n is cluster size. At each non-leaf level, we project the $O(k)$ columns selected for a non-leaf cluster to the cluster bases of its two children, and use the nested relationship to find transfer matrices efficiently. For electrically large surface integral operators, taking into account the rank's growth at each tree level, the time complexity of the proposed algorithm scales as $O(N^{1.5}\log N)$, and the memory complexity is $O(N\log N)$ for generating a rank-minimized $\mathcal{H}^2$-representation. This complexity is the same as in chapter 4.2. But this algorithm is kernel-independent. Since the new method does not depend on MLFMA and is purely algebraic, it is also easier to implement.

In addition to an efficient $\mathcal{H}^2$-matrix construction, the underlying algorithms of this work can be used to efficiently construct an $\mathcal{H}$-matrix representation of IE operators, and be used to perform efficient $\mathcal{H}$- to $\mathcal{H}^2$-matrix conversion with a reduced complexity than existing methods for performing the same task. We have introduced $\mathcal{H}^2$-matrix, now we briefly introduce the $\mathcal{H}$-matrix [6]. The $\mathcal{H}$-matrix is similar to the $\mathcal{H}^2$-matrix. It also constitutes a hierarchical and often low-rank representation of the original dense matrix, the structure of which can be utilized to accelerate matrix computation. $\mathcal{H}$-matrix structure is also built using a row binary tree and a column one, an example of which is illustrated in Fig. 1.1. In an $\mathcal{H}$-matrix, by checking the admissibility condition level by level between a row cluster tree and a column cluster tree, the original matrix is partitioned into multilevel admissible and inadmissible blocks. Physically, an admissible block represents the interaction between separated sources (column cluster) and observers (row cluster). An inadmissible block is stored in its original full matrix format, while an admissible block has compact storage. Take an admissible block (t, s) formed between a row cluster t and a column cluster s as an example. In an $\mathcal{H}$-matrix, the admissible block is represented as $\mathbf{AB}^T$, whose rank is k which can be smaller than the row and column dimension of the block. Actually, the $\mathcal{H}^2$-matrix is a special class of the $\mathcal{H}$-matrix. The difference lays in that the latter does not have a nested hierarchical representation. Both are more compact and yield lower computational costs. Thus, the $\mathcal{H}^2$-matrix is more difficult to be generated.

Numerical experiments have demonstrated the accuracy, efficiency, and complexity of the proposed method. In addition to surface integral equations, the proposed algorithms can also be applied to solving other electrically large integral equations.

The rest of this chapter is organized as follows. In Section I, we review the background of this paper. In Section II, we present the proposed Nested Construction Algorithm for $\mathcal{H}^2$-construction and analyze its computational complexity. In Section III, we show how to use the proposed algorithm to convert an $\mathcal{H}$-matrix to an $\mathcal{H}^2$-matrix efficiently. In Section IV, a number of numerical results are presented to validate the accuracy and computational complexity of the proposed algorithms. Section VI relates to our conclusions.

5.1 Background

5.1.1 Pseudo-Skeleton Approximation

In [16], [17], it is shown that there exists a Pseudo-Skeleton Approximation (PSA) of a low-rank matrix with prescribed accuracy, which uses selected rows and columns of the original matrix to generate a low-rank matrix. Let the set of selected rows be r and the set of selected columns be c, for a low-rank matrix $\mathbf{M}$, a PSA takes the following form

$$\mathbf{M} \approx \mathbf{C}\mathbf{M}(r, c)^{\dagger}\mathbf{R}, \tag{5.1}$$

where $\mathbf{C}$ contains selected columns of $\mathbf{M}$ whose indexes belong to c, thus being

$$\mathbf{C} = \mathbf{M}(:, c), \tag{5.2}$$

$\mathbf{R}$ contains selected rows of $\mathbf{M}$ whose indexes belong to r, thus being

$$\mathbf{R} = \mathbf{M}(r, :), \tag{5.3}$$

$\mathbf{M}(r, c)$ is the intersection between the rows in r and columns in c of $\mathbf{M}$, having $|r| \times |c|$ entries, and $\mathbf{M}(r, c)^{\dagger}$ denotes the pseudo-inverse of $\mathbf{M}(r, c)$.

In this work, we compute the pseudo-inverse of $\mathbf{M}(r, c)$ using reduced SVD (rSVD), obtaining $\mathbf{M}(r, c) = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$, where $\mathbf{\Sigma}$ is truncated based on prescribed accuracy ϵ. Then, we obtain

$$\mathbf{M}(r, c)^{\dagger} = \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^H. \tag{5.4}$$

Such an implementation eliminates the instability of the pseudo-inverse even if $\mathbf{M}(r, c)$ is rank-deficient. It also reveals the rank of $\mathbf{M}$ at the same time.

5.2 Nested Construction Algorithm

To obtain a low-rank representation of an IE dense matrix algebraically, a brute-force method would cost $O(N^3)$ since it is equivalent to performing an SVD of the matrix and truncating it based on prescribed accuracy. In this section, we present a fast Nested Construction (NC) algorithm to generate such a low-rank representation, and also make it nested, thus producing an $\mathcal{H}^2$-matrix. **Algorithms** 18, 19, and 20 provide an overview of this algorithm. As can be seen, the algorithm consists of one bottom-up traversal of the cluster tree to construct cluster bases (**Algorithm** 19), and one traversal of block cluster tree to construct coupling matrices (**Algorithm** 20). Next, we will present how the nested cluster bases are generated at the leaf level, and non-leaf levels respectively, and then explain the coupling matrix generation.

Algorithm 18 Nested_Construction_Algorithm

```
1: procedure nested_construction_algorithm($t, b$)
2:     construct_cluster_basis($t$)
3:     construct_coupling_matrix($b$)
4: end procedure
```

Algorithm 19 construct_cluster_basis

```
1: procedure construct_cluster_basis($t$)
2:     if $t$ is a non-leaf cluster then
3:         for i is $t$'s child do
4:             construct_cluster_basis($t_\mathrm{i}$)
5:         end for
6:         fill_non_leaf_cluster($t$)
7:     else
8:         fill_leaf_cluster($t$)
9:     end if
10: end procedure
```

Algorithm 20 construct_coupling_matrix

```
1: procedure construct_coupling_matrix($b$)
2:     if $b$ is an admissible block then
3:         fill_coupling_matrix($b$)
4:     else if i is $b$'s child then
5:         construct_coupling_matrix($b^\mathrm{i}$)
6:     else if $b$ is an inadmissble block then
7:         fill dense original matrix for $b$
8:     end if
9: end procedure
```

5.2.1 Cluster Basis Generation at Leaf Level

For an arbitrary leaf cluster t, instead of only considering admissible blocks formed by t at t's level, we use the interaction between t and its far-field t^+ to generate the cluster basis.

The same is performed for non-leaf clusters. This is to facilitate the generation of nested cluster bases, which will soon become clear in the next section.

The far field t^+ of a cluster t is defined as

$$t^+ = \{s : \text{ s. t. } (t_p, s) \text{ is admissible}, \exists t_p \in [\mathcal{P}(t), t]\} \tag{5.5}$$

where $\mathcal{P}(t)$ denotes the set of ancestors of cluster t. Hence, t^+ includes all clusters s that are admissible with t at t's level, as well as those that are admissible with t's ancestors. Define $\mathbf{G}_t$ as the matrix formed by t and t^+. An example of $\mathbf{G}_t$ is shown in Fig. 5.1 for cluster $t = t_4$, which is highlighted in yellow.

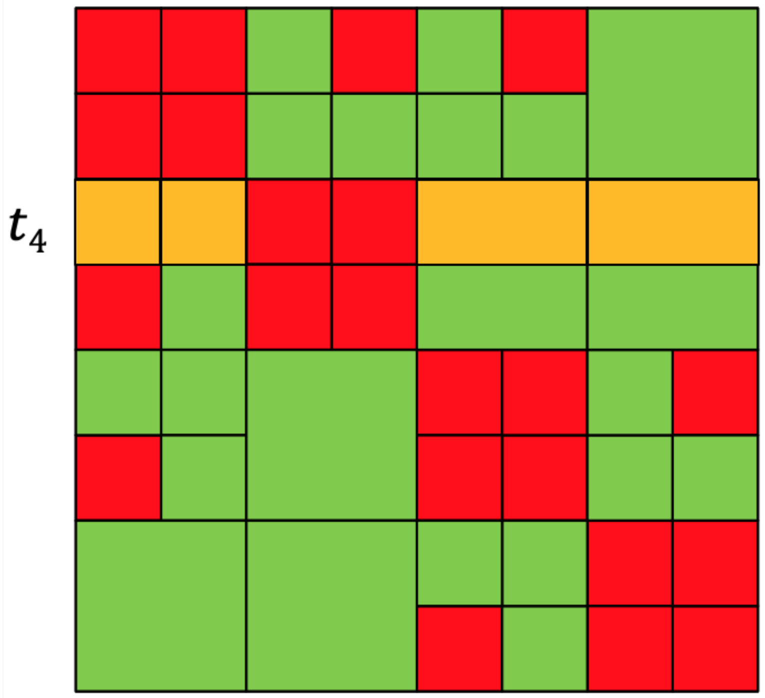


Figure 5.1. Submatrices colored in yellow are $\mathbf{G}_{t_4}$ of cluster t_4.

Now consider a leaf cluster t, to generate its cluster basis, we select $O(k)$ rows from t, and $O(k)$ columns from t^+, where k is the rank of $\mathbf{G}_t$ for a prescribed accuracy, from which we obtain a PSA of $\mathbf{G}_t$ as

$$\mathbf{G}_t \approx \mathbf{C}_t \mathbf{G}_t(r, c)^\dagger \mathbf{R}_t, \tag{5.6}$$

where $\mathbf{G}_t(r,c)^\dagger$ denotes the pseudo-inverse of the intersection between the selected rows r and columns c, $\mathbf{C}_t = \mathbf{G}_t(:,c)$, and $\mathbf{R}_t = \mathbf{G}_t(r,:)$. Since the goal here is to generate the cluster basis of t, which falls into the column space of $\mathbf{C}_t$, we only need to compute $\mathbf{C}_t$. For efficiency, we randomly select c columns from $\mathbf{G}_t$, and choose $|c| = c_0 k_t$, where c_0 is a constant coefficient larger than 1. The rank of cluster t, k_t, can be estimated as $leafsize$ at the leaf level. At a nonleaf level, k_t can be estimated from the sum of the two children clusters' rank or from the square root of the cluster size based on [9]. The actual rank will be determined after the rSVD is performed on the selected columns as follows.

Performing an rSVD on $\mathbf{C}_t$, we obtain

$$\mathbf{C}_t = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H. \tag{5.7}$$

Truncating the above based on prescribed accuracy ϵ, we obtain the cluster basis of the leaf cluster t as

$$\mathbf{V}^t \approx \mathbf{U}_{k_t}, \tag{5.8}$$

in which $\mathbf{U}_{k_t}$ denotes the k_t left singular vectors corresponding to the largest k_t singular values satisfying prescribed accuracy. The pseudo-code of the aforementioned leaf-level computation is shown in **Algorithm** 21.

Algorithm 21 fill_leaf_cluster

```
1: procedure fill_leaf_cluster($t$)
2:     Generate $\mathbf{C}_t$ from randomly selected column pivots.
3:     Do rSVD on $\mathbf{C}_t$ to obtain leaf cluster basis $\mathbf{V}^t$.
4: end procedure
```

5.2.2 Cluster Basis Generation at Nonleaf Levels

For a non-leaf cluster t, to build a nested cluster basis, we do not do rSVD on $\mathbf{C}_t$ directly since it would be expensive. Instead, we do the rSVD on the projection of $\mathbf{C}_t$ onto its two children cluster bases. Since the children cluster bases are built to account for the interaction with all ancestor-level admissible clusters, i.e., with t^+, such a projection can be accurately performed. In other words, the children cluster bases can accurately represent parent cluster's $\mathbf{G}_t$, and thus its selected columns $\mathbf{C}_t$ as well, since children cluster bases are generated with their parent's $\mathbf{C}_t$ taken into account.

Let the projection of $\mathbf{C}_t$ onto children cluster bases be $\mathbf{C}_t^{proj}$, which is

$$\mathbf{C}_t^{proj} = \begin{bmatrix} (\mathbf{V}^{t_1})^H & \\ & (\mathbf{V}^{t_2})^H \end{bmatrix} \mathbf{C}_t, \tag{5.9}$$

where t_1 and t_2 denote t's two children, $\mathbf{V}^{t_1}$ and $\mathbf{V}^{t_2}$ are corresponding cluster bases that have been generated at previous level. The $\mathbf{C}_t$ in (5.9) has $O(k_t)$ columns, and its row dimension is the size of cluster t, and hence being n_l, the number of unknowns at tree level l. Even though generating $\mathbf{C}_t$ costs $O(k_t n_l)$ operations only, a brute-force multiplication of $\mathbf{C}_t$ with $(\mathbf{V}^{t_1})^H$ and $(\mathbf{V}^{t_2})^H$ would cost $O(k_t^2 n_l)$, which is expensive. Hence, we need to speed up the multiplication in (5.9). To do so, we propose to use the nested property of cluster bases $\mathbf{V}$ level by level. The technique used here is essential the same as a part described in section 4.2. Here we state again.

For each column vector in $\mathbf{C}_t$, we can obtain its product with $(\mathbf{V}^{t_1})^H$ and $(\mathbf{V}^{t_2})^H$ in $O(n_l \log n_l)$ operations, using the nested property of cluster bases $\mathbf{V}$. Basically, for cluster t, we go down the cluster tree until we reach t's leaf level clusters. Starting from this level (leaf level $l = L$), we compute $(\mathbf{V}^{t_\mathrm{i}})^H$ multiplying corresponding segments of a column in $\mathbf{C}_t$, where t_i is the i-th cluster at leaf level, and we do so for all leaf clusters of t. The total cost at leaf level is $O(n_l)$. We then move one level up to $l = L - 1$, for each cluster t_i at this level, we compute $\mathbf{T}^{t_\mathrm{i}H}$ multiplying the union of t_i's two children's vectors obtained at previous level, the cost of which is $O(k_{t_\mathrm{i}}^2)$. We do so level by level up for each child cluster of t. When we reach nonleaf cluster t at level l, we obtain a matrix of size $(k_{t_1} + k_{t_2}) \times k_t$, which is the result of (5.9).

Denoting each column of $\mathbf{C}_t$ by $\boldsymbol{v}$, the pseudo-code of the aforementioned $O(n_l \log n_l)$ algorithm is shown in **Algorithm** 22. In this pseudo-code, the input is $\boldsymbol{v}$ and its associated cluster. The $\boldsymbol{\omega}^t$ is a vector belonging to cluster t and of size k_t (rank of cluster t); and $\boldsymbol{v}^t$ is the segment of $\boldsymbol{v}$ that corresponds to cluster t. After **Algorithm** 22 is executed, the $\boldsymbol{\omega}^t$

of the input cluster t is the result being pursued, i.e, one column of $\mathbf{C}_t^{proj}$. **Algorithm** 22 allows us to reduce the time of one matrix-vector multiplication in (5.9) from $O(n_l k_t)$ to $O(n_l \log n_l)$, by taking advantage of nested property. Here, the former is larger than the latter because the rank k_t scales as $\sqrt{n_l}$ in an electrically large surface IE [9], and hence can be much larger than $\log n_l$. Since the column dimension of $\mathbf{C}_t^{proj}$ is $O(k_t)$, the total time cost of computing $\mathbf{C}_t^{proj}$ is $O(k_t n_l \log n_l)$.

Algorithm 22 recursively_multi_new_trans

This algorithm performs a bottom-up tree traversal to compute $\mathbf{V}^H \boldsymbol{v}$

```
1: procedure recursively_multi_new_trans($t, \boldsymbol{v}$)
2:     if $t$ is a non-leaf cluster then
3:         for i is $t$'s child do
4:             recursively_multi_new_trans($t_\mathrm{i}, \boldsymbol{v}$)
5:         end for
6:         $\boldsymbol{\omega}^t \leftarrow \begin{bmatrix} (\mathbf{T}^{t_1})^H \boldsymbol{\omega}^{t_1} \\ (\mathbf{T}^{t_2})^H \boldsymbol{\omega}^{t_2} \end{bmatrix}$
7:     else
8:         $\boldsymbol{\omega}^t \leftarrow (\mathbf{V}^t)^H \boldsymbol{v}^t$
9:     end if
10: end procedure
```

After obtaining $\mathbf{C}_t^{proj}$, we apply an rSVD with a prescribed accuracy ϵ, the resultant left singular vectors are the transfer matrix of cluster t,

$$\mathbf{T}^t = \begin{bmatrix} \mathbf{T}^{t_1} \\ \mathbf{T}^{t_2} \end{bmatrix}. \tag{5.10}$$

The entire procedure for generating the cluster basis of a nonleaf cluster is shown by the fill_non_leaf_cluster(t) in the following.

5.2.3 Coupling Matrix Generation

For each coupling matrix, denote the corresponding admissible block by $\mathbf{G}_{t,s}$, where t is a row cluster, and s is a column one admissible to t. We first randomly choose r rows and c columns from $\mathbf{G}_{t,s}$. Similar to how cluster bases are generated, we choose $|r| = c_0 k_s$ and $|c| = c_0 k_t$, where $c_0 \geq 1$ is the enlarging constant coefficient, and k_t and k_s are the rank of

Algorithm 23 fill_non_leaf_cluster

```
1: procedure fill_non_leaf_cluster($t$)
2:     generate $c_t$ (column pivots of cluster $t$).
3:     for $\boldsymbol{v}$ is a column corresponding to a pivot of $c_t$ do
4:         recursively_multi_new_trans($t$, $\boldsymbol{v}$) $\rightarrow \omega$
5:         store $\omega$ as one column of $\mathbf{C}_t^{proj}$
6:     end for
7:     Do rSVD on $\mathbf{C}_t^{proj}$ to get transfer matrix $\mathbf{T}^t$.
8: end procedure
```

corresponding row, and column cluster basis, respectively. Then we form three submatrices $\mathbf{C}_{t,s}$, $\tilde{\mathbf{G}}_{t,s}$ and $\mathbf{R}_{t,s}$ based on t, s, r and c to construct a PSA, hence obtaining

$$\mathbf{G}_{t,s} = \mathbf{C}_{t,s}\tilde{\mathbf{G}}^{\dagger}_{t,s}\mathbf{R}_{t,s}, \tag{5.11}$$

in which

$$\mathbf{C}_{t,s} = \mathbf{G}_{t,s}(:, c),\ \tilde{\mathbf{G}}_{t,s} = \mathbf{G}_{t,s}(r, c),\ \mathbf{R}_{t,s} = \mathbf{G}_{t,s}(r, :). \tag{5.12}$$

Multiplying the $\mathbf{C}_{t,s}$ by $(\mathbf{V}^t)^H$ in front yields $\tilde{\mathbf{C}}$, which is the projection of $\mathbf{C}_{t,s}$ onto the cluster basis of t. Again, this multiplication can be efficiently carried out using the nested algorithm shown in **Algorithm** 22. Similarly, multiplying the $\mathbf{R}_{t,s}$ by $(\mathbf{V}^s)$, we obtain $\tilde{\mathbf{R}}$. As a result, the coupling matrix of the admissible block can be found as

$$\mathbf{S}^{t,s} = \tilde{\mathbf{C}}\tilde{\mathbf{G}}^{\dagger}_{t,s}\tilde{\mathbf{R}}, \tag{5.13}$$

where $\tilde{\mathbf{G}}^{\dagger}_{t,s}$ is the pseudo-inverse of $\tilde{\mathbf{G}}_{t,s}$.

We formalize the aforementioned in the fill_coupling_matrix(t) pseudo-code shown in **Algorithm** 24.

Algorithm 24 fill_coupling_matrix

```
1: procedure fill_coupling_matrix($b$)
2:     generate $\mathbf{C}$, $\tilde{\mathbf{G}}$ and $\mathbf{R}$.
3:     do $\tilde{\mathbf{C}} = (\mathbf{V}^t)^H\mathbf{C}$.
4:     do $\tilde{\mathbf{R}} = \mathbf{R}(\mathbf{V}^s)$.
5:     compute $\mathbf{S}^{t,s} = \tilde{\mathbf{C}}\tilde{\mathbf{G}}^{\dagger}\tilde{\mathbf{R}}$
6: end procedure
```

5.2.4 Complexity Analysis

The cost of generating leaf cluster bases is $O(N)$. For a non-leaf cluster, we have analyzed that the time complexity of getting $\mathbf{C}_t$ is of $O(k_t n \log n)$. Doing rSVD on $\mathbf{C}_t^{proj}$ is $O(k_t^3)$ since $\mathbf{C}_t^{proj}$ is of size $O(k_t) \times O(k_t)$. As for the cost of coupling matrix generation, computing $\tilde{\mathbf{G}}^{\dagger}_{t,s}$

in (5.13) has $O(k^3)$ complexity, where $k = \max(k_t, k_s)$. The cost of computing $\tilde{\mathbf{C}}$, $\tilde{\mathbf{R}}$ and $\tilde{\mathbf{S}}^{t,s}$ are $O(kn \log n)$, $O(kn \log n)$, and $O(k^3)$, respectively. Since in an SIE, rank scales as the square-root of cluster size at each tree level, the generation of coupling matrix is of $O(k_t^3 \log n)$ cost. Hence, the total time complexity of the proposed Nested Construction algorithm is

$$\begin{aligned} C_t &= \sum_{l=0}^{L} 2^l O(k_t^3 \log n) \\ &= \sum_{l=0}^{L} 2^l O\left(\left(\sqrt{\frac{N}{2^l}} \right)^3 \log\left(\frac{N}{2^l}\right) \right) = O(N^{1.5} \log N). \end{aligned} \tag{5.14}$$

Memory-wise, for cluster basis generation, the dimension of $\mathbf{C}_t$ is $O(n) \times O(k_t)$, but we can avoid storing it as a whole. We can generate one column of $\mathbf{C}_t$, multiply this column with V^{tH} to get one column of $\mathbf{C}_t^{proj}$. Then we use the same memory space to generate another column of $\mathbf{C}_t$ and thereby compute another column of $\mathbf{C}_t^{proj}$. In this way, we only need a space sufficient to store one column of $\mathbf{C}_t$, which costs $O(n)$ only, and a space for storing $\mathbf{C}_t^{proj}$, which is $O(k_t^2)$. The transfer matrix $\mathbf{T}$ also only requires storage of $O(k_t^2)$. Hence, for each non-leaf cluster, the memory requirement is $O(k_t^2)$ for generating and storing the cluster basis.

For coupling matrix, we can use a similar technique to avoid the direct storage of $\mathbf{C}_{t,s}$ and $\mathbf{R}_{t,s}$. Thus, the memory requirement for generating each coupling matrix can also be made $O(k_t^2)$. As a result, the total memory complexity of the proposed algorithm can be calculated as

$$\begin{aligned} C_m &= \sum_{l=0}^{L} (2^l O(k_t^2) + C_{sp} 2^l O(k_t^2)) \\ &= \sum_{l=0}^{L} 2^l O\left(\frac{N}{2^l}\right) = O(N \log N). \end{aligned} \tag{5.15}$$

5.3 Efficient Conversion of $\mathcal{H}$-matrix to $\mathcal{H}^2$-matrix

The proposed new method can also be used as an efficient way to convert an $\mathcal{H}$-matrix to an $\mathcal{H}^2$-matrix, which has a much reduced complexity as compared to using the $\mathcal{H}$ to $\mathcal{H}^2$ conversion in [6] for solving electrically large problems. Starting from an $\mathcal{H}$-matrix $\mathbf{M}$, next, we show how to convert it to a minimal-rank $\mathcal{H}^2$-matrix. This $\mathcal{H}$-matrix can be generated using PSA as shown in this work. It can also be generated using other approaches. Let the admissible block formed by cluster t and cluster s be $\mathbf{M}^{t,s} = \mathbf{A}^{t,s}(\mathbf{B}^{t,s})^H$ in the $\mathcal{H}$-matrix. We first construct nested cluster bases as follows.

1. For each leaf cluster t, form a new matrix $\mathbf{M}_t = \begin{bmatrix}\mathbf{A}^{t,s_1} & \mathbf{A}^{t,s_2} & ... & \mathbf{A}^{t,s_n}\end{bmatrix}$, for $s_{\text{i}} \in \hat{t}^+$. Here, for s_{i} that is admissible with an arbitrary ancestor of t, t_a, $\mathbf{A}^{t,s_{\text{i}}}$ denotes the submatrix of $\mathbf{A}^{t_a,s_{\text{i}}}$ corresponding to cluster t. Perform an SVD on $\mathbf{M}_t$ based on accuracy ϵ to get the orthogonalized cluster basis of t, $\mathbf{V}^t$.

2. For each non-leaf cluster t, we also form a matrix $\mathbf{M}_t = \begin{bmatrix}\mathbf{A}^{t,s_1} & \mathbf{A}^{t,s_2} & ... & \mathbf{A}^{t,s_n}\end{bmatrix}$, where $s_{\text{i}} \in \hat{t}^+$, similar to what is done at the leaf level. We then multiply $\mathbf{M}_t$ by $\begin{bmatrix}(\mathbf{V}^{t_1})^H & \\ & (\mathbf{V}^{t_2})^H\end{bmatrix}$, i.e., find the projection of the $\mathbf{M}_t$ onto its two children cluster bases. This multiplication is done efficiently using the nested property of $\mathbf{V}$ as shown in **Algorithm** 22. Let the resultant matrix be $\mathbf{M}_t^{proj}$. Do SVD on $\mathbf{M}_t^{proj}$ based on accuracy ϵ to obtain transfer matrix $\mathbf{T}^t$.

After cluster bases are generated using the aforementioned scheme, we compute coupling matrices as follows. For each admissible block of (t,s), we compute $\mathbf{M}_r = (\mathbf{V}^t)^H\mathbf{A}^{t,s}$, and $\mathbf{M}_c = (\mathbf{V}^s)^H\mathbf{B}^{t,s}$. Then the $\tilde{\mathbf{S}} = \mathbf{M}_r(\mathbf{M}_c)^H$ is the coupling matrix of the new $\mathcal{H}^2$-matrix.

We next analyze the complexity of the proposed algorithm for converting an $\mathcal{H}$-matrix to an $\mathcal{H}^2$-matrix. For each leaf cluster t, the largest column dimension of $\mathbf{A}^{t,s_{\text{i}}}$ is of $O(\sqrt{N})$, where s_{i} is the column cluster of the largest admissible block formed with t's ancestor. This is because the size of the largest admissible block is of $O(N)$, and the rank scales as square root of it. The next largest column dimension of $\mathbf{A}^{t,s_{\text{i}}}$ is $O(\sqrt{N/2})$. Similarly, the next is $O(\sqrt{N/4})$, etc. The number of $\mathbf{A}^{t,s_{\text{i}}}$ blocks at each level is bounded by C_{sp}. Hence, the total column dimension of $\mathbf{M}^t$ at the leaf level is bounded by $C_{sp}O(\sqrt{N})(1+1/\sqrt{2}+1/2+...) = O(\sqrt{N})$. Hence, doing SVD on $\mathbf{M}_t$ costs $O(\sqrt{N})$ since the row dimension of $\mathbf{M}_t$ is $leafsize$ which is a constant. There are $O(N)$ leaf clusters, thus, the total time complexity for computing leaf cluster bases is

$$C_{t,l} = O(N) \times O(\sqrt{N}) = O(N^{1.5}). \tag{5.16}$$

Similarly, for a non-leaf cluster t, the column dimension of $\mathbf{M}_t$ is bounded by $O(\sqrt{N})$. Multiplying each column of $\mathbf{M}_t$ with $\mathbf{V}^H$ takes $O(n\log n)$ operations, where n is t's cluster size. Thus, the complexity of computing $\mathbf{M}_t^{proj}$ is $O(\sqrt{N}n\log n)$. The size of $\mathbf{M}_t^{proj}$ is $O(k_t)\times O(\sqrt{N})$. Doing rSVD on it takes $O((k_t)^2\sqrt{N}) \approx O(n\sqrt{N})$. Thus, the time complexity of computing all nonleaf cluster bases is

$$C_{t,nl} = \sum_{l=0}^{L-1} 2^l O(\sqrt{N}\; n\log n) = O(N^{1.5}\log^2 N). \tag{5.17}$$

For coupling matrix, using the nested property, the computation of $\mathbf{M}_r$ and $\mathbf{M}_c$ is of complexity $O(k_t n\log n)$. The computation of $\tilde{\mathbf{S}}$ is of complexity $O(k_t^3)$. Thus, the total complexity of computing coupling matrices is

$$C_{t,coupling} = \sum_{l=0}^{L} 2^l C_{sp} O(k_t n\log n) = O(N^{1.5}\log N). \tag{5.18}$$

Adding (5.16), (5.17), and (5.18), the total time complexity of converting an $\mathcal{H}$-matrix to an $\mathcal{H}^2$-matrix is $O(N^{1.5}\log^2 N)$. For comparison, using the conversion method in [6], the total complexity of doing the same is $O(N^2)$. Hence, the proposed algorithm has a reduced complexity. In the complexity analysis above, we utilize the rank at each tree level for SIE scaling as $O(n^{0.5})$ for a cluster of size n, based on [9]. For volume IEs and $\mathcal{H}$-matrices of other ranks, the resultant complexity would be different but can be analyzed similarly using the complexity analysis given here. It is also worth mentioning that one can employ the random approach used in the previous section, and only choose $O(k_t)$ columns to construct $\mathbf{M}_t$ for each cluster. If doing so, the complexity of the above $\mathcal{H}$-matrix to $\mathcal{H}^2$-matrix conversion can be further reduced to $O(N^{1.5}\log N)$ in time, and $O(N\log N)$ in memory.

5.4 Numerical Results

In this section, we conduct a number of numerical experiments to validate the accuracy and computational complexity of the proposed algorithm. A common set of simulation parameters are used for all examples simulated in this section. Specifically, we choose $\eta = 1.2$ in the admissibility condition, $leafsize$ to be 40 and excitation frequency to be $f = 300$ MHz. The accuracy parameter ϵ in the nested construction algorithm is chosen based on the desired accuracy. The enlarging coefficient c_0 for the random choice of pivots is chosen to be 4. In the case that the direct solver of [7] is used to solve the $\mathcal{H}^2$-matrix generated from the proposed algorithm, the accuracy of the direct solver is set to be 10^{-3}, unless stated otherwise.

5.4.1 Pseudo-Skeleton Approximation with Random Choice of Pivots

We first validate the accuracy of PSA constructed by randomly choosing $O(k)$ rows and columns, in the context of $\mathcal{H}^2$-matrix and EFIE. We use two cubes to set up this test. The distance between the two cubes is chosen to be $D = \eta\ \max(d_1, d_2)$, where d_1 is the diameter of the first cube, d_2 is that of the second cube, and $\eta = 1.2$. We have studied a variety of configurations with $d_1 = d_2$ or $d_1 \neq d_2$, and with different electrical sizes, as shown in Table 5.1. The n_1 in the Table denotes the number of degrees of freedom of the first cube, while n_2 is that of the second cube. The MoM matrix generated for the interaction between these two cubes, $\mathbf{M}$, is of size $n_1 \times n_2$. In this matrix, we randomly choose a row set of size $|r| = c_0\sqrt{n_1}$, a column set of size $|c| = c_0\sqrt{n_2}$, where the enlarging coefficient c_0 is 4. Then we perform a PSA with $\epsilon = 10^{-3}$ and obtain an approximation as shown in (5.1). We assess the error of the approximation with the original MoM matrix as the following

$$E_1 = \frac{\|\mathbf{M} - \mathbf{C}\mathbf{M}(r,c)^{\dagger}\mathbf{R}\|_F}{\|\mathbf{M}\|_F}, \tag{5.19}$$

in which subscript F denotes a Frobenius norm. This error is shown by E_1 in Table 5.1. As can be seen, good accuracy is obtained for all configurations, validating the PSA with random choices implemented in this work. Theoretically speaking, this is mainly due to the nature of the MoM matrix whose column represents the fields generated by a source at various observation points via Green's function. The columns randomly selected across observation points have good linear independence.

Table 5.1. Accuracy of PSA with random choice pivots for the interaction between two cubes.

d_1	d_2	D	n_1	n_2	$\lvert r\rvert$	$\lvert c\rvert$	E_1
1.56	1.56	1.87	600	600	98	98	6.41e-04
3.29	3.29	3.94	2400	2400	196	196	7.04e-04
6.75	6.75	8.10	9600	9600	392	392	1.15e-03
13.68	13.68	16.41	38400	38400	784	784	1.90e-03
1.56	3.29	3.94	600	2400	98	196	6.88e-04
1.56	6.75	8.10	600	9600	98	392	4.54e-04
1.56	13.68	16.41	600	38400	98	784	0.0011
1.56	27.53	33.03	600	153600	98	1568	8.64e-04
3.29	13.68	16.41	600	38400	196	784	9.46e-04

We then use the same example to test the accuracy of representing an admissible block by using randomly selected $O(k)$ columns from the admissible block. Since PSA in the above test is accurate with a random choice of pivots, we expect this to be accurate. In this test, we randomly choose from $\mathbf{M}$ a column set of size $|c| = c_0\sqrt{n_1}$, where $c_0 = 4$. Let the resultant submatrix be denoted by $\mathbf{M}_r$. We then perform an rSVD on $\mathbf{M}_r$ to obtain left singular vectors $\mathbf{U}$ under $\epsilon = 10^{-3}$. The parameter *rank* shown in Table 5.2 is the rank obtained from the rSVD. We then assess the following error

$$E_2 = \frac{\|\mathbf{M} - \mathbf{U}\mathbf{U}^H\mathbf{M}\|_F}{\|\mathbf{M}\|_F}. \tag{5.20}$$

This error is listed in Table 5.2 for various configurations of the admissible block. Clearly, a good accuracy can be observed. Hence, the randomly selected columns contained in $\mathbf{M}_r$ can be used to accurately represent $\mathbf{M}$.

Table 5.2. Accuracy of using randomly selected columns to represent an admissible block.

d_1	d_2	D	n_1	n_2	$\|c\|$	rank	E_2
1.56	1.56	1.87	600	600	98	17	2.00e-03
1.56	3.29	3.94	600	2400	98	15	1.69e-03
1.56	6.75	8.10	600	9600	98	15	1.26e-03
1.56	13.68	16.41	600	38400	98	14	1.97e-3
1.56	27.53	33.03	600	153600	98	15	1.02e-3
1.56	55.23	66.27	600	614400	98	15	9.40e-4
3.29	3.29	3.95	2400	2400	196	30	1.41e-3
3.29	6.75	8.10	2400	9600	196	26	1.30e-3
3.29	13.68	16.41	2400	38400	196	26	9.92e-4
3.29	27.53	33.03	2400	153600	196	26	9.11e-4
3.29	55.23	66.27	2400	614400	196	26	9.83e-4
6.75	6.75	8.10	9600	9600	392	60	1.03e-3
6.75	13.68	16.41	9600	38400	392	54	1.18e-3
6.75	27.53	33.03	9600	153600	392	51	1.05e-3
13.68	13.68	16.41	38400	38400	784	135	9.96e-04

Next, we examine the effect of the enlarging constant c_0 on the accuracy of PSA. Results are shown in Table 5.3 for two different configurations of the admissible block. The ϵ of rSVD is chosen to be 10^{-10}. As can be seen, the accuracy of PSA is good for various choices of c_0, and it can be improved by enlarging c_0.

Table 5.3. The effect of c_0 on the accuracy of PSA.

c_0	d_1	d_2	D	n_1	n_2	$\|\|c\|\|$	rank	E_2
1	3.29	3.29	3.95	2400	2400	49	48	5.124e-04
2	3.29	3.29	3.95	2400	2400	98	96	2.737e-06
4	3.29	3.29	3.95	2400	2400	196	144	1.164e-08
8	3.29	3.29	3.95	2400	2400	392	161	7.271e-10
1	6.75	6.75	8.10	9600	9600	98	97	4.232e-04
2	6.75	6.75	8.10	9600	9600	196	192	4.259e-07
4	6.75	6.75	8.10	9600	9600	392	237	1.480e-09
8	6.75	6.75	8.10	9600	9600	784	249	5.098e-10

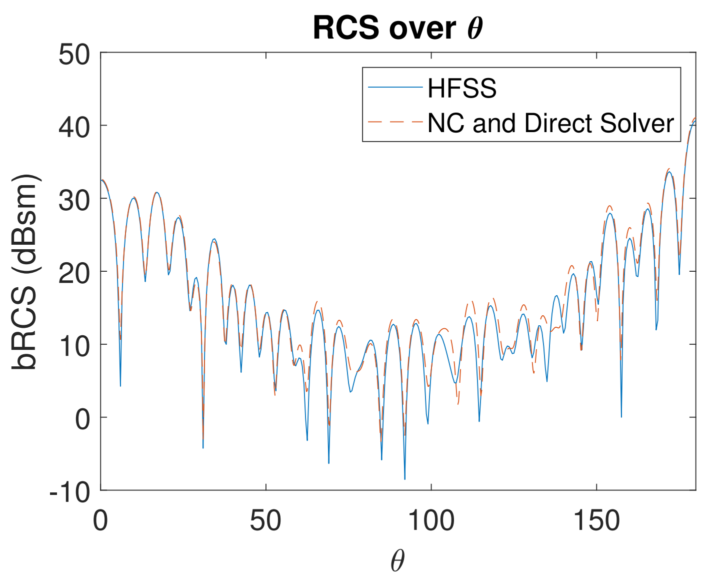


Figure 5.2. RCS of a coil simulated using Nested Construction.

5.4.2 Accuracy of the Proposed Nested Construction Algorithm

Next, we validate the accuracy of the proposed nested construction algorithm before examining its computational complexity in time and memory.

Scattering from an Irregularly Shaped Coil

We simulate the scattering from a coil, which is shown in Fig. 3.3. This coil has a diameter of 14.156 m and is illuminated by an incident plane wave at 300 MHz. After discretization, the number of unknowns is 120,058. The $\mathcal{H}^2$-matrix is first generated from the proposed NC algorithm and then solved using the fast direct solver of [7]. The resultant bistatic RCS is compared with HFSS's result in Fig. 5.2. As can be seen, the two agree well with each other.

Scattering from an Array of Spheres

In the second example, we simulate an array of spheres, having $6 \times 6 \times 6$ spheres, each of which has a diameter of 0.8288 m and is discretized with 648 unknowns at 300 MHz. The distance between two adjacent spheres is 2.0720 m. The total number of unknowns is 139,968. The bistatic RCS is computed using GMRES with Block Diagonal Preconditioner

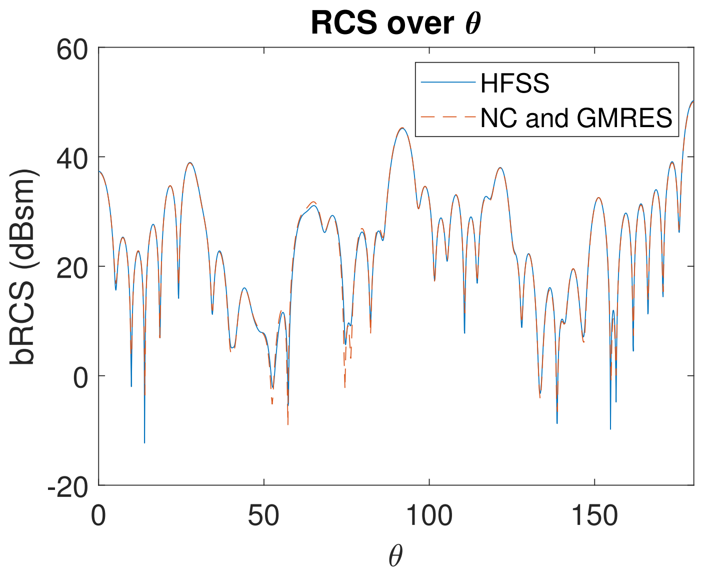


Figure 5.3. RCS of an array of spheres having $6 \times 6 \times 6$ elements using Nested Construction.

on the minimal-rank $\mathcal{H}^2$-matrix generated by the proposed NC algorithm. The RCS Result is compared with HFSS's result. Very good agreement is observed, as can be seen from Fig. 5.3. The accuracy criterion used is $\epsilon = 10^{-3}$.

Scattering from a Large Conducting Cube

In this example, we compute the bistatic RCS of a conducting cube of side length 21.324 m at 300 MHz, which has 1,179,648 unknowns. The new $\mathcal{H}^2$-matrix generated from the NC algorithm is solved by using the direct solver of [7]. The resultant bistatic RCS is compared with that from HFSS in Fig. 5.4, which shows very good agreement. In this example, the accuracy criterion used in the NC algorithm is set to be $\epsilon = 10^{-3}$.

5.4.3 Complexity of the Proposed Nested Construction Algorithm

With the accuracy validated, next, we examine the complexity of the proposed algorithm.

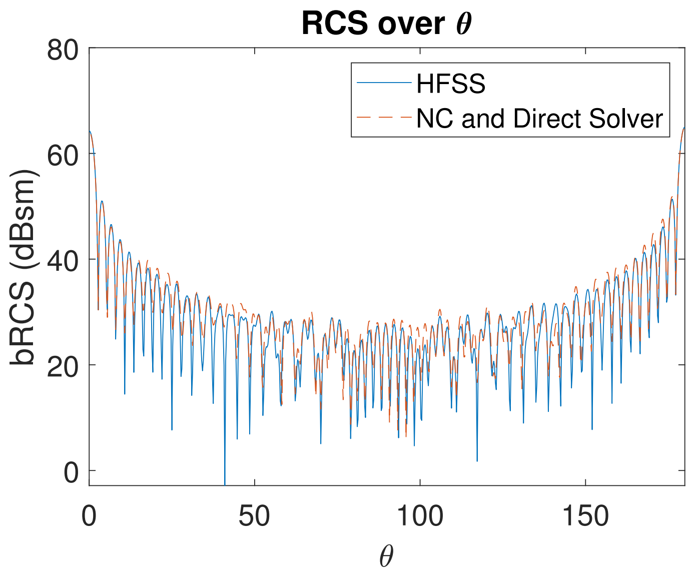


Figure 5.4. Scattering from a large cube with over one million unknowns simulated using Nested Construction.

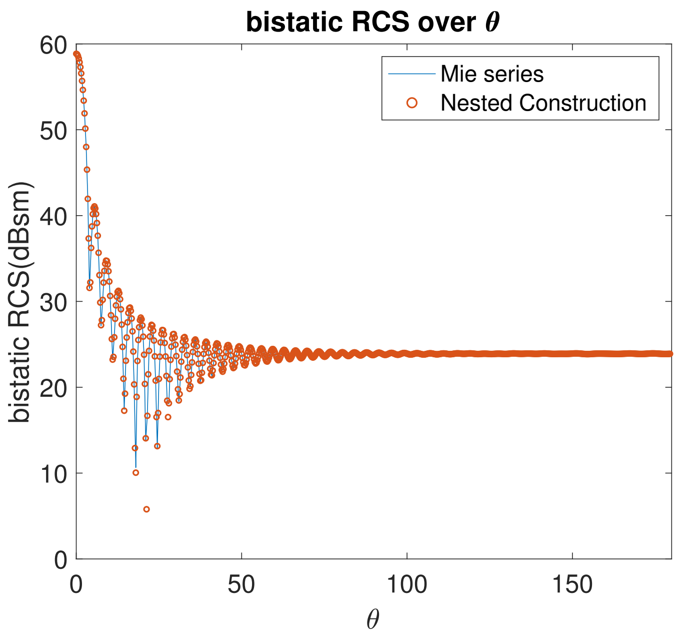


Figure 5.5. Scattering from a conducting sphere having $N = 294,912$ using Nested Construction.

The Growth Rate of the Rank

First, we examine the growth rate of the rank with electrical size since it is one of the key parameters in the complexity analysis. We use a conducting sphere as an example, and find its rank level by level by using the proposed Nested Construction algorithm with $\epsilon = 10^{-3}$. The results are listed in Table 5.4. In this Table, e_s denotes the largest electrical size of all clusters at a tree level, n is the maximal cluster size on one level and k is the maximal rank revealed by Nested Construction at this level.

Table 5.4. Rank versus Tree Level Using the NC Algorithm with $\epsilon = 10^{-3}$ for a Conducting Sphere.

Tree level	e_s	C_{sp}	n	k
3	17.18	7	49152.00	484
4	13.12	27	24576.00	521
5	8.55	39	12288.00	364
6	6.91	82	6144.00	238
7	4.38	112	3072.00	149
8	3.74	178	1536.00	102
9	2.32	216	768.00	67
10	2.05	368	384.00	49
11	1.25	407	192.00	34
12	1.16	753	96.00	27
13	0.74	780	48.00	21
14	0.70	1592	24.00	16

We also test a conducting cube and find its rank level by level using the proposed Nested Construction algorithm with $\epsilon = 10^{-3}$. The results are shown in Table 5.5. From this Table and Table 5.4, we observe that rank's growth rate with electrical size follows the linear rate closely.

Table 5.5. Rank versus Tree Level Using NC Algorithm with $\epsilon = 10^{-3}$ for a Conducting Cube.

Tree level	e_s	C_{sp}	n	k
3	15.08	4	49152.00	362
4	11.92	16	24576.00	355
5	7.54	38	12288.00	308
6	5.96	33	6144.00	236
7	3.77	45	3072.00	152
8	2.98	33	1536.00	103
9	1.88	45	768.00	68
10	1.49	33	384.00	49
11	0.94	45	192.00	36
12	0.75	33	96.00	27
13	0.47	45	48.00	21
14	0.37	64	24.00	17

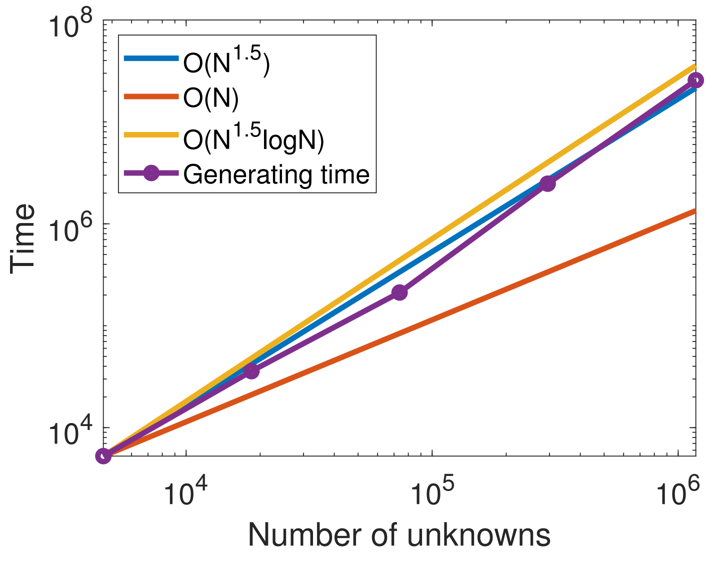


Figure 5.6. Time Complexity of the Proposed Nested Construction Algorithm.

Time and Memory Complexity

We use the scattering from a sphere to examine the complexity of the proposed algorithm. First, we simulate one sphere of diameter 17.68 m, whose number of unknowns is 294,912,

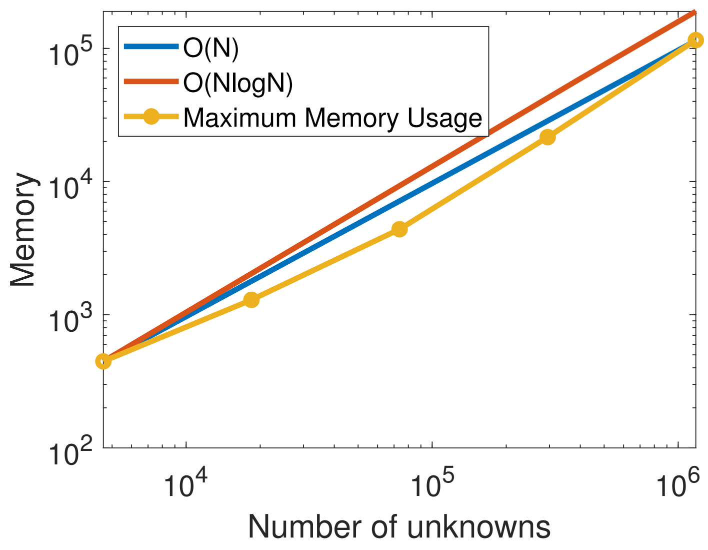


Figure 5.7. Memory Complexity of the Proposed Nested Construction Algorithm.

at 300 MHz. We use the proposed nested construction algorithm to construct an $\mathcal{H}^2$-matrix with $\epsilon = 10^{-3}$, and then employ the direct solver of [7] to solve the $\mathcal{H}^2$-matrix and compute the bi-static RCS. We then compare the results obtained with the MIE series solution in Fig. 5.5. Good agreement is observed.

Then we change the size of the sphere and simulate a range of sizes to examine the time and memory complexity of the proposed algorithm. The results are shown in Table 5.6. In this Table, N is the number of unknowns, t (s) is the total CPU time, mem (Mb) is the memory cost, E_{add} and E_{all} denote the relative error of admissible blocks of the generated $\mathcal{H}^2$-matrix, and the whole $\mathcal{H}^2$-matrix, respectively. The E_{add} and E_{all} are not feasible to assess for $N = 1,179,648$ case, and hence left blank. The $\max(k_t)$ denotes the maximum rank of the constructed $\mathcal{H}^2$-matrix.

Table 5.6. Scaling of CPU Time, Memory, Accuracy, Rank vs. N.

N	4608	18432	73728	294,912	1,179,648
t (s)	5.258e3	3.583e4	2.117e5	2.476e6	2.563e7
mem (Mb)	4.467e2	1.294e3	4.380e3	2.157e4	1.154e5
E_{add}	1.997e-3	2.431e-3	3.519e-3	5.157e-3	-
E_{all}	1.277e-4	1.959e-4	3.284e-4	5.373e-4	-
$\max(k_t)$	27	51	106	254	689

We also plot the time and memory scaling of the proposed NC algorithm with respect to N in Fig. 5.6 and 5.7 for the minimal-rank $\mathcal{H}^2$-generation. They are shown to agree very well with our theoretical complexity analysis.

5.5 Conclusion

We present a new algebraic and kernel independent method, nested construction method, to generate a minimal-rank $\mathcal{H}^2$-matrix to represent electrically large surface integral operators. Instead of approximating each admissible block, we find the low-rank approximation of the interaction between a cluster and its far-field. We also employ PSA with a random choice of pivots to reduce the computational cost of generating the low-rank approximation. We then develop an efficient nested algorithm to construct nested cluster bases level by level. After the cluster bases are built, we generate coupling matrices. The accuracy and complexity of the proposed are demonstrated by extensive numerical experiments. The proposed method not only can be used for constructing a rank-minimized $\mathcal{H}^2$-matrix representation of IE operators but also can be employed to carry out an efficient $\mathcal{H}$- to $\mathcal{H}^2$-matrix conversion.

6. NESTED PSEUDO SKELETON APPROXIMATION

In the previous chapter, we developed an algebraic method. In this chapter, we propose another algebraic kernel-independent method to construct a minimal rank $\mathcal{H}^2$-matrix to represent electrically large surface IE operators. We call the method a Nested Pseudo-Skeleton Approximation (NPSA), in which the cluster bases and coupling matrices are built to have an explicit relationship with the dense IE matrices being compressed. In this method, for each cluster in the $\mathcal{H}^2$-tree, we consider the interaction between the cluster (row cluster) and other admissible clusters (column clusters) at the same tree level, as well as all ancestor levels. To find the low-rank representation of such an interaction efficiently, we employ Pseudo-Skeleton Approximation [16], [17] (PSA), and also randomly choose $O(k)$ rows from the row cluster, and $O(k)$ columns from the column clusters to build a low-rank representation. In this way, only $O(N \log N)$ elements of the original dense matrix are used to generate the $\mathcal{H}^2$-matrix irrespective of the large electrical size. It also provides a closed-form expression of the nested cluster bases and coupling matrices using original matrix entries. In contrast, the algebraic method developed in the previous chapter needs to find a nested relationship numerically via computation, which is more time consuming. The computational cost of generating each cluster basis and coupling matrix is only of $O(k^3)$, while the memory consumption scales as $O(k^2)$, where k is its rank. The resultant $\mathcal{H}^2$-matrix is then solved directly or iteratively in an efficient manner. Taking the rank's growth with electrical size into consideration [9], the total time complexity of the proposed method is $O(N^{1.5})$ for the SIE and the memory complexity scales as $O(N \log N)$. Meanwhile, it is kernel-independent. Since the new method does not depend on MLFMA and is purely algebraic, it is also easier to implement. In addition to surface integral equations, the proposed algorithms can also be applied to solving other electrically large integral equations. Comparisons with analytical Mie series and reference solutions from a commercial tool have validated the accuracy and efficiency of the proposed method for solving electrically large SIEs.

In this chapter, to simplify the notation, let $\boldsymbol{Z}$ denote the MoM matrix of SIEs. For a set of rows r and a set of columns c of $\boldsymbol{Z}$, let $\boldsymbol{Z}(r, c)$ denote the submatrix of $\boldsymbol{Z}$ composed of r and c. The $\hat{t}^{+}$ includes all clusters s that are admissible with t at t's level, as well as those that are admissible with t's ancestors, termed far field of t. We also use t to denote the set of indexes for cluster t and $\hat{t}^+$ to denote the indexes of far field of t. Thus, $\boldsymbol{G}_t = \boldsymbol{Z}(t, \hat{t}^+)$.

Let r^t denote a set of randomly selected rows from t, where the superscript denotes the index of cluster. Let c^t denote a set of randomly selected columns from $\hat{t}^+$ for cluster t. Based on the PSA, we have

$$\mathbf{G}^t = \boldsymbol{Z}(t, \hat{t}^+) \approx \boldsymbol{Z}(t, c^t)\boldsymbol{Z}(r^t, c^t)^{\dagger}\boldsymbol{Z}(r^t, \hat{t}^+), \tag{6.1}$$

where $\boldsymbol{Z}(t, c^t)$ are selected columns of $\mathbf{G}^t$ based on c^t, while $\boldsymbol{Z}(r^t, \hat{t}^+)$ are selected rows of $\mathbf{G}^t$ based on r^t, $\mathbf{G}(r^t, c^t)$ is the intersection between the rows in r^t and columns in c^t of $\boldsymbol{G}^t$, having $|r^t| \times |c^t|$ entries, and $\dagger$ denotes a pseudo-inverse, which is again achieved using reduced SVD. The robustness and accuracy of the PSA can be improved if the pivots are selected properly. If we make the locations of the pivots spread evenly in the far field and cover as many directions as possible, the PSA will have a good accuracy. We can also artificially introduce imaginary pivots to represent the far field interaction. The improvement of the PSA will be in next chapter, where we eliminate the uncertainty arising from a random choice of pivots and make sure it succeeds every time.

6.1 Nested Pseudo-Skeleton Approximation

The PSA does not result in a nested structure. In this section, we develop a nested PSA algorithm to construct an $\mathcal{H}^2$-matrix. This method can construct a minimal-rank or low-rank $\mathcal{H}^2$-matrix in $O(k^3)$ for each cluster, which is better than the algorithm proposed in the previous chapter that results in an $O(nk \log n)$ complexity for each cluster. Thus, although this method has the similar complexity as Nested Construction for SIE, this method can be of lower complexity since k is in general smaller than $n^{0.5}$, the constant involved in the new algorithm is also smaller.

We will introduce a few variations of the main algorithm. But first, we will introduce a framework that is common in all these variations. The framework is similar to our previous algorithms. NPSA can be done by calling **Algorithm** 18. We first construct the cluster bases using **Algorithm** 19, which consists of one bottom-up traversal of the cluster tree. Then we construct the coupling matrices using **Algorithm** 20, which consists of one bottom-up traversal of block cluster tree. In different variations, the methods of fill_non_leaf_cluster(t), fill_leaf_cluster(t) and fill_coupling_matrix(b) are different.

6.1.1 Nested Pseudo-Skeleton Approximation

We first describe our first version of the NPSA algorithm, and then proceed to more advanced ones. For a leaf cluster, we randomly choose a subset of $\hat{t}^+$, c^t. It is enough to choose the size of c^t to be $O(k)$, where k is the numerical rank of $\boldsymbol{G}$. But we can choose $|c^t| = O(n)$, where n is the size of a leaf cluster because this will not affect our desired complexity bound. Then $\boldsymbol{V}^t$ is just the c^t columns of $\boldsymbol{G}^t$, $\boldsymbol{V}^t = \boldsymbol{Z}(t, c^t)$. This is formalized in **Algorithm** 25. Based on the existence of PSA, $\boldsymbol{V}^t$ can represent $\boldsymbol{G}^t$, meaning $\boldsymbol{G}^t$ can be obtained by multiplying $\boldsymbol{V}^t$ with another matrix on the right with high accuracy.

Algorithm 25 fill_leaf_cluster

```
1: procedure fill_leaf_cluster($t$)
2:     $\boldsymbol{V}^t = \boldsymbol{Z}(t, c^t)$
3: end procedure
```

For a non-leaf cluster, we also choose randomly a set of pivots c^t from $\hat{t}^+$. This time, the size of c^t must be guided by the estimation of the rank for $\boldsymbol{G}^t$, which is $O(n^{0.5})$ as shown in [9]. The cluster basis for t is also $\mathbf{V}^t$ that formed by t and c^t, $\mathbf{V}^t = \boldsymbol{Z}(t, c^t)$. But we do not compute or store it directly, because it is too expensive. Suppose t has two children t_1 and t_2. We want $\mathbf{T}^{t_1}$ and $\mathbf{T}^{t_2}$ such that $\mathbf{V}^t = \begin{bmatrix} \mathbf{V}^{t_1} & \\ & \mathbf{V}^{t_2} \end{bmatrix} \begin{bmatrix} \mathbf{T}^{t_1} \\ \mathbf{T}^{t_2} \end{bmatrix}$. Split $\mathbf{V}^t = \begin{bmatrix} \mathbf{V}^t_1 \\ \mathbf{V}^t_2 \end{bmatrix}$ correspondingly, such that $\boldsymbol{V}^t_1 = \boldsymbol{Z}(t_1, c^t)$. Now we need $\mathbf{V}^{t_1}\mathbf{T}^{t_1} = \mathbf{V}^t_1$. Remember $\mathbf{V}^{t_1} = \boldsymbol{Z}(t_1, c^{t_1})$ can represent $\boldsymbol{Z}(t_1, \hat{t}_1^+)$ and thus can represent $\boldsymbol{V}^t_1 = \boldsymbol{Z}(t_1, c^t)$. We can do the PSA on $\begin{bmatrix} \mathbf{V}^{t_1} & \mathbf{V}^t_1 \end{bmatrix}$, which is $\begin{bmatrix} \boldsymbol{Z}(t_1, c^{t_1}) & \boldsymbol{Z}(t_1, c^t) \end{bmatrix}$. According to the PSA, we can choose randomly a set of rows r^{t_1} from t_1, whose size is $|r^{t_1}| = |c^{t_1}|$. Then we have

$$\boldsymbol{Z}(t_1, c^t) = \boldsymbol{Z}(t_1, c^{t_1})\boldsymbol{Z}(r^{t_1}, c^{t_1})^\dagger \boldsymbol{Z}(r^{t_1}, c^t) \tag{6.2}$$

This will hold no matter c^{t_1} and c^t have overlap, or they are completely separated, since

$$\boldsymbol{Z}(t_1, c^{t_1} \cup c^t) = \boldsymbol{Z}(t_1, c^{t_1})\boldsymbol{Z}(r^{t_1}, c^{t_1})^\dagger \boldsymbol{Z}(r^{t_1}, c^{t_1} \cup c^t) \tag{6.3}$$

Naturally, $\mathbf{T}^{t_1} = \boldsymbol{Z}(r^{t_1}, c^{t_1})^\dagger \boldsymbol{Z}(r^{t_1}, c^t)$. Similarly, $\mathbf{T}^{t_2} = \boldsymbol{Z}(r^{t_2}, c^{t_2})^\dagger \boldsymbol{Z}(r^{t_2}, c^t)$. This is formalized in **Algorithm** 26

Algorithm 26 fill_non_leaf_cluster

```
1: procedure fill_non_leaf_cluster($t$)
2:     for $t$'s ith child do
3:         $\mathbf{T}^{t_i} = \boldsymbol{Z}(r^{t_i}, c^{t_i})^\dagger \boldsymbol{Z}(r^{t_i}, c^t)$
4:     end for
5: end procedure
```

For coupling matrix b composed of t and s, we want to approximate $\boldsymbol{Z}(t,s)$. According to the PSA, we have $\boldsymbol{Z}(t,s) = \boldsymbol{Z}(t,c^b)\boldsymbol{Z}(r^b,c^b)^\dagger \boldsymbol{Z}(r^b,s)$, where c^b is a set of selected columns from s and r^b is a set of selected rows from t. Remember we have $\boldsymbol{V}^t = \boldsymbol{Z}(t,c^t)$. Thus, as before, we can do the PSA on $\begin{bmatrix} \boldsymbol{Z}(t,c^t) & \boldsymbol{Z}(t,c^b) \end{bmatrix}$. Then we will need a set of rows r^t from t such that $\boldsymbol{Z}(t,c^b) = \boldsymbol{Z}(t,c^t)\boldsymbol{Z}(r^t,c^t)^\dagger \boldsymbol{Z}(r^t,c^b)$. Obviously, r^t can be chosen the same as r^b. Similarly, for s, we can have a set of columns r^s from s such that $\boldsymbol{Z}(s,r^b) = \boldsymbol{Z}(s,c^s)\boldsymbol{Z}(r^s,c^s)^\dagger \boldsymbol{Z}(r^s,r^b)$. Finally, we obtain the following coupling matrix

$$\mathbf{S}^{t,s} = \boldsymbol{Z}(r^t,c^t)^\dagger \boldsymbol{Z}(r^t,c^b)\boldsymbol{Z}(r^b,c^b)^\dagger (\boldsymbol{Z}(r^s,c^s)^\dagger \boldsymbol{Z}(r^s,r^b))^T \tag{6.4}$$

It can be readily verified that $\boldsymbol{Z}(t,s) = \boldsymbol{V}^t \mathbf{S}^{t,s} (\boldsymbol{V}^s)^T$. This is formalized in **Algorithm** 27. Although (6.4) seems to be a long equation, in fact, each matrix is of size $O(k) \times O(k)$, which is $O(n^{0.5}) \times O(n^{0.5})$. Thus, total time complexity if $O(k^3)$ and memory complexity is $O(k^2)$.

Algorithm 27 fill_coupling_matrix

```
1: procedure fill_coupling_matrix($b$)
2:     Fill $\mathbf{S}^{t,s}$ using (6.4).
3: end procedure
```

In the above algorithms, we choose randomly the set of pivots directly from $\hat{t}^+$ for each cluster. We can also choose c^t as $c^t = \cup_{s\in\hat{t}^+} c^t_s$, where c^t_s is the column set for the cluster s in $\hat{t}^+$, and construct leaf and non-leaf cluster bases using aforementioned methods. Each c^t_s can have a size of $|c^t_s| = O(k)$, thus $|c^t| = O(k \log N)$, since t has at most $\log N$ ancestors and each ancestor has at most C_{sp} admissible blocks at its own level. We can construct leaf and nonleaf clusters using this c^t. In this way, for a coupling matrix block $b^{t,s}$, we need a column set c^b and a row set r^b. By construction, $\mathbf{V}^t$ has already contained the necessary c^b, which is c^t_s. We can set $c^b = c^t_s$. Similarly, $\mathbf{V}^s$ contains necessary r^b, which is c^s_t. We can set $r^b = c^s_t$. we can directly form the coupling matrix $\mathbf{S}^{t,s} = \boldsymbol{Z}(c^s_t, c^t_s)^\dagger$ based on c^s_t and c^t_s. When computing the $\boldsymbol{Z}(t,s)$, $\mathbf{S}^{t,s}$ is multiplied with corresponding part of $\boldsymbol{V}^t$ and $\boldsymbol{V}^s$. Seeing from another way, we put $\boldsymbol{Z}(c^s_t, c^t_s)^\dagger$ in corresponding position in $\mathbf{S}^{t,s}$ based on the position of c^t_s in $\mathbf{V}^t$ and c^s_t in $\mathbf{V}^s$, and make the rest of $\mathbf{S}^{t,s}$ as zero. Then $\mathbf{S}^{t,s}$ does not have to be formed explicitly since $\mathbf{S}^{t,s}$ has so many zeros.

If the minimized rank is desired, a rank-minimized procedure [18] can be applied to the resultant $\mathcal{H}^2$-matrix in $O(k^3)$ complexity, since current $\mathcal{H}^2$-matrix has a rank of $O(k)$.

6.1.2 Rank-Minimization On the Fly

In the above, we describe an NPSA algorithm to construct a low-rank $\mathcal{H}^2$-matrix. The resultant cluster bases are not orthonormal, and the rank is not minimized for accuracy either. We have to add one rank-minimization process to make the cluster basis in the above $\mathcal{H}^2$-matrix minimal-rank and orthonormal. But actually, we can do the rank-minimization at the same time when constructing the $\mathcal{H}^2$-matrix. This can reduce the memory requirement. This can be done by introducing an auxiliary matrix $\tilde{\boldsymbol{U}}^t$. Let the new set of orthonormal cluster basis be $\tilde{\boldsymbol{V}}^t$, $\tilde{\boldsymbol{T}}^t$, and the corresponding coupling matrix be $\tilde{\boldsymbol{S}}^{t,s}$.

For a leaf cluster, we can do a reduced SVD on $\boldsymbol{V}^t$ to get $\boldsymbol{V}^t = \tilde{\boldsymbol{V}}^t(\tilde{\boldsymbol{U}}^t)^H$ based on a certain accuracy criterion ϵ, where $\tilde{\boldsymbol{V}}^t$ is the left singular vectors. Then $\tilde{\boldsymbol{V}}^t$ is our new minimal-rank cluster basis which is orthonormal. The $(\tilde{\boldsymbol{U}}^t)^H$ is nothing but $(\tilde{\boldsymbol{V}}^t)^H\boldsymbol{V}^t$. This procedure is formalized in **Algorithm** 28

Algorithm 28 fill_leaf_cluster

```
1: procedure fill_leaf_cluster($t$)
2:     $\boldsymbol{V}^t = \boldsymbol{Z}(t, c^t)$
3:     $\tilde{\boldsymbol{V}}^t$, $(\tilde{\boldsymbol{U}}^t)^H \leftarrow$ do SVD on $\boldsymbol{V}^t$
4: end procedure
```

For a non-leaf cluster, we do rSVD on $\tilde{\mathbf{G}}^t = \begin{bmatrix} (\tilde{\mathbf{U}}^{t_1})^H \mathbf{T}^{t_1} \\ (\tilde{\mathbf{U}}^{t_2})^H \mathbf{T}^{t_2} \end{bmatrix}$ to get $\tilde{\mathbf{G}}^t = \begin{bmatrix} \tilde{\mathbf{T}}^{t_1} \\ \tilde{\mathbf{T}}^{t_2} \end{bmatrix} (\tilde{\mathbf{U}}^t)^H$, where $\begin{bmatrix} \tilde{\mathbf{T}}^{t_1} \\ \tilde{\mathbf{T}}^{t_2} \end{bmatrix}$, the left singular vectors will be the minimal-rank transfer matrix, and $(\tilde{\mathbf{U}}^t)^H$ will be used to update the transfer matrix of its parents. This procedure is formalized in **Algorithm** 29.

Algorithm 29 fill_nonleaf_cluster

```
1: procedure fill_nonleaf_cluster($t$)
2:     for $t$'s ith child do
3:         $\mathbf{T}^{t_i} = \boldsymbol{Z}(r^{t_i}, c^{t_i})^\dagger \boldsymbol{Z}(r^{t_i}, c^t)$
4:     end for
5:     compute $\tilde{\mathbf{G}}^t = \begin{bmatrix} (\tilde{\mathbf{U}}^{t_1})^H \mathbf{T}^{t_1} \\ (\tilde{\mathbf{U}}^{t_2})^H \mathbf{T}^{t_2} \end{bmatrix}$
6:     $\begin{bmatrix} \tilde{\mathbf{T}}^{t_1} \\ \tilde{\mathbf{T}}^{t_2} \end{bmatrix}$, $(\tilde{\mathbf{U}}^t)^H \leftarrow$ do rSVD on $\tilde{\mathbf{G}}^t$
7: end procedure
```

For coupling matrix, we can generate the minimal-rank coupling matrix as $\tilde{\mathbf{S}}^{t,s} = (\tilde{\mathbf{U}}^t)^H \mathbf{S}^{t,s} \tilde{\mathbf{U}}^s$, as in **Algorithm** 30.

Algorithm 30 fill_coupling_matrix

```
1: procedure fill_coupling_matrix($b$)
2:     Fill $\mathbf{S}^{t,s}$ using (6.4).
3:     $\tilde{\mathbf{S}}^{t,s} = (\tilde{\mathbf{U}}^t)^H \mathbf{S}^{t,s} \tilde{\mathbf{U}}^s$
4: end procedure
```

There is another way of choosing $\tilde{\mathbf{U}}^t$, which simplifies the above procedure. Consider (6.1). For a cluster t, Do rSVD on $\boldsymbol{Z}(t, c^t)$ to get $\boldsymbol{Z}(t, c^t) = \tilde{\boldsymbol{V}}^t \boldsymbol{\Sigma}^t (\boldsymbol{U}^t)^H$. Then select the set of rows out of $\tilde{\mathbf{V}}^t$ to form a new matrix $\hat{\mathbf{V}}^r$ such that $\boldsymbol{Z}(r^t, c^t) = \hat{\mathbf{V}}^r \mathbf{S}^r (\mathbf{U}^r)^H$. This can be done because $\boldsymbol{Z}(r^t, c^t)$ consists of some rows of $\boldsymbol{Z}(t, c^t)$. Then it is easy to verify

$$\boldsymbol{Z}(t, c^t) \boldsymbol{Z}(r^t, c^t)^\dagger = \tilde{\mathbf{V}}^r (\hat{\mathbf{V}}^r)^\dagger \tag{6.5}$$

Thus, for leaf cluster, we first get $\mathbf{V}^t$ as in subsection 6.1.1 and do SVD on it to get left singular vectors $\tilde{\mathbf{V}}^t$. Then select r^t rows on this cluster. As before, $|c^r| = O(k)$. Form a submatrix of $\tilde{\mathbf{V}}^t$ based on r^t and denote this submatrix as $\hat{\mathbf{V}}^t$. Then $(\tilde{\mathbf{U}}^t)^H = (\hat{\mathbf{V}}^t)^\dagger$.

Then for non-leaf cluster, we compute $\tilde{\mathbf{G}}^t = \begin{bmatrix} (\tilde{\mathbf{U}}^{t_1})^H \boldsymbol{Z}(t_1, c^{t_1}) \\ (\tilde{\mathbf{U}}^{t_2})^H \boldsymbol{Z}(t_2, c^{t_2}) \end{bmatrix}$. Then do rSVD on $\tilde{\mathbf{G}}^t$ to get left singular vectors $\tilde{\mathbf{T}}^t$. Splitting it into two will give us new rank-minimal transfer matrix $\tilde{\mathbf{T}}^t = \begin{bmatrix} \tilde{\mathbf{T}}^{t_1} \\ \tilde{\mathbf{T}}^{t_2} \end{bmatrix}$. To get $(\tilde{\mathbf{U}}^t)^H$ for a non-leaf cluster t, we need rows of $\tilde{\mathbf{V}}^t$. But we do not store them in the $\mathcal{H}^2$-matrix. We can maintain a short version of $\tilde{\mathbf{V}}^t$ for every cluster t as $\check{\mathbf{V}}^t = \begin{bmatrix} \hat{\mathbf{V}}^{t_1}\tilde{\mathbf{T}}^{t_1} \\ \hat{\mathbf{V}}^{t_2}\tilde{\mathbf{T}}^{t_2} \end{bmatrix}$ for example. We can choose r^t from $r^{t_1} \cup r^{t_2}$. In this way, we can choose $\hat{\mathbf{V}}^t$ from $\check{\mathbf{V}}^t$. Here $\check{\mathbf{V}}^t$ is also of size $k^t \times k^t$.

One major benefit of doing this is the computation of coupling matrix is greatly simplified. For a coupling matrix $b^{t,s}$, we simply compute $\boldsymbol{S}^{t,s} = (\tilde{\mathbf{U}}^t)^H \mathbf{Z}(r^s, r^t)^\dagger \tilde{\mathbf{U}}^s$. Thus the simplified computation of coupling matrices might worth the complication of the computation of cluster basis.

6.1.3 NPSA Algorithm Using Original Matrix Entries for Coupling Matrices

We can make a little modification to make the coupling matrices be generated using original matrix entries. One additional advantage of this variation is that the size of the set of randomly chosen pivots can be guided, or can be chosen adaptively. In this subsection, we use $\boldsymbol{V}^t$, $\boldsymbol{T}^t$ to denote cluster bases and transfer matrices and they are not orthonormal.

For leaf cluster, we first randomly choose r^t and c^t from t and $\hat{t}^+$, respectively, and compute $\boldsymbol{V}^t = \boldsymbol{Z}(t, c^t)\boldsymbol{Z}(r^t, c^t)^\dagger$. Note the pseudo-inverse of $\mathbf{V}^t$ will give us the minimal-rank of cluster t, k^t. This rank will guide the size of c and r for its parent cluster. Thus, this new scheme is also adaptive in terms of $\mathcal{H}^2$-rank. For leaf cluster, we can set $r^t = \hat{t}$, where $\hat{t}$ denotes the set of all row pivots of cluster t, since this will not affect overall complexity. Observe (6.1), we have $\boldsymbol{G}^t \approx \boldsymbol{V}^t\boldsymbol{Z}(r^t, \hat{t}^+)$. $\boldsymbol{V}$ has another important property. For an arbitrary subset c of $\hat{t}^+$, we have $\boldsymbol{V}^t\boldsymbol{Z}(r^t, c) = \boldsymbol{Z}(t, c)$. This procedure is formalized in **Algorithm** 31.

Algorithm 31 fill_leaf_cluster

```
1: procedure fill_leaf_cluster($t$)
2:     $\boldsymbol{V}^t = \boldsymbol{Z}(t, c^t)\boldsymbol{Z}(r^t, c^t)^\dagger$
3: end procedure
```

For a non-leaf cluster t, the size of c^t and r^t can be chosen as $|c^t| = |r^t| = c(\sum_{\mathrm{i}} k^{t_{\mathrm{i}}})$, where c is a constant coefficient, t_{i} is the i-th child of t and $k^{t_{\mathrm{i}}}$ is the minimal rank of t_{i}. This choice is due to the fact that the numerical rank k^t must be smaller than $\sum_{\mathrm{i}} k^{t_{\mathrm{i}}}$ because the union of $\boldsymbol{G}^{t_{\mathrm{i}}}$ contains $\boldsymbol{G}^t$. Thus, we have $\boldsymbol{G}^t \approx \boldsymbol{Z}(t, c^t)\boldsymbol{Z}(r^t, c^t)^\dagger \boldsymbol{Z}(r^t, \hat{t}^+)$. Since the children cluster's $\hat{t}^+$ include parent clusters' $\hat{t}^+$, we have $\boldsymbol{Z}(t, c^t) = \begin{bmatrix} \mathbf{V}^{t_1} & \\ & \mathbf{V}^{t_2} \end{bmatrix} \begin{bmatrix} \boldsymbol{Z}(r^{t_1}, c^t) \\ \boldsymbol{Z}(r^{t_2}, c^t) \end{bmatrix}$. Hence, it can be readily recognized that the transfer matrix $\boldsymbol{T}^{t_{\mathrm{i}}} = \boldsymbol{Z}(r^{t_{\mathrm{i}}}, c^t)\mathbf{Z}(r^t, c^t)^\dagger$. Again, the pseudo-inverse of $\boldsymbol{Z}(r^t, c^t)$ gives us the minimal-rank of this cluster t, k^t. This rank will determine the size of c and r for its parent cluster. As in the leaf cluster, we have $\boldsymbol{G}^t \approx \boldsymbol{V}^t \boldsymbol{Z}(r^t, \hat{t}^+)$, when we write $\boldsymbol{V}^t$ in terms of $\begin{bmatrix} \boldsymbol{V}^{t_1} & \\ & \boldsymbol{V}^{t_2} \end{bmatrix} \begin{bmatrix} \boldsymbol{T}^{t_1} \\ \boldsymbol{T}^{t_2} \end{bmatrix}$. Again, $\boldsymbol{V}^t$ can represent $\boldsymbol{G}^t$. $\boldsymbol{T}$ also has an similar property as for leaf cluster. For an arbitrary subset c of $\hat{t}^+$, we have $\boldsymbol{V}^t \boldsymbol{Z}(r^t, c) = \boldsymbol{Z}(t, c)$. These properties ensure the bottom-up traversal process can be done from leaf level all the way to the highest level that has admissible block. This procedure is formalized in **Algorithm** 32.

Algorithm 32 fill_nonleaf_cluster

```
1: procedure fill_nonleaf_cluster($t$)
2:     for $t$'s ith child do
3:         $\boldsymbol{T}^{t_{\mathrm{i}}} = \boldsymbol{Z}(r^{t_{\mathrm{i}}}, c^t)\boldsymbol{Z}(r^t, c^t)^\dagger$
4:     end for
5: end procedure
```

The advantage of the above procedure finally emerges when it comes to the computation of the coupling matrix. For a coupling matrix $b^{t,s}$, $\boldsymbol{S}^{t,s}$ is just the submatrix of $\mathbf{M}^{t,s}$ corresponding to r^t of row cluster t and r^s of column cluster s. Thus, $\boldsymbol{S}^{t,s} = \boldsymbol{Z}(r^t, r^s)$. This is because $\boldsymbol{V}^t \boldsymbol{Z}(r^t, r^s) = \boldsymbol{Z}(t, r^s)$, and $\boldsymbol{V}^s \boldsymbol{Z}(r^s, t) = \boldsymbol{Z}(s, t)$. Since $|r^t| = |r^s| = O(k)$, the complexity of generating each coupling matrix is $O(k^2) = O(n)$. This is formalized in **Algorithm** 33.

In this scheme, we make the generation of coupling matrix $O(k^2)$. But the resulting $\mathcal{H}^2$-matrix is not minimal-rank, but a low-rank one. If minimal-rank is wanted, another rank minimization procedure of complexity $O(k^3)$ for each cluster basis and each coupling matrix

Algorithm 33 fill_coupling_matrix

```
1: procedure fill_coupling_matrix($b$)
2:     $\boldsymbol{S}^{t,s} = \boldsymbol{Z}(r^t, r^s)$
3: end procedure
```

can be applied [8]. The rank-minimization can also be done on the fly using the algorithm described in previous section.

6.1.4 Accuracy and Complexity

The accuracy and efficiency of the PSA with a random choice of pivots are discussed in chapter 5. The improvement of the PSA will be discussed in the next chapter, where we eliminate the uncertainty and make sure it succeeds every time.

In the following complexity analysis, we use

$$k^t = \sqrt{n} \tag{6.6}$$

for each cluster basis for electrically large SIE operators.

For subsection 6.1.1, for each leaf cluster, the time complexity is $O(N)$. For each non-leaf cluster, we need to generate $\boldsymbol{Z}(r^{t_{\mathrm{i}}}, c^{t_{\mathrm{i}}})$, $\boldsymbol{Z}(r^{t_{\mathrm{i}}}, c^{t})$ and do $\boldsymbol{Z}(r^{t_{\mathrm{i}}}, c^{t_{\mathrm{i}}})^{\dagger}\boldsymbol{Z}(r^{t_{\mathrm{i}}}, c^{t})$. Both $\boldsymbol{Z}(r^{t_{\mathrm{i}}}, c^{t_{\mathrm{i}}})$ and $\boldsymbol{Z}(r^{t_{\mathrm{i}}}, c^{t})$ are of size $O(k^t) \times O(k^t)$. Thus, the time complexity for each cluster is $O((k^t)^3)$, while memory complexity for each cluster is $O((k^t)^2)$. For coupling matrix, we have concluded that the time complexity for each coupling matrix is $O((k^t)^3)$ while memory complexity is $O((K^t)^2)$. Hence, the total time complexity for NPSA is

$$\begin{aligned} C_t &= \sum_{l=0}^{L} 2^l O((k^t)^3) \\ &= \sum_{l=0}^{L} 2^l O\left(\left(\sqrt{\frac{N}{2^l}}\right)^3\right) = O(N^{1.5}). \end{aligned} \tag{6.7}$$

While, total memory complexity is

$$\begin{aligned} C_m &= \sum_{l=0}^{L} (2^l O(k^2)) \\ &= \sum_{l=0}^{L} 2^l O\left(\frac{N}{2^l}\right) = O(N \log N) \end{aligned} \tag{6.8}$$

Subsection 6.1.2 introduces another $O(k^t) \times O(k^t)$ matrix $\tilde{\mathbf{U}}$, and thus it will not affect the above complexity.

Subsection 6.1.3 makes time complexity $O((k^t)^2)$ for each coupling matrix, thus the total time complexity of generating coupling matrices will be $O(N \log N)$.

6.2 Numerical Results

A common set of simulation parameters are used for all examples simulated in this section. Specifically, we choose $\eta = 1.2$ in the admissibility condition, $leafsize$ to be 40 and citation frequency to be f = 300MHz. The only simulation parameter in the proposed NPSA is accuracy parameter ϵ, which is a user-defined parameter. The enlarging coefficient c is normally chosen to be 4. The c can be chosen as an arbitrary constant, but a larger choice of c would increase CPU run time. In the case that the direct solver of [7] is used to solve the $\mathcal{H}^2$-matrix generated from the proposed algorithm, the accuracy is set to be 10^{-3}, unless stated otherwise.

6.2.1 Accuracy

We first examine the accuracy of the proposed algorithm before examining its complexity.

Scattering from a Conducting Cube

In this example, we compute the bistatic RCS of a conducting cube of size $21.324m \times 21.324m \times 21.324m$ at 300 MHz, which has 1,179,648 unknowns. The new $\mathcal{H}^2$-matrix generated from the NPSA in subsection 6.1.2 is solved using the direct solver of [7]. The resultant bistatic RCS is compared with that from HFSS, which shows very good agreement, as can be seen from Fig. 6.1. In this example, the accuracy criterion used in the NPSA is set to be $\epsilon = 10^{-3}$.

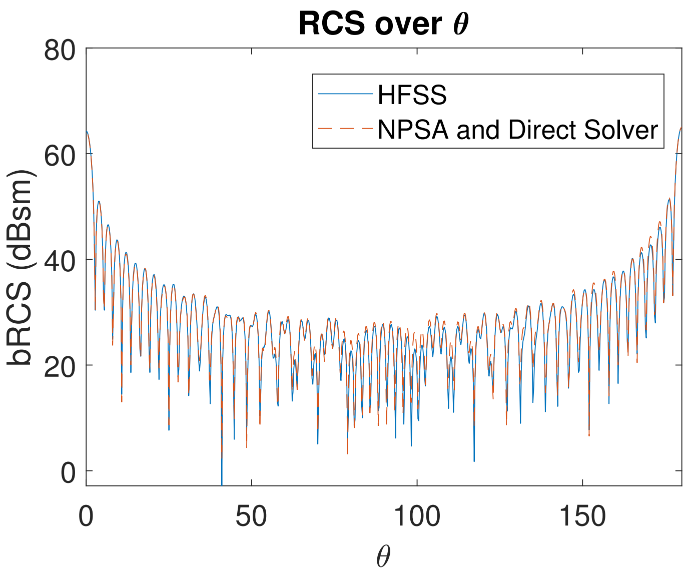


Figure 6.1. bRCS, Nested Pseudo-Skeleton Approximation, N=1179648, Cube

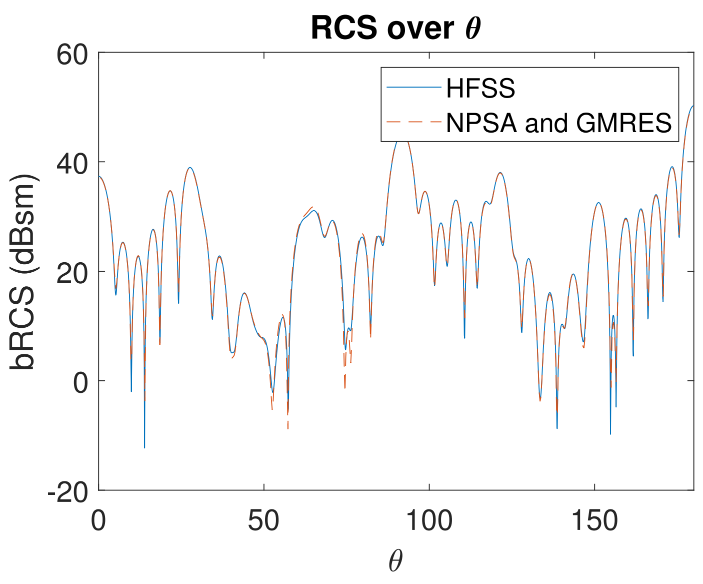


Figure 6.2. bRCS, Nested Pseudo-Skeleton Approximation, N=139968, array of spheres, $6 \times 6 \times 6$ array

Scattering from an Array of Spheres

In this example, we simulate an array of spheres, having $6 \times 6 \times 6$ spheres, each of which has a diameter of 0.8288 m and is discretized with 648 unknowns at 300 MHz. The distance between the two adjacent spheres is 2.0720 m. The bistatic RCS is computed using GMRES with Block Diagonal Preconditioner on the minimal-rank $\mathcal{H}^2$-matrix generated by NPSA in subsection 6.1.2. The result is compared with HFSS's result. Very good agreement is observed, as can be seen from Fig. 6.2. The accuracy criterion used in the NPSA is $\epsilon = 10^{-3}$.

Scattering from More Complicated Structures

We also simulate a coil, whose shape is shown in Fig. 3.3. This coil has a diameter of 14.156 m and is illuminated by an incident field at 300 MHz. After discretization, the number of unknowns is 121,914. The new $\mathcal{H}^2$-matrix generated from the NPSA in subsection 6.1.2 is then solved using the fast direct solver of [7]. The resultant bistatic RCS is compared with HFSS's result in Fig. 6.3. As can be seen, the two agree well with each other.

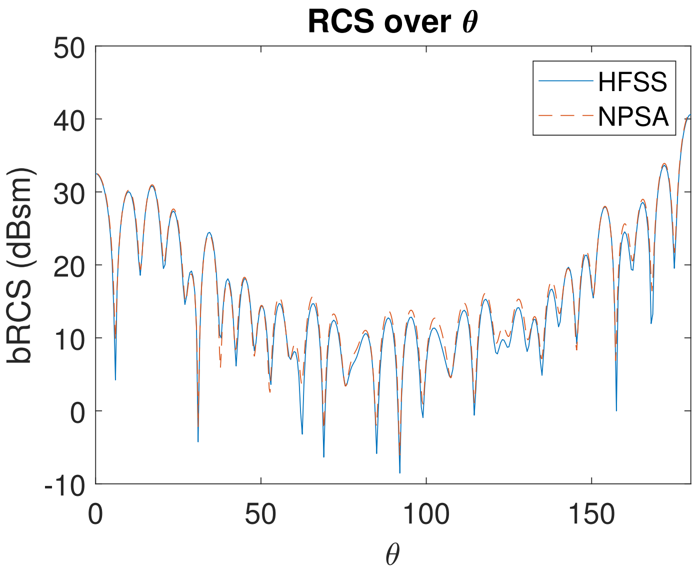


Figure 6.3. bRCS, Nested Pseudo-Skeleton Approximation, N=139968, coil

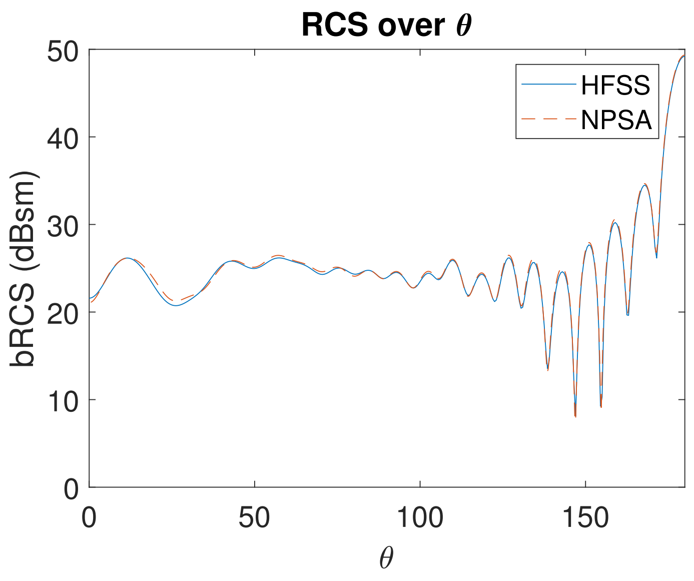


Figure 6.4. bRCS, Nested Pseudo-Skeleton Approximation, N=139968, joint

Another example we simulated is shown in Fig. 3.5. The structure has a diameter of 17.302 m and is discretized into 172,077 unknowns at 300 MHz. The new $\mathcal{H}^2$-matrix generated from NPSA in subsection 6.1.2 is again solved using the fast direct solver of [7]. The resultant bistatic RCS is compared with HFSS's result in Fig. 6.4. Good agreement can be observed.

6.2.2 The Growth Rate of the Rank

Here we examine the growth rate of the rank with electrical size since it is one of the key parameters in the complexity analysis. We use a conducting sphere as an example, and find its rank level by level using the proposed NPSA algorithm in subsection 6.1.2 with $\epsilon = 10^{-3}$. The results are listed in Table 6.1. In this table, e_s denotes the largest electrical size of all clusters at a tree level, n is the largest unknown number of all clusters, k is the minimal rank of the new $\mathcal{H}^2$-matrix of the entire SIE obtained from the proposed algorithm. It can be seen the rank scales linearly with electrical size, which agrees with the one used in our complexity analysis.

Table 6.1. Rank versus tree level, $\epsilon = 10^{-3}$, Sphere, diameter is 29.482 λ, N= 1,179,648

tree level	e_s	n	k
3	17.18	49152	483
4	13.12	24576	516
5	8.55	12288	366
6	6.91	6144	242
7	4.38	3072	151
8	3.74	1536	103
9	2.32	768	68
10	2.05	384	50
11	1.25	192	34
12	1.16	96	27
13	0.74	48	21
14	0.70	24	16

6.2.3 Complexity

We use the proposed NPSA in subsection 6.1.2 to construct an $\mathcal{H}^2$-matrix. Then we use the direct solver to solve this new $\mathcal{H}^2$-matrix equation and compute the bRCS. We then compare the results obtained in such way to the MIE series [15], and show it in Fig. 6.5. In this figure, the diameter of the sphere is 14.734 m, whose number of unknowns is 294,912, f=300MHz. Then we also simulate a suite of spheres of various sizes to numerically show the complexity of our converting algorithm at Table 6.2. In this table, $\epsilon = 10^{-3}$, N is the number of unknowns, t(s) is the time needed to construct the $\mathcal{H}^2$-matrix using NPSA, t_{cb}(s) is the time of generating cluster basis, t_{cm}(s) is the time of generating coupling matrix. m_g(Mb) is the memory needed to construct the $\mathcal{H}^2$-matrix using NPSA, m_s(Mb) the memory needed to store the resultant $\mathcal{H}^2$-matrix, ϵ_{add} and ϵ_{all} are the relative error of admissible block of $\mathcal{H}^2$-matrix and whole $\mathcal{H}^2$-matrix, compared to MoM matrix, respectively. ϵ_{add} and ϵ_{all} will take too long to get for $N = 1179648$ and thus are left empty. max(k^t) is the maximum of the new rank of the constructed $\mathcal{H}^2$-matrix.

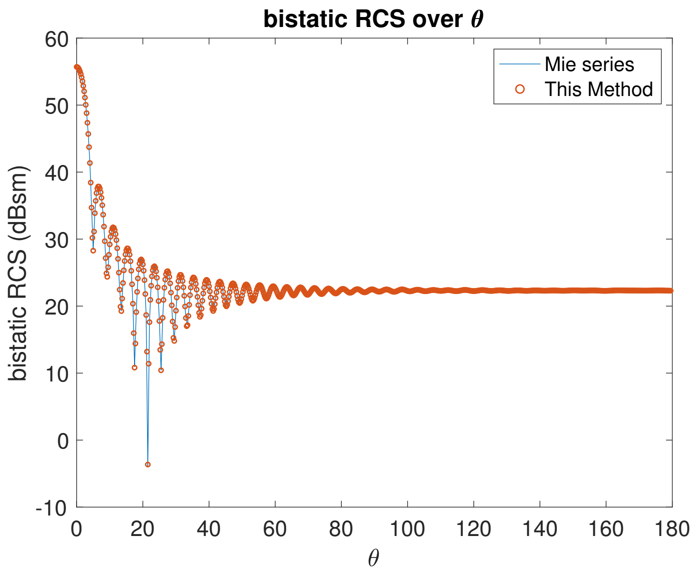


Figure 6.5. bRCS, NPSA, N=294912, Sphere

Table 6.2. Data of NPSA, sphere, $\epsilon = 10^{-3}$

N	4608	18432	73728	294912	1179648
t(s)	1.166e4	6.978e4	3.022e5	1.271e6	5.210e6
t_{cb} (s)	1.585e3	7.156e3	2.957e4	1.234e5	5.382e5
t_{cm} (s)	1.008e4	6.263e4	2.726e5	1.155e6	4.675e6
m_g (Mb)	118.23	520.01	2438.08	11433.72	58027.96
m_s (Mb)	84.09	397.23	1854.26	8707.85	43393.26
ϵ_{add}	3.575e-3	5.101e-3	6.604e-3	8.700e-3	
ϵ_{all}	1.743e-4	3.144e-4	4.716e-4	6.910e-4	
$\max(k^t)$	24	42	84	195	516

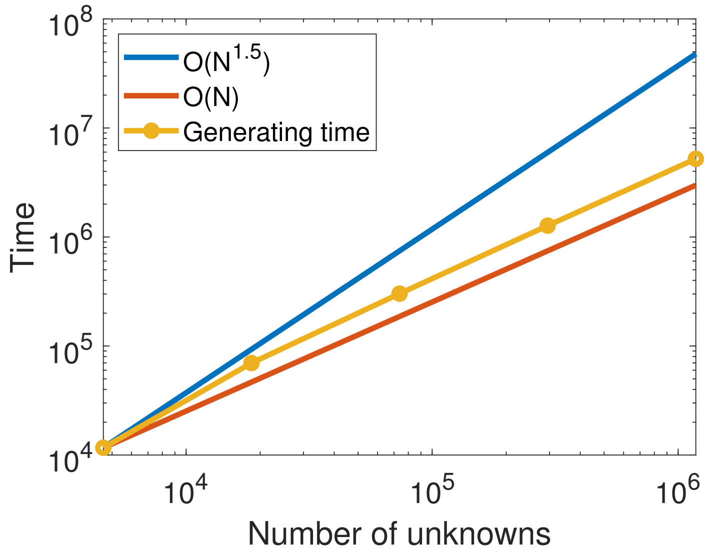


Figure 6.6. Time Complexity of Nested Pseudo Skeleton Approximation.

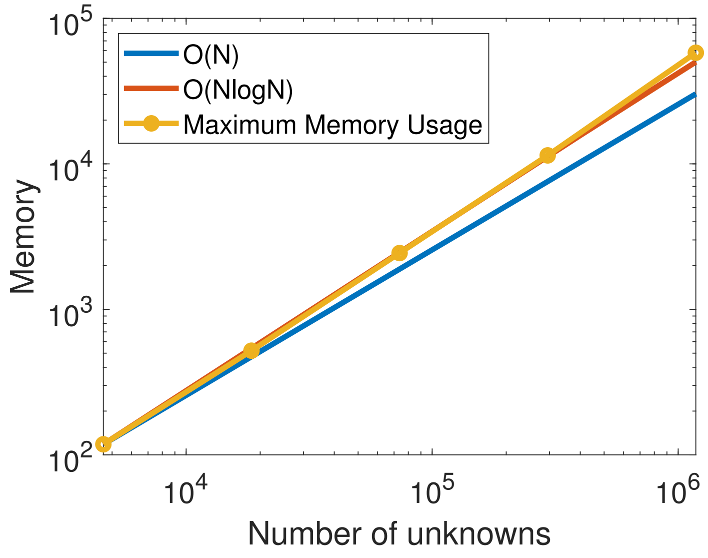


Figure 6.7. Memory Complexity of Nested Pseudo Skeleton Approximation.

We plot the time and memory usage in log scale in Fig. 6.6 and Fig. 6.7 for the minimal-rank $\mathcal{H}^2$-generation time. They are shown to agree very well with our theoretical complexity analysis.

We also generated another suite of data for different sizes of spheres using the algorithm in subsection 6.1.3, the results are shown in Table 6.3. We can see that the absolute run time of using this method is much less than that shown in Table 6.2.

Table 6.3. Data of NPSA, shown in subsection 6.1.3, Sphere, $\epsilon = 10^{-3}$

N	4608	18432	73728	294912
t(s)	585.23	4478.50	22531.84	120066.85
t_{cb}(s)	440.32	2780.39	11750.16	53218.94
t_{cm}(s)	144.90	1698.08	10781.57	66847.46
m_g(Mb)	754.06	4147.12	22531.84	170146.46
e_{all}	3.08e-3	4.02e-3	5.08e-3	6.795e-3
e_{ad}	1.97e-4	3.24e-4	4.74e-4	7.080e-4
$\max(k^t)$	204	384	796	1900

6.3 Conclusion

We present a new algebraic and kernel independent method, Nested Pseudo-Skeleton Approximation, to generate a minimal-rank $\mathcal{H}^2$-matrix to represent electrically large surface integral operators. Instead of approximating each admissible block, we find the low-rank approximation of the interaction between a cluster and its far-field. We also employ PSA with a random choice of pivots to reduce the computational cost of generating the low-rank approximation. We then develop an efficient nested algorithm to construct nested cluster bases level by level. After the cluster bases are built, we generate coupling matrices. The accuracy and complexity of the proposed are demonstrated by extensive numerical experiments. Compared to previous work, our new algorithm scales $O(k^3)$ for each cluster and $O(k^2)$ each coupling matrix and only uses $O(N \log N)$ elements of the original matrix.

7. ANALYTICAL SKELETON APPROXIMATION

In this chapter, we present a new method to approximate electrically large Green's function by a reduced set of parameters. Using this new method, we can compress electrically large integral operators in a simple and effective manner. The compression has a reduced complexity of $O(kn)$ for a dense matrix of size n and rank k, as compared to existing techniques that can cost $O(n^3)$ or $O(k^2n)$. Meanwhile, the accuracy is theoretically guaranteed. Numerical experiments have demonstrated its accuracy and effectiveness. Thus, this new method can be readily applied to the algorithms in previous algebraic methods to improve the accuracy and robustness.

7.1 Introduction

The low-rank representation of electrically large integral operators (IEs) is of great importance to the efficient solution of integral equations. For a dense matrix of size n and rank k, a brute-force SVD would cost $O(n^3)$ to find its low-rank approximation. Using a full cross approximation costs $O(kn^2)$, and employing a fast adaptive cross approximation (ACA) costs $O(k^2n)$ [19]. Interpolation based methods result in a full-rank matrix to represent electrically large IEs. Randomized approaches save time, but may lack accuracy control.

Green's Function is the kernel function of various IE formulations, including surface IEs (SIEs), volume IEs (VIEs), electric field IEs (EFIEs), magnetic field IEs, etc. In a fast multipole method (FMM), based on the addition theorem, Green's function is decomposed into a number of plane waves whose directions are distributed over a unit sphere.

The resultant rank scales quadratically with electrical size, which leads to a full-rank representation in a surface-IE based analysis. However, when performing an SVD on the dense matrix characterizing the Green's function based interaction between separated sources and observers, the rank is found to be much smaller than the matrix dimension for a prescribed accuracy. To understand this discrepancy, it is necessary to study whether there exist other representations of Green's function which are more compact than the addition theorem based approximation.

In this work, we begin with a new approach to approximate Green's function, whose rank has a good correlation with SVD's rank, which is the minimal rank required by accuracy. This approach is simple, fast, and also accurate. We then show how to apply it to various IE operators to efficiently obtain a low-rank representation of $O(kn)$ cost with theoretically guaranteed accuracy.

7.2 Proposed Work

To pursue more efficient representations of Green's function, we utilize the following cross approximation theorem [16], [17]: For a matrix $\mathbf{R}$ whose rank is k for accuracy ϵ, there exist k rows and k columns of the matrix which determine a cross approximation of the matrix within error tolerance as follows:

$$\tilde{\mathbf{R}} = \mathbf{R}(:, \hat{J}) \left(\mathbf{R}(\hat{I}, \hat{J})\right)^{-1} \mathbf{R}(\hat{I}, :) \tag{7.1}$$

and $\|\mathbf{R} - \tilde{\mathbf{R}}\|$ is within error tolerance. In (7.1), $\mathbf{R}(:, \hat{J})$ denotes the k selected columns of $\mathbf{R}$, whose indexes belong to $\hat{J} = \{\mathrm{j}_1, \mathrm{j}_2, ..., \mathrm{j}_k\}$, $\mathbf{R}(\hat{I}, :)$ denotes $\mathbf{R}$'s selected rows whose indexes are in the set of $\hat{I} = \{\mathrm{i}_1, \mathrm{i}_2, ..., \mathrm{i}_k\}$, and $\mathbf{R}^{-1}_{\hat{I},\hat{J}}$ is the inverse of the submatrix in $\mathbf{R}$ formed between the k rows and k columns. The computational cost of (7.1) lies in the determination of the row and column pivots. To find them, a full cross approximation costs $O(k \times m \times n)$, where m and n are the row, and column dimension of $\mathbf{R}$ respectively, while an adaptive cross approximation (ACA) costs $O(k^2(m + n))$. If one randomly chooses row and column pivots, the computational cost is low. However, the accuracy cannot be guaranteed.

Using the aforementioned theorem (also known as skeleton approximation), there exists the following representation of Green's function for the interaction between m observers in set t, and n sources in set s, if they are separated,

$$\tilde{\mathbf{G}}_{t,s} = \mathbf{G}(t, \hat{J})_{m\times k} \left(\mathbf{G}(\hat{I}, \hat{J})\right)^{-1}_{k\times k} \mathbf{G}(\hat{I}, s)_{k\times n} \tag{7.2}$$

in which k is the rank for a prescribed accuracy, $\mathbf{G}(t, \hat{J})$ denotes the k selected columns of dense matrix $\mathbf{G}$, whose indexes belong to $\hat{J} = \{\mathrm{j}_1, \mathrm{j}_2, ..., \mathrm{j}_k\}$, and having entries of $\frac{e^{-\mathrm{j}k\left|\mathbf{r}_\mathrm{i} - \mathbf{r}'_{\mathrm{j}_\nu}\right|}}{\left|\mathbf{r}_\mathrm{i} - \mathbf{r}'_{\mathrm{j}_\nu}\right|}$

$(\mathrm{i} = 1, 2, ..., m; \nu = 1, 2, ..., k)$. $\mathbf{G}(\hat{I}, :)$ denotes $\mathbf{G}$'s selected rows whose indexes are in the set of $\hat{I} = \{\mathrm{i}_1, \mathrm{i}_2, ..., \mathrm{i}_k\}$, and $(\mathbf{G}(\hat{I}, \hat{J}))^{-1}$ is the inverse of the submatrix in $\mathbf{G}$ formed between the k rows and k columns, having entries of

$$\mathbf{G}(\hat{I}, \hat{J}) = \frac{\mathrm{e}^{-\mathrm{j}k\left|\mathbf{r}_{\mathrm{i}_\gamma} - \mathbf{r}'_{\mathrm{j}_\nu}\right|}}{\left|\mathbf{r}_{\mathrm{i}_\gamma} - \mathbf{r}'_{\mathrm{j}_\nu}\right|} \quad \text{for } (\gamma, \nu = 1, 2, ..., k). \tag{7.3}$$

A representation like (7.2) is equivalent to approximating Green's function in the following way:

$$\frac{\mathrm{e}^{-\mathrm{j}k_0|\mathbf{r}-\mathbf{r}'|}}{4\pi|\mathbf{r}-\mathbf{r}'|} \approx \sum_{\nu=1}^{k} G(\mathbf{r}, \mathbf{r}'_{\mathrm{j}_\nu}) \sum_{\gamma=1}^{k} G^\dagger(\mathbf{r}_{\mathrm{i}_\gamma}, \mathbf{r}'_{\mathrm{j}_\nu}) G(\mathbf{r}_{\mathrm{i}_\gamma}, \mathbf{r}'), \tag{7.4}$$

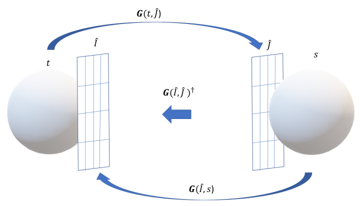
where $G^\dagger$ is the inverse of the reduced Green's function matrix in (7.2). The above means the interaction between separated sources and observers can be built by letting these sources at $\mathbf{r}'$ first radiate to the k skeleton points located at $\mathbf{r}_{\mathrm{i}_\gamma}$ $(\gamma = 1, 2, ..., k)$ in the observer domain, and then reciprocally, the k observers generate fields in the source domain at the k skeleton points located at $\mathbf{r}'_{\mathrm{j}_\nu}$ $(\nu = 1, 2, ..., k)$. After that, the k sources at $\mathbf{r}'_{\mathrm{j}_\nu}$ radiate to the observer points $\mathbf{r}$ to complete the connection. In this way, the original square interaction can be characterized by a reduced number of interactions. This is very different from an FMM-based decomposition of Green's function, where the sources first radiate to the center of the source group, which is then translated to the center of the observer group. After that, the field is emanated from the center of the observer group to the observer points. The latter (FMM) has a full rank in an electrically large SIE, while the former has a low rank similar to the rank of SVD. The major computation of (7.2) and (7.4) lies in the determination of the row and column pivots $\hat{I}$, and $\hat{J}$. Existing techniques cost at least $O(k^2(m+n))$ for prescribed accuracy. Here, we present an accurate and efficient way to obtain (7.2). It analytically determines $\hat{I}$ and $\hat{J}$, and hence having no computational cost in finding desired rows and columns (or deciding skeleton points); while generating the k columns and rows costs $O(k(m+n))$ only.

Our method is as follows. The center block of (7.2) actually represents an interaction between sources and observers using reduced degrees of freedom. As shown in [9], computing the SVD of the Green's function matrix is equivalent to performing a Fourier transform of the Green's function. The latter is found to be $1/(\kappa^2 - \kappa_0^2)$, where κ is the wave number of a Fourier mode, and κ_0 is the wave number of the Green's function. Clearly, Fourier modes having κ far away from κ_0 can be truncated without affecting desired accuracy. This means when a source and an observer domain are separated, a coarse discretization can be used to characterize the interaction between the two domains, since this would not change the Green's function's Fourier transform but to cut its fast decaying tail in the κ domain. Furthermore, this does not mean we do not discretize the original spatial domain using the right mesh size for the given frequency. As can be seen from (7.2), the row and column dimension of $\mathbf{G}$ are kept the same. It is the center block which captures coupling now has a reduced sampling. Such a representation also agrees with our physical intuition. When two regions are far from each other, their detailed structures are blurred to each other. In light of this fact, the only thing we need to do is to use a coarse resolution to select samples in the source domain, the set of which makes column pivots $\hat{J}$, and do the same in the observer domain to obtain $\hat{I}$. The coarse resolution can be as coarse as half wavelength, and even larger, which can be adaptively decided during the construction.

Moreover, if the uniform down sampling is troublesome to achieve in an irregularly shaped structure, we can enclose the source domain in an auxiliary bounding box, and do the same for the observer domain. The shape of the box can be as simple as a rectangular box. Then we select points uniformly based on a coarse resolution on the surface of the bounding box. If the center block of (7.2) is rank deficient, a pseudo-inverse can be performed using SVD (this is operated on a small matrix of rank size) and discarding singular values smaller than threshold. Due to the low rank property, the density of auxiliary points on the surface of the auxiliary box can be chosen much smaller than the that of the source/observer cluster itself. This density can also be adaptively determined during numerical computation. Basically, we start from an initial density that is very coarse. We then increase it step by step until the rank of the resultant block has saturated for a prescribed accuracy. In this process, the rank is small at the beginning, and then increases with the density. However, the rank would not increase any more after the density has increased to certain level.

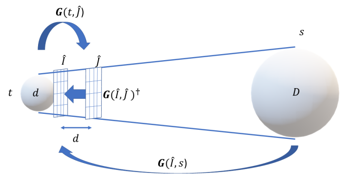
When the diameter of t and s are different,let Δ_{Bt} denote the distance between auxiliary points on the auxiliary box of t, while Δ_{Bs} denotes the same on the auxiliary box of s. Since the rank of the interaction between two clusters is determined by the electrical size of the smaller one, the Δ_{Bt} and Δ_{Bs} are determined by the diameter of the smaller cluster. Hence, the density on the larger auxiliary surface is smaller than that on the smaller auxiliary surface.

Besides the auxiliary box, we can also use auxiliary plates to act as $\hat{I}$ and $\hat{J}$. As shown in Fig. 7.1, the two auxiliary plates are placed in between the two clusters to block the directions along which the interaction occurs between t and s. Similar to the auxiliary box, the $\hat{I}$ and $\hat{J}$ on the auxiliary plates can be chosen sparsely and uniformly. The distance between auxiliary points on the auxiliary plates can also be determined adpatively until the resultant tank saturates. In addition, for better accuracy, the plate needs to be enlarged to cover more directions than those predicted by a straight ray tracing. This also agrees with the physics of waves at electrodynamic frequencies. The plates can also be arranged to be parallel with each other. If the two plates are made the same, we can further explore the nice structure of Toeplitz matrix to accelerate the computation. If t and s are different in size, we can move the plate of the larger cluster toward the smaller one, as shown in Fig. 7.2, the Toeplitz structure of $\mathbf{G}(\hat{I}, \hat{J})$ will still be maintained.



$\boldsymbol{G}(t,\hat{J})$
$\hat{I}$
$\hat{J}$
t
s
$\boldsymbol{G}(\hat{I},\hat{J})^{\dagger}$
$\boldsymbol{G}(\hat{I},s)$

Figure 7.1. Using auxiliary plates to build the ASA.



$\boldsymbol{G}(t,\hat{J})$
s
$\hat{I}$
$\hat{J}$
t
d
$\boldsymbol{G}(\hat{I},\hat{J})^{\dagger}$
D
d
$\boldsymbol{G}(\hat{I},s)$

Figure 7.2. Use auxiliary plates to build the ASA for rectangular matrices.

7.3 Application to IE operators

The aforementioned representation of Green's function can be readily applied to compress various IE operators in EM analysis. Take ϕ part of the EFIE as an example,

$$\mathbf{Z}_{\phi,mn} = \frac{1}{k^2}\int_{S_m}\int_{S_n}(\nabla_s \cdot \vec{f_m}(r)\nabla_s \cdot \vec{f_n}(r'))G(r,r')dS'dS, \tag{7.5}$$

where $\vec{f_m}$ and $\vec{f_n}$ are RWG bases on triangular patches S_m, and S_n respectively. If $m \in t$, $n \in s$, and t and s are separated, we can readily obtain its low-rank compression as

$$\mathbf{Z}_{\phi,mn} = \mathbf{V}_t\mathbf{G}^{\dagger}\mathbf{V}_s^T, \tag{7.6}$$

where

$$\mathbf{V}_t(:,\nu) = \frac{1}{k_0}\int_{S_m}(\nabla_s \cdot \vec{f_m})G(\mathbf{r},\mathbf{r}'_{\mathrm{j}_\nu})dS, \tag{7.7}$$

$$\mathbf{V}_s^T(\gamma,:) = \frac{1}{k_0}\int_{S_n}(\nabla_s \cdot \vec{f_n})G(\mathbf{r}_{\mathrm{i}_\gamma},\mathbf{r}')dS', \tag{7.8}$$

and $\mathbf{G}^{\dagger} = (\mathbf{G}(\mathbf{r}_{\mathrm{i}_\gamma},\mathbf{r}'_{\mathrm{j}_\nu}))^{-1}$ is the pseudo-inverse of the center Green's function matrix, and $\nu,\gamma = 1,2,...,k$. The application of (7.2) to vector potential $\mathbf{Z}_{A\xi,mn}, \xi \in \{x,y,z\}$ is similar, as follow:

$$\mathbf{Z}_{A\xi,mn} = \mathbf{V}_{\xi,m}\mathbf{G}^{\dagger}\mathbf{V}_{\xi,n}^T, \tag{7.9}$$

where $v_{\xi,m}$, $w_{\xi,n}$ are two vectors and are given as follow

$$\mathbf{V}_{\xi,m}(:,\nu) = \int_{S_m}(\vec{f_m})_\xi G(\mathbf{r},\mathbf{r}_{\mathrm{j}_\nu})dS, \tag{7.10}$$

where $(\vec{f_m})_\xi$ is the ξ component for $\vec{f_m}$, and

$$\mathbf{V}_{\xi,n}(:,\gamma) = \int_{S_n}(\vec{f_n})_\xi G(\mathbf{r}_{\mathrm{i}_\gamma},\mathbf{r}')dS', \tag{7.11}$$

where $(\vec{f_n})_\xi$ is the ξ component for $\vec{f_n}$, Thus, $\boldsymbol{Z}$ can obtained by

$$\mathbf{Z} = -\mathbf{Z}_\phi + \mathbf{Z}_{Ax} + \mathbf{Z}_{Ay} + \mathbf{Z}_{Az}. \tag{7.12}$$

7.4 Numerical Results

In this section, we examine the accuracy and efficiency of the proposed algorithm. We also apply the algorithm to generate an $\mathcal{H}$-matrix representation of the S-EFIE equation, and compare its performance with an ACA-based compression method.

7.4.1 Using Auxiliary Boxes

The first example is to represent the Green's function matrix formed between two solid cubes, each of which is discretized into a 3-D uniform grid. The distance between adjacent points is $\frac{\lambda}{10}$ along each of the x-, y- and z-directions. We choose such an example because it is universal in the sense that an arbitrarily shaped object can be interpolated from such a discretization. We use an auxiliary box to enclose each cube, and generate a low-rank representation accordingly.

The frequency considered is 300 MHz. Let d be the diameter of each cube, and $D = 1.2d$ be the distance between the center of the two cubes. For a range of d from small to large, we use the proposed method to obtain the low-rank representation and examine its accuracy. The results are shown in Table 7.1. In this Table, the n denotes the number of points in each grid. Thus, the Green's function matrix, $\boldsymbol{G}$, is of size $n \times n$. The s denotes the side length of each solid cube, and s_B is the side length of the auxiliary bounding box. The Δ_B is the distance between adjacent points on the surface of the auxiliary bounding box, and the n_B is the resultant number of auxiliary points on the bounding box. Hence, the $n_B \times n_B$ is the size of center matrix $\mathbf{G}(\hat{I}, \hat{J})$ shown in (7.2). The r_1 is the rank of $\mathbf{G}(\hat{I}, \hat{J})$ given the criterion of $\epsilon_1 = 10^{-10}$ when doing the pseudo inverse in the ASA. The matrix error is measured by

$$\mathrm{e}_1 = \frac{\|\boldsymbol{G} - \tilde{\boldsymbol{G}}\|_F}{\|\boldsymbol{G}\|_F}, \tag{7.13}$$

where $\|\cdot\|_F$ denotes the Frobenius norm, and $\tilde{\boldsymbol{G}}$ is the matrix obtained from the proposed ASA. The r_2 is the rank of $\mathbf{G}(t, \hat{J})$ given the criterion of $\epsilon_2 = 10^{-15}$. The error e_2 is measured by

$$\mathrm{e}_2 = \frac{\|\boldsymbol{G} - uu^H\boldsymbol{G}\|_F}{\|\boldsymbol{G}\|_F}, \tag{7.14}$$

where u is left singular vectors of $\mathbf{G}(t, \hat{J})$, obtained from a reduced SVD with ϵ_2 accuracy. As can be seen, the proposed method produces an accurate low-rank representation for all the d cases.

Table 7.1. ASA with auxiliary bounding boxes for the interaction between two solid cubes.

d (m)	0.693	1.558	2.424	3.289	4.155
D (m)	0.831	1.870	2.909	3.947	4.986
n	125	1000	3375	8000	15625
s (m)	0.400	0.900	1.399	1.899	2.399
s_B (m)	0.400	0.900	1.496	2.179	2.448
Δ_B (m)	0.200	0.300	0.374	0.436	0.490
n_B	26	56	98	152	152
r_1	26	56	98	138	137
e_1	3.215e-04	2.274e-05	3.842e-06	9.012e-07	5.483e-06
r_2	26	56	98	152	152
e_2	1.924e-04	1.807e-05	2.757e-06	9.156e-08	2.719e-07

Relationship between Δ_B and d for a fixed accuracy

Next, we examine how the Δ_B changes with respect to the diameter d for a fixed accuracy e_1. The results are listed in Table 7.2, where e_1 is fixed to be 10^{-4} for a variety of d. In Table 7.3, we list the same for $\mathrm{e}_1 = 10^{-6}$. It can be observed that when d is larger, the allowed cell size Δ_B for the same accuracy also increases but will saturate.

Cases with $d_t \neq d_s$

In Table 7.4, we examine those cases when $d_t \neq d_s$. In this Table, d_1 is the diameter of the first cube which changes, and $d_2 = 6.752$ m is fixed, with $D = 8.102$ m. The n_1 is the number of grid points on the first cube, and n_2 is that on the second cube. The $n_2 = 64000$ is

Table 7.2. ASA with auxiliary bounding boxes for two solid cubes: Effects of d on Δ_B, $\mathrm{e}_1 = 10^{-4}$.

d (m)	0.693	1.558	2.424	3.289	4.155
D (m)	0.831	1.870	2.909	3.947	4.986
n	125	1000	3375	8000	15625
s (m)	0.400	0.900	1.399	1.899	2.399
c	1.184	0.763	0.801	0.689	0.815
s_B (m)	0.506	1.180	1.400	2.531	2.403
Δ_B (m)	0.169	0.393	0.467	0.633	0.601
n_B	56	56	56	98	98
r_1	56	56	56	98	98
e_1	9.185e-05	1.006e-04	1.721e-04	5.964e-05	1.702e-04
r_2	56	56	56	98	98
e_2	6.867e-06	2.440e-05	1.420e-04	2.639e-05	9.080e-05

Table 7.3. ASA with auxiliary bounding boxes for two 3D uniform grids: Effects of d on Δ_B, $\mathrm{e}_1 = 10^{-6}$.

d (m)	0.693	1.558	2.424	3.289	4.155
D (m)	0.831	1.870	2.909	3.947	4.986
n	125	1000	3375	8000	15625
s (m)	0.400	0.900	1.399	1.899	2.399
c	1.719	1.156	1.068	0.988	1.019
s_B (m)	0.465	1.037	1.400	2.206	2.402
Δ_B (m)	0.116	0.259	0.350	0.441	0.480
n_B	98	98	98	152	152
r_1	83	91	92	138	135
e_1	9.485e-07	1.036e-06	3.602e-06	9.485e-07	5.640e-06
r_2	98	98	98	152	152
e_2	2.068e-09	2.515e-07	2.741e-06	9.637e-08	2.787e-07

fixed, while the n_1 changes. The s_1 and s_2 are the side length of the two cubes, respectively. The s_{B1} and s_{B2} are the side length of the two auxiliary boxes, and Δ_{B1} and Δ_{B2} are the cell size on the surface of the two auxiliary boxes. The n_{B1} and n_{B2} are the resultant number of grid points on the auxiliary box surfaces. Other conditions are set to be the same as those in the previous case. From the last column of Table 7.4, it can be seen that a $\mathbf{G}(\hat{I}, \hat{J})$ of size 152×152 can represent the original $\boldsymbol{G}$ of size $15,625 \times 15,625$, while achieving a good accuracy of 5.483e-06. From this table, we can also see that Δ_B for the larger auxiliary box is affected by the diameter of the smaller one, and $n_{Bt} = n_{Bs}$ can hold true even though the two clusters are very different in size.

Table 7.4. ASA with auxiliary bounding boxes for two solid cubes: $d_t \neq d_s$ case.

d_1	0.693	1.558	2.424	3.289
n_1	125	1000	3375	8000
s_1	0.400	0.900	1.399	1.899
s_2	3.898	3.898	3.898	3.898
s_{B1}	0.400	0.900	1.496	2.179
s_{B2}	3.898	3.898	4.167	4.472
Δ_{B1}	0.200	0.300	0.374	0.436
Δ_{B2}	1.949	1.299	1.042	0.894
n_{B1}	26	56	98	152
n_{B2}	26	56	98	152
r_1	26	54	86	128
e_1	2.709e-05	5.286e-06	9.444e-07	2.511e-07
r_2	26	56	98	152
e_2	2.696e-05	4.940e-06	8.071e-07	1.158e-08

Choice of Δ_B

We choose $\Delta_B = \frac{1}{c}\sqrt{d\Delta_N}$, with $\Delta_N = \lambda/10$, and test the effect of enlarging coefficient c in Table 7.5. In this table, the two cubes have the same $d = 1.558$ m, $n = 1000$, and $s = 0.900m$. In addition, $D = 1.870m$. The effect of c is also recorded in Table 7.6, in which $d = 4.155m$, $D = 4.986m$, $n = 15625$, $s = 2.399m$, and $\epsilon_1 = 10^{-10}$, $\epsilon_2 = 10^{-15}$. This table shows we can adaptively decide c, and hence Δ_B based on the convergence of rank r_1, which can be obtained in a relatively low cost.

Table 7.5. ASA with auxiliary bounding boxes for two solid cubes: Choice of Δ_B ($d = 1.558$ m).

c	s_B(m)	Δ_B(m)	n_B	r_1	e_1	r_2	e_2
1	0.900	0.300	56	56	2.274e-05	56	1.807e-05
1.5	1.000	0.200	152	113	1.180e-07	152	3.190e-09
2	0.900	0.150	218	115	2.169e-07	192	4.001e-11

Table 7.6. ASA with auxiliary bounding boxes for two solid cubes: Choice of Δ_B ($d = 4.155m$).

c	s_B(m)	Δ_B(m)	n_B	r_1	e$_1$	r_2	e$_2$
0.2	2.448	2.448	8	8	1.718	8	2.947e-01
0.3	3.264	1.632	26	26	4.193e-01	26	3.286e-02
0.5	2.938	0.979	56	56	3.939e-03	56	1.820e-03
1	2.448	0.490	152	152	5.483e-06	152	2.719e-07
1.5	2.612	0.326	386	202	1.537e-07	306	1.707e-12
2	2.448	0.245	602	193	1.266e-07	333	9.532e-15

Effects of side length s_B

In Table 7.7, we show the effect of side-length s_B on the accuracy of ASA. In this table, $c = 1$, $d = 3.289$ m, $D = 3.947$m, $n = 8000$, $s = 1.899$, $\Delta_B = 0.436$, $\epsilon_1 = 10^{-10}$, and $\epsilon_2 = 10^{-15}$. Table 7.8 also shows the effect of side length s_B on the accuracy of ASA, where $c = 0.5$, $d = 4.155$ m, $D = 4.986$ m, $n = 15625$, $s = 2.399$, $\Delta_B = 0.979$, $\epsilon_1 = 10^{-10}$, $\epsilon_2 = 10^{-15}$. In Table 7.9, we examine the same but D is changed to $D = 8.310$ m.

Table 7.7. ASA with auxiliary bounding boxes for two solid cubes: Effects of side length s_B ($d = 3.289$ m, $D = 3.947$m).

s_B	n_B	r_1	e$_1$	r_2	e$_2$
2.179	152	138	9.012e-07	152	9.156e-08
2.614	218	191	1.081e-07	218	3.937e-09
3.050	296	248	1.114e-07	291	1.109e-10
3.486	386	317	1.937e-07	360	4.141e-12

Table 7.8. ASA with auxiliary bounding boxes for two solid cubes: Effects of side length s_B ($d = 4.155$ m, $D = 4.986$ m).

s_B	n_B	r_1	e$_1$	r_2	e$_2$
2.938	56	56	3.939e-03	56	1.820e-03
3.917	98	98	1.727e-03	98	1.720e-04

Table 7.9. ASA with auxiliary bounding boxes for two solid cubes: Effects of side length s_B ($d = 4.155$ m, $D = 8.310$ m).

s_B	n_B	r_1	e_1	r_2	e_2
2.938	56	56	1.009e-04	56	6.113e-05
3.917	98	98	2.455e-05	98	1.753e-06
4.897	152	152	1.596e-05	152	8.067e-09
5.876	218	218	2.072e-05	218	1.821e-10

Effects of η or distance D

In Table 7.10, we show the effects of η on the accuracy of ASA. In this table, we fix $d = 4.155$ m while changing D, and see how Δ_B should be selected to maintain a reasonable accuracy of e_1 or e_2. In this table, $c = 1$, $n = 15625$, $s = 2.399$, $\epsilon_1 = 10^{-10}$, $\epsilon_2 = 10^{-15}$ and we try to fix e_1 to be 10^{-4}. In table 7.11, we try to fix e_1 to be 10^{-6}. We can see D play a role in determining Δ_B, and thus determining n_B, the approximate rank of the ASA. With D increasing, n_B clearly decreases.

Table 7.10. Auxiliary bounding boxes enclosing two solid cubes: Effects of D on Δ_B, $e_1 = 10^{-4}$.

η	1.2	2	5	10
D	4.986	8.310	20.775	41.550
c	0.697	0.503	0.322	0.228
Δ_B	0.706	0.973	1.521	2.147
s_B	2.823	2.920	3.043	4.293
n_B	98	56	26	26
r_1	98	56	26	26
e_1	1.513e-04	9.776e-05	1.075e-04	8.819e-05
r_2	98	56	26	26
e_2	9.166e-05	6.098e-05	6.541e-05	4.053e-06

Electrically large cases

Next we show in Table 7.12 that the proposed method can be used to compress electrically very large cases. We use the convergence of rank r_1 to adaptively decide Δ_B to ensure the accuracy of ASA. In this table, $\epsilon = 10^{-8}$ and $\eta = 2$. We increase enlarging coefficient

Table 7.11. Auxiliary bounding boxes enclosing two solid cubes, Effects of D on Δ_B, $\mathrm{e}_1 = 10^{-6}$.

η	2	4	15
D	8.310	16.620	62.325
c	0.769	0.484	0.271
Δ_B	0.637	1.011	1.806
s_B	2.548	3.033	3.612
n_B	98	56	26
r_1	98	56	26
e_1	1.002e-06	9.839e-07	9.522e-07
r_2	98	56	26
e_2	8.055e-07	7.670e-07	7.716e-07

Table 7.12. Auxiliary bounding boxes enclosing 3D solid cubes: Electrically very large cases.

d (m)	1.558	3.289	6.752	13.677	27.527	55.227	110.628	221.428
D (m)	3.116	6.579	13.504	27.354	55.054	110.455	221.255	442.857
n	1000	8000	6.4e4	5.12e5	4.096e6	3.277e7	2.621e8	2.097e9
s (m)	0.900	1.899	3.898	7.896	15.893	31.886	63.871	127.842
c	1.700	1.200	0.950	0.950	1.450	1.450	1.450	1.950
s_B (m)	1.058	2.179	3.942	8.417	16.515	32.011	64.474	128.322
Δ_B (m)	0.176	0.363	0.657	0.935	0.869	1.231	1.743	1.833
n_B	218	218	218	488	2168	4058	8216	29402
r_1	62	83	103	194	369	734	1815	4943
e_1	1.142e-9	2.874e-9	2.116e-8	3.902e-8	7.483e-9	1.544e-8	3.835e-8	1.916e-8

c each time by 0.25 starting from 0.2 and stop once r_1 stops increasing with c. Then we measure e_1. For $d \geq 13.677$, we randomly choose a submatrix of size 50000×50000 from the original MoM matrix to measure e_1 to test the accuracy of ASA. This example also shows the proposed compression method is efficient, and can handle very large electrical sizes.

7.4.2 Using Original Pivots

In Table 7.13, we test the ASA without using an auxiliary box, but using the original columns and rows of $\boldsymbol{Z}$, which is from an EFIE matrix. Here we use two spheres to represent two admissible clusters, d for each sphere is 5.756 m, the center to center distance between two spheres D is set to be $D = 1.5d$, which is 8.633 m. Using RWG basis with a discretization of

$\lambda/10$, the original EFIE matrix has a size of 45000×45000. We extract uniformly n_B elements from each sphere, based on a different c and thereby Δ_B. The $\epsilon_1 = 10^{-10}$, $\epsilon_2 = 10^{-15}$. As can be seen, a good accuracy of ASA is achieved.

Table 7.13. ASA with original pivots applied to the EFIE of two conducting spheres.

c	d	D	n	n_B	e$_1$	e$_2$
0.5	5.756	8.633	45000	288	6.608e-04	2.557e-05
1	5.756	8.633	45000	1056	6.821e-08	5.869e-11
2	5.756	8.633	45000	5202	2.374e-08	2.533e-14

7.4.3 Using Auxiliary Plates

When using auxiliary plates, the choice of the side length of the plate is important. Thus, we need to introduce a new variable p, which is the number of padding along each direction of the plate. The padding has the same density as that on the auxiliary plate. In Table 7.14, $\epsilon_1 = 10^{-10}$, $d = 3.289$ m, $s = 1.899$ m, $D = 6.579$ m, $n = 8000$. From this table, we can see that increasing c and p can both improve e$_1$. However, increasing p is a better option than increasing c because it can better improve e$_1$ with the same n_B. In addition, the numerical rank of the resultant $\boldsymbol{G}$ under ϵ_1 is closer to its dimension.

Table 7.14. ASA with auxiliary plates for two solid cubes, $\eta = 1.2$.

c	p	Δ_B	n_B	s_B	r_1	e$_1$
1	1	0.436	36	2.178	36	4.780e-04
2	1	0.218	100	1.961	49	4.711e-05
3	1	0.145	225	2.033	50	1.774e-05
1	1	0.436	36	2.178	36	4.780e-04
1	3	0.436	64	3.050	62	4.775e-05
1	5	0.436	100	3.921	95	6.732e-06

Finally, we show when $d_t \neq d_s$, the plate of the larger cluster does not need to cling to the larger one. Fig. 7.3 illustrates the side view of the setting that is shown in Fig. 7.2, where t and s represent two solid cubes, P_t and P_s represent two auxiliary plates that cover all the directions of the interaction. The plate P_t is on the right side of t, whereas P_s is D_p away from P_t. The ASA results for this setting are shown in Table 7.15.

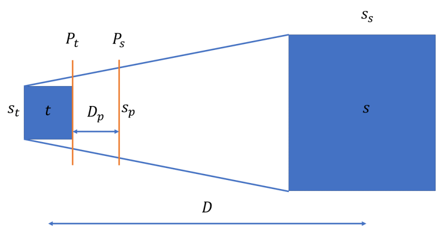


Figure 7.3. Use auxiliary plates with equal Δ_B

Table 7.15. ASA with auxiliary plates for clusters with a different size using equal Δ_B for two solid cubes, $\eta = 1.2$.

c	2	2	2
p	5	5	5
s_t	0.900	0.900	1.899
s_s	3.898	7.896	3.898
D	13.504	27.354	13.504
n_t	1000	1000	8000
n_s	64000	512000	64000
D_p	0.900	0.900	1.899
Δ_B	0.150	0.150	0.218
n_B	169	169	256
s_B	1.799	1.799	3.268
r_1	143	143	171
e_1	9.262e-3	1.160e-2	1.541e-3

7.4.4 Application to realistic S-EFIE problems

Next, we use the proposed method to construct an $\mathcal{H}$-matrix of the EFIE. For a sphere of diameter 7.37λ and of 73,728 unknowns with a $\frac{\lambda}{10}$ discretization. The MoM matrix is partitioned into an $\mathcal{H}$-matrix of 10 levels, with admissiblility condition $\max(d_t, d_s) < 0.6D$ and $leafsize = 40$. There are 7 tree levels that have admissible blocks. Each cluster of an admissible block is enclosed by an auxiliary box, and the matrix is approximated using the proposed ASA. The $c = 1$ is chosen. The whole matrix relative error compared with the original MoM matrix $\boldsymbol{Z}$ is shown to be 1.440e-05 (the relative error for the admissible part of $\boldsymbol{Z}$ is 2.176e-04). Under the same condition, for a sphere of diameter 14.741λ and of 294,912 unknowns, the $\mathcal{H}$-matrix has 12 levels, among which 9 levels have admissible blocks.

The whole matrix relative error compared with the original MoM matrix $\boldsymbol{Z}$ is found to be 1.479e-05 (relative error for admissible part of $\boldsymbol{Z}$ is 1.972e-04).

7.4.5 Comparison of ASA with ACA

Finally, we compare the proposed ASA with the ACA to demonstrate its efficiency. For a sphere with diameter 3.685λ with 1,432 unknowns, the $\mathcal{H}$-matrix has 8 levels, we use both the proposed ASA and the ACA to genertae the low-rank representation of each admissible block. The total time measured for ASA includes the time used for defining each auxiliary box, the generation of $\boldsymbol{G}_o$, $\boldsymbol{G}_s$ for four components of $\boldsymbol{G}$, the generation of $\boldsymbol{G}_{cnt}$ and pseudo-inverse of $\boldsymbol{G}_{cnt}$. Total time measured for ACA includes directly applying ACA to $\boldsymbol{Z}$ to generate $\boldsymbol{A}$ and $\boldsymbol{B}$. A suite of PEC spheres are simulated with $\frac{\lambda}{10}$ discretization in Table 7.16. The number of Gauss Quadrature points on each triangle patch is 3. As can be seen, the proposed method has advantages. It achieves a better accuracy while using less CPU run time. The comparsion is made for a sequential implementation. If run parallel, the proposed would outperform more since its parallelization is straightforward while ACA is difficult to be parallelized since it is an iteration-based algorithm.

Table 7.16. Comparison Between ASA and ACA.

d	0.92λ	3.69λ	7.37λ	14.741λ
N	4608	18432	73728	294912
$t_{ASA,full}$ (s)	2885.88	17680.60	104667.95	451577.60
$t_{ASA,ad}$ (s)	2726.41	17111.00	102221.79	441825.77
e_{ad}	1.873e-4	1.713e-4	1.592e-4	1.354e-4
e_{all}	7.226e-6	8.773e-6	9.678e-6	1.015e-5
$t_{ACA,ad}$ (s)	4589.22	38422.44	189526.73	916931.91
e_{ad}	1.044e-4	1.644e-4	1.316e-4	1.568e-4

7.5 Conclusion

We developed a new method to compress Green's Function. It is analytical, accurate, efficient, simple and straightforward. It helps intuitively understand why the Green's function has a low-rank representation irrespective of electrical size. The resultant rank has a good correlation with the minimal rank required by accuracy. Comparison with existing compression methods such as ACA has demonstrated the clear advantages of the proposed method. Such a compression technique can be readily applied to various IE equations to accelerate their solutions.

8. SUMMARY

8.1 Conclusions

In this work, we develop various methods to construct a low-rank or minimal-rank $\mathcal{H}^2$-matrix for electrically large SIE operators. These methods are efficient and accurate. The $\mathcal{H}^2$-matrix constructed by these methods can be used to accelerate both iterative and direct solvers. These methods can either leverage the low complexity of MLFMA or directly construct $\mathcal{H}^2$-matrix algebraically.

- First, the method described in [8] is improved when applying it to convert an FMM $\mathcal{H}^2$-matrix to a minimal rank $\mathcal{H}^2$-matrix. The key contribution in this method is to process and store those large matrices in their factorized form. At a lower level, we factor those matrices in their factorized forms. Then we use the factorized form and the nested property of $\mathcal{H}^2$-matrix to directly compute the factorized form of a large matrix at the upper level. By doing this in every step of the algorithm in [8], we can reduce both the time and memory complexity.

- A NRA is developed to convert an FMM $\mathcal{H}^2$-matrix to a minimal rank $\mathcal{H}^2$-matrix. In this new algorithm, we focus on the treatment of cluster basis $\mathbf{V}^t$ instead of $\mathbf{G}_2^t$. In this way, we can still do the converting process in low complexity. But this new method can cost much less absolute time. We do SVD on $\mathbf{V}^t$ on leaf level. Then we use the factorized information of lower level $\mathbf{V}^t$ and the nested property of $\mathcal{H}^2$-matrix to compute the $\mathbf{T}^t$ in an upper level.

- Then we develop a few algorithms based on NRA that possess even lower complexity. The first one is Fast NRA that utilizes randomness. We can randomly select a few column pivots of $\tilde{\mathbf{G}}$ and do FCA. In this way, we do not have to do SVD on the large $\tilde{\mathbf{G}}$. We then develop DRA that performs another recursive procedure to compute new $\tilde{\mathbf{V}}$ and $\tilde{\mathbf{S}}$. Finally, we exploit the sparsity of $(\tilde{\mathbf{U}}^t)^H$ and $\tilde{\mathbf{G}}$ to reduce the time and memory complexity.

- Then we introduce an algebraic method to construct low-rank $\mathcal{H}^2$-matrix or minimal-rank $\mathcal{H}^2$-matrix, Nested Construction. It relies on Pseudo-Skeleton Approximation and nested property of $\mathcal{H}^2$-matrix. This method can achieve the same complexity as the aforementioned methods but can avoid the complicated implementation of MLFMA, and being purely algebraic.

- Then we develop a new kernel-free pure algebraic method to construct minimal-rank $\mathcal{H}^2$-matrix, Nested Pseudo-Skeleton Approximation. It improves on the previous algebraic method in that the time cost for each cluster and coupling matrix is only $O(k^3)$ while memory is $O(k^2)$. This leads to a total of $O(N^{1.5})$ time complexity and $O(N \log N)$ memory complexity for SIE.

- Finally, we improve the PSA, which is heavily relied on in our algebraic methods. We call this Analytical Skeleton Approximation. It can eliminate the randomness of PSA and retain the same complexity. It beats ACA in terms of time complexity while does not suffers from the potential failure of ACA when applying to off-diagonal blocks of the EFIE matrix. It is analytical, accurate, efficient, simple to implement, and straightforward.

8.2 Future work

- We will further reduce the complexity of constructing minimal-rank $\mathcal{H}^2$-matrix for electrically large SIE analysis by using ASA.

REFERENCES

[1] W. C. Chew, E. Michielssen, J. Song, and J.-M. Jin, *Fast and efficient algorithms in computational electromagnetics*. Artech House, Inc., 2001.

[2] E. Darve, "The fast multipole method: Numerical implementation," *Journal of Computational Physics*, vol. 160, no. 1, pp. 195–240, 2000.

[3] R. Coifman, V. Rokhlin, and S. Wandzura, "The fast multipole method for the wave equation: A pedestrian prescription," *IEEE Antennas and Propagation Magazine*, vol. 35, no. 3, pp. 7–12, 1993.

[4] J. M. Song and W. C. Chew, "Multilevel fast-multipole algorithm for solving combined field integral equations of electromagnetic scattering," *Microwave and Optical Technology Letters*, vol. 10, no. 1, pp. 14–19, 1995.

[5] 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, 1999.

[6] S. Börm, L. Grasedyck, and W. Hackbusch, "Hierarchical matrices," *Lecture notes*, vol. 21, 2003.

[7] M. Ma and D. Jiao, "Accuracy directly controlled fast direct solution of general $\mathcal{H}^2$-matrices and its application to solving electrodynamic volume integral equations," *IEEE Trans. Microwave Theory and Techniques*, vol. 66, no. 1, pp. 35–48, 2017.

[8] W. Chai and D. Jiao, "Linear-complexity direct and iterative integral equation solvers accelerated by a new rank-minimized $\mathcal{H}^2$-representation for large-scale 3-D interconnect extraction," *IEEE Trans. Microwave Theory and Techniques*, vol. 61, no. 8, pp. 2792–2805, 2013.

[9] W. Chai and D. Jiao, "Theoretical study on the rank of integral operators for broadband electromagnetic modeling from static to electrodynamic frequencies," *IEEE Trans. on Components, Packaging and Manufacturing Technology*, vol. 3, no. 12, pp. 2113–2126, 2013.

[10] C. Yang and D. Jiao, "Method for generating a minimal-rank $\mathcal{H}^2$-matrix from fmm for electrically large analysis," in *IEEE International Symp. on Antennas and Propagation*, IEEE, 2018, pp. 2503–2504.

[11] C. Yang, M. Ma, and D. Jiao, "Fast algorithms for converting an FMM-based representation of electrically large integral operators to a minimal-rank $\mathcal{H}^2$-matrix," in *IEEE International Symp. on Antennas and Propagation*, IEEE, 2019.

[12] Y. Zhao, D. Jiao, and J. Mao, "Fast nested cross approximation algorithm for solving large-scale electromagnetic problems," *IEEE Transactions on Microwave Theory and Techniques*, vol. 67, no. 8, pp. 3271–3283, 2019.

[13] K. Zhao, M. N. Vouvakis, and J.-F. Lee, "The adaptive cross approximation algorithm for accelerated method of moments computations of emc problems," *IEEE transactions on electromagnetic compatibility*, vol. 47, no. 4, pp. 763–773, 2005.

[14] D. Jiao and S. Omar, "Minimal-rank $\mathcal{H}^2$-matrix-based iterative and direct volume integral equation solvers for large-scale scattering analysis," in *2015 IEEE International Symp. Antennas and Propagation*, IEEE, 2015, pp. 740–741.

[15] C. Balanis, *Advanced Engineering Electromagnetics*, ser. CourseSmart Series. Wiley, 2012, ISBN: 9780470589489. [Online]. Available: https://books.google.com/books?id=cRkTuQAACAAJ.

[16] 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.

[17] A. Osinsky and N. L. Zamarashkin, "Pseudo-skeleton approximations with better accuracy estimates," *Linear Algebra and its Applications*, vol. 537, pp. 221–249, 2018.

[18] C. Yang and D. Jiao, "Nested reduction algorithms for generating a rank-minimized $\mathcal{H}^2$-matrix from FMM for electrically large analysis," *IEEE Trans. Antennas and Propagation*, vol. 69, no. 7, pp. 3945–3956, 2021. DOI: 10.1109/TAP.2020.3044584.

[19] M. Bebendorf, *Hierarchical matrices*. Springer, 2008.

VITA

Chang Yang received the B.S. degree in electronic science and technology from the Xi'an Jiaotong University, Xi'an, CHINA, in 2016. Since 2016, he has been working toward the Ph.D. degree at Purdue University, West Lafayette, IN, USA. He is with the School of Electrical and Computer Engineering, Purdue University, as a member of the On-Chip Electromagnetics Group. His current research interests include computational electromagnetics, fast and high-capacity numerical methods. He was the recipient of an Honorable Mention Award of the 2020 IEEE AP-S Symposium on Antennas and Propagation.