2019

# Fast, Sparse Matrix Factorization and Matrix Algebra via Random Sampling for Integral Equation Formulations in Electromagnetics

Owen Tanner Wilkerson
*University of Kentucky*, otwi222@uky.edu
Digital Object Identifier: https://doi.org/10.13023/etd.2020.006

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Owen Tanner Wilkerson, Student

Dr. Robert J. Adams, Major Professor

Dr. Aaron Cramer, Director of Graduate Studies

# FAST, SPARSE MATRIX FACTORIZATION AND MATRIX ALGEBRA VIA RANDOM SAMPLING FOR INTEGRAL EQUATION FORMULATIONS IN ELECTROMAGNETICS

---

THESIS

---

A thesis submitted in partial
fulfillment of the requirements for
the degree of Masters of Science in
Electrical Engineering in the College
of Engineering at the University of
Kentucky

By
Owen Tanner Wilkerson
Lexington, Kentucky

Co-Directors: Dr. Robert J. Adams, Professor of Electrical and Computer Engineering
and Dr. John Young, Professor of Electrical and Computer Engineering
Lexington, Kentucky

2019

ABSTRACT OF THESIS

# FAST, SPARSE MATRIX FACTORIZATION AND MATRIX ALGEBRA VIA RANDOM SAMPLING FOR INTEGRAL EQUATION FORMULATIONS IN ELECTROMAGNETICS

Many systems designed by electrical & computer engineers rely on electromagnetic (EM) signals to transmit, receive, and extract either information or energy. In many cases, these systems are large and complex. Their accurate, cost-effective design requires high-fidelity computer modeling of the underlying EM field/material interaction problem in order to find a design with acceptable system performance. This modeling is accomplished by projecting the governing Maxwell equations onto finite dimensional subspaces, which results in a large matrix equation representation ($Zx = b$) of the EM problem. In the case of integral equation-based formulations of EM problems, the $M$-by-$N$ system matrix, $Z$, is generally dense. For this reason, when treating large problems, it is necessary to use compression methods to store and manipulate $Z$. One such sparse representation is provided by so-called $H^2$ matrices. At low-to-moderate frequencies, $H^2$ matrices provide a controllably accurate data-sparse representation of $Z$.

The scale at which problems in EM are considered "large" is continuously being redefined to be larger. This growth of problem scale is not only happening in EM, but respectively across all other sub-fields of computational science as well. The pursuit of increasingly large problems is unwavering in all these sub-fields, and this drive has long outpaced the rate of advancements in processing and storage capabilities in computing. This has caused computational science communities to now face the computational limitations of standard linear algebraic methods that have been relied upon for decades to run quickly and efficiently on modern computing hardware. This common set of algorithms can only produce reliable results quickly and efficiently for small to mid-sized matrices that fit into the memory of the host computer. Therefore, the drive to pursue larger problems has even began to outpace the reasonable capabilities of these common numerical algorithms; the deterministic numerical linear algebra algorithms that have gotten matrix computation this far have proven to be inadequate for many problems of current interest. This has computational science communities focusing on improvements in their mathematical and software approaches in order to push further advancement. Randomized numerical linear al-

gebra (RandNLA) is an emerging area that both academia and industry believe to be strong candidates to assist in overcoming the limitations faced when solving massive and computationally expensive problems.

This thesis presents results of recent work that uses a random sampling method (RSM) to implement algebraic operations involving multiple $H^2$ matrices. Significantly, this work is done in a manner that is non-invasive to an existing $H^2$ code base for filling and factoring $H^2$ matrices. The work presented thus expands the existing code's capabilities with minimal impact on existing (and well-tested) applications. In addition to this work with randomized $H^2$ algebra, improvements in sparse factorization methods for the compressed $H^2$ data structure will also be presented. The reported developments in filling and factoring $H^2$ data structures assist in, and allow for, the further pursuit of large and complex problems in computational EM (CEM) within simulation code bases that utilize the $H^2$ data structure.

KEYWORDS: numerical simulations, randomized numerical linear algebra, computational electromagnetics, computational linear algebra

Author's signature: Owen Tanner Wilkerson

Date: December 20, 2019

# FAST, SPARSE MATRIX FACTORIZATION AND MATRIX ALGEBRA VIA RANDOM SAMPLING FOR INTEGRAL EQUATION FORMULATIONS IN ELECTROMAGNETICS

By
Owen Tanner Wilkerson

Dr. Robert J. Adams
Co-Director of Thesis

Dr. John Young
Co-Director of Thesis

Dr. Aaron Cramer
Director of Graduate Studies

December 20, 2019
Date

# ACKNOWLEDGMENTS

This thesis would have not been possible without the trust and support of my advisor, Dr. R. J. Adams. Dr. Adams trusted and assigned me to implement the group's start-of-the-art diagonal factorization into the main code base as just an undergraduate junior. Then, entering into my senior year, he encouraged me to embark on my own investigations which lead to the development of innovations like the Random Sampling Method detailed in this document. In my mind, these ventures made possible by Dr. Adams defined my higher education. His confidence in me is what gave me the very unique opportunity to solidify my own confidence in my passion fields of algorithm development and computational linear algebra. This confidence, that I am very fortunate to gain at such a young age by working under Dr. Adams, will undoubtedly have massive influence in molding my future. Thank you, Dr. Adams.

I also need to thank my family for always embracing my freedom to define my own path in life. My family has always exuded confidence, love, and support for me. This support has improved every facet of my life. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

## Chapter 1 Introduction

Randomized numerical linear algebra (RandNLA) is a relatively recent development in the field of linear algebra that has arisen in response to the need to quickly analyze large problems and/or data sets [1][2][3]. This is because RandNLA is able to overcome some of the challenges encountered in pursuing large and computationally expensive problems in various subfields of computational science. This document will discuss recent derivations and implementations of RandNLA based algebraic methods for the $H^2$ data structure. The $H^2$ representation of a matrix relies on a data structure that is similar to the fast multipole method's nested data representation.

$H^2$ matrices were developed by Wolfgang Hackbusch and are commonly used in computational physics applications to form data-sparse (compressed) representations of large, dense system matrices, $Z$. Hackbusch has presented developments in the algebra of $H$ and $H^2$ matrices utilizing deterministic linear algebra methods as recent as 2015 [4][5]. Related recent work is also available in [6]. These $H^2$ algebra approaches appear to be somewhat complicated and, while it may be possible, it is unclear how easily these approaches can be incorporated into an existing code base for filling an $H^2$ representation of $Z$. In this thesis, we pursue a similar functionality for $H^2$ matrix algebra using methods from the field of RandLNA. In particular, the approach pursued here requires only the ability to perform multiplication of an $H^2$ matrix (or its transpose) against a set of randomly generated vectors.

The RandNLA based $H^2$ methods discussed herein are capable of performing $H^2$ algebra in a manner that mimics the existing input/output argument paradigm of ACA-based fill methods [7][8] that are commonly used to fill the $H^2$ representation from the original system $Z$ [9]. This mimicry of the ACA fill's input/output argument paradigm enables a noninvasive incorporation of $H^2$ algebraic methods within a code base already capable of performing $H^2$ matrix fills using ACA. Throughout this document, the new non-invasive, $H^2$ algebra enabling fill method will be referred to as the Random Sampling Method (RSM). It is important to understand that the RSM does not replace an ACA fill approach as the RSM can not be used to efficiently fill an $H^2$ from scratch by sparsely sampling $Z$; ACA-type methods are still required for this operation. RSM methods provide a complementary fill method that enables algebra to be performed on already existing $H^2$ matrices.

Computational linear algebra for the solution of partial differential equations can be sometimes described, in a trivialized manner, in 3 fundamental steps: matrix fill (compression), matrix factorization (only if utilizing a direct solver), and solution (generally iterative [10] or direct solvers [11]). A complete modeling tool will experience limitations from whichever one of its fundamental steps is the weakest with respect to memory and run-time efficiency. Therefore, the RSM-based $H^2$ algebraic methods proposed herein would be less useful if they were not complimented by efficient factorization methods that can manipulate sparse $H^2$ matrices into a form enabling efficient solution of an underlying matrix equation. In this direction, the last section of this thesis discusses additional work that was performed to develop

efficient and robust factorization methods for $H^2$ matrices. This portion of the work presented in this thesis does not involve RandNLA methods.

## Chapter 2 Background

In order to assist understanding, before detailing the contributions of the $H^2$ algebra via RandNLA based fill method and a fast $H^2$ factorization, the topics and data structures that these contributions were built upon must be described. First, the octree structure that is used to identify spatial relationships in the underlying problem geometry/mesh as well as the data structures that are used to store the $H^2$ matrix are detailed and discussed. Then, preexisting work by the University of Kentucky's Computational Electromagnetics Group (UKCEM) in $H^2$ fill and factorization methods are summarized.

### 2.1 Integral Equations

The focus of the work considered here is on integral equation (IE) formulations of CEM problems. Integral equation formulations use equivalence concepts (or Green's theorems) to develop a solvable set of constraints on the degrees of freedom in a given problem [12]. The development of IE formulations is not trivial. However, for the purposes of this thesis, it is sufficient to restrict our attention to the resulting matrix equation, which has the following form,

$$Zx = b. \tag{2.1}$$

In this equation, the $N$-by-1 vector $x$ denotes the unknowns (or degrees of freedom) in the problem, the $M$-by-$N$ matrix $Z$ is dense, and $b$ is the $M$-by-1 excitation vector. The examples in this thesis consider the particular IE obtained for magnetostatic applications when a locally corrected Nyström (LCN) method [13][14][15] is used to discretize the magnetostatic volume IE [16].

### 2.2 Octree Decomposition

The octree decomposition is a popular algorithm that constructs a tree structure for grouping and partitioning elements in three-dimensional space. It is well-known for its role in 3D computer graphics and spatial indexing. In CEM, the octree is used to spatially partition points (perhaps associated with mesh elements) of a problem geometry into groups in order to assist in a variety of phases within the analysis. In the context of this work, the octree provides the spatial groupings needed to identify near interactions and far interactions at each level of the octree.

(The following discussion of the octree usage refers primarily to mesh cells, which are treated as if they exist at a point in space. In practice, mesh cells span a finite region of space. Furthermore, the matrix $Z$ represents interactions between a large set of source and field basis functions, which themselves represent the underlying electromagnetic quantities' variations over the finite-sized cells of the mesh. A given basis function may in practice span multiple mesh cells; conversely, a single mesh cell

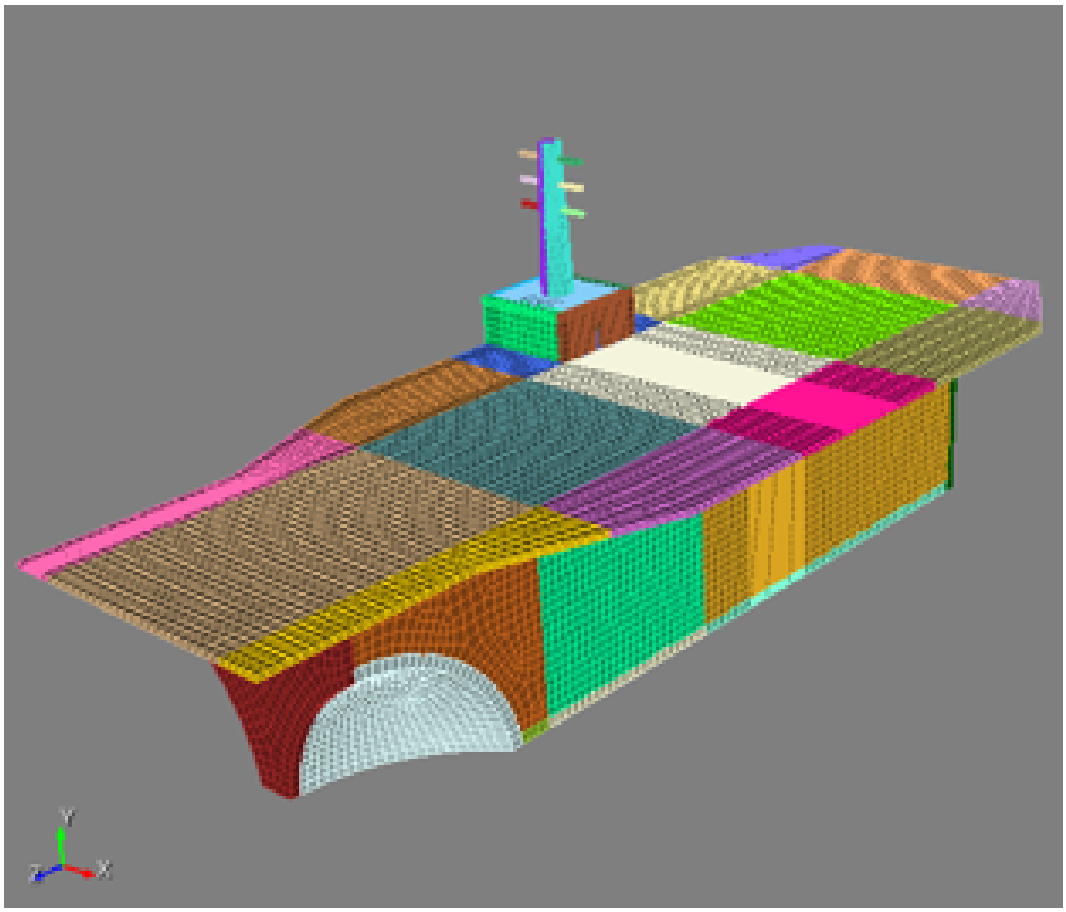


Figure 2.1: Meshed mock aircraft carrier.

may contain several basis functions. For these reasons, there are some practical details associated with developing an octree for IE formulations of problems in CEM that are not addressed in the following discussion, which treats all quantities, including mesh cells, as if they are located at (infinitesimal) points in space. However, the details that are not addressed here are not important for the discussion presented in this thesis, and they are well understood by most practitioners working in the field of CEM. For the purposes of this thesis, it is sufficient to have a basic understanding of how one can use an octree to obtain a distinct, multilevel organization of the rows and columns of the matrix $Z$ in (2.1).)

To pursue the octree decomposition, the problem geometry is first meshed; through this process, mesh cells are distributed across the target problem's geometry. These mesh cells are defined by points in 3-dimensional space and provide a spatially discretized representation of the geometry. The granularity of this discretized representation, the distribution of mesh cells, and how many points make up a cell depends on the meshing algorithm that is used. The specific details of the mesh (such as whether it is a hexahedral or tetrahedral mesh) are not important for the work reported in this document. This is because the methods used in this document depend on the properties of the underlying EM problem, and any mesh that enables accurate nu-

merical representation of those properties will suffice. An example of a mesh is shown in Figure 2.1.

Once a meshed geometry is available, the octree decomposition begins by encapsulating all of these mesh cells in a single cube; this is the root level of the octree. (The root level is herein referred to as level-1 of the octree.) Next, the single root-level cube is subdivided into octants, which reside at level-2 of the tree. Then, the octree decomposition recursively subdivides any octants that have mesh elements present within its boundaries. Octants continue to get subdivided into a maximum of 8 children as long as an octant contains mesh elements. For a static octree that has a desired number of recursive levels defined, the decomposition halts when that level is reached. This decomposition forms a tree structure where each node has a maximum of 8 children. Of course, if a subdivision results in a cube with no mesh cells present within it, then that cube is discarded and not subdivided any further. In this application, all of the points that make up the mesh cells that are encapsulated in the same cube at a given level of the octree is referred to as a *group*. (Mesh elements that span multiple groups are assigned to one of the spanned groups and none of the others.)

In the following discussion, the terms "near" groups, "far" groups, and "interaction" groups will be frequently used to refer to the relationship between a given group and all other groups at a given level of the octree. For a given group at a given level, *near groups* refer to all non-empty groups that share a face, edge, or vertex with that group (the self group is included in the near group list). *Far groups* are the set of remaining groups. Finally, the *interaction groups* are the subset of the far groups that are also the group's parent's touching neighbors' children. In 3-D, the maximum number of near groups is $3^3 = 27$, and the maximum number of interaction groups is $6^3 - 27 = 289$. (The maximum number of far groups is unbounded, growing geometrically with the tree depth.)

For the sake of visualizing the basic process of this tree-based spatial decomposition algorithm, a simple spatial decomposition is diagrammed. The quadtree decomposition is the 2-dimensional (2-D) equivalent to the 3-D octree decomposition and is easier to diagram. Figure 2.2 demonstrates the 3-level quadtree decomposition of an 11 cell mesh obtained using the 2-D equivalent of the octree decomposition described above. In Figure 2.2, the top half of the figure steps through the decomposition process on the mesh. Each gray circle represents a single mesh cell and the squares at each level represent the quadtree partitioning groups for the corresponding level. The bottom half of the figure shows the quadtree structure that is built from the decomposition process. The black circles in the tree represent children that contain mesh cells and the empty circles represent partition groups that are ignored. As discussed, partitions are ignored if the partition contains no mesh cells as a result of the subdivision of the parent group's contents.

Similar to how quadtree groupings can be depicted using squares, cubes can be projected onto target geometries to demonstrate the groupings generated by the group's octree decomposition. In Figure 2.3, the octree cubes of a 3-level decomposition of the aircraft carrier from Figure 2.1 are overlaid on the original geometry. Then in Figure 2.4, the result of a 5 level octree decomposition is overlaid. Each

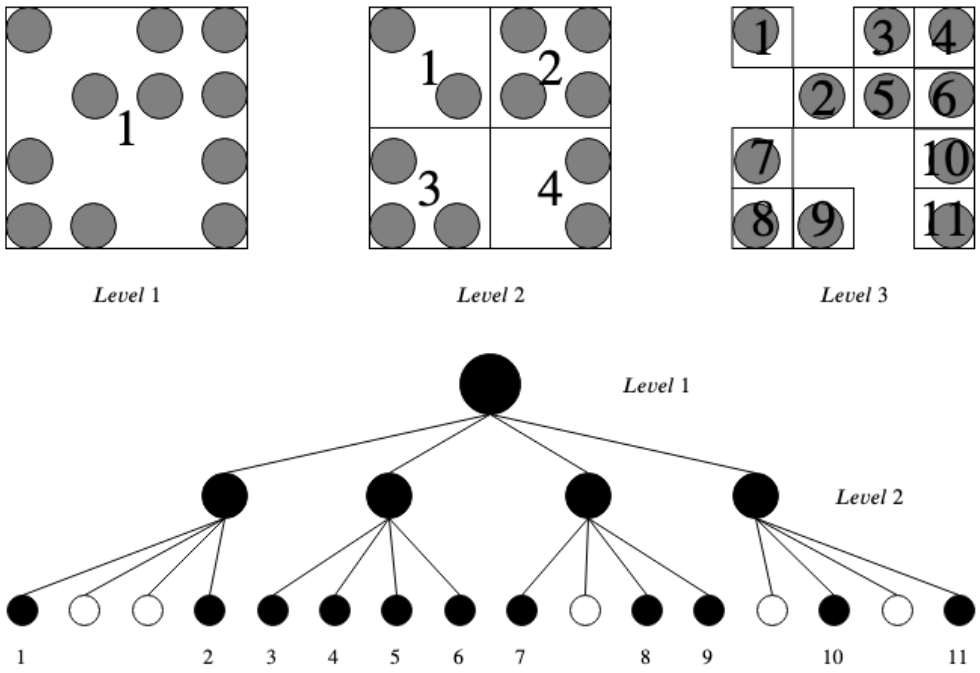



Figure 2.2: Illustration of a 3-level quadtree decomposition.

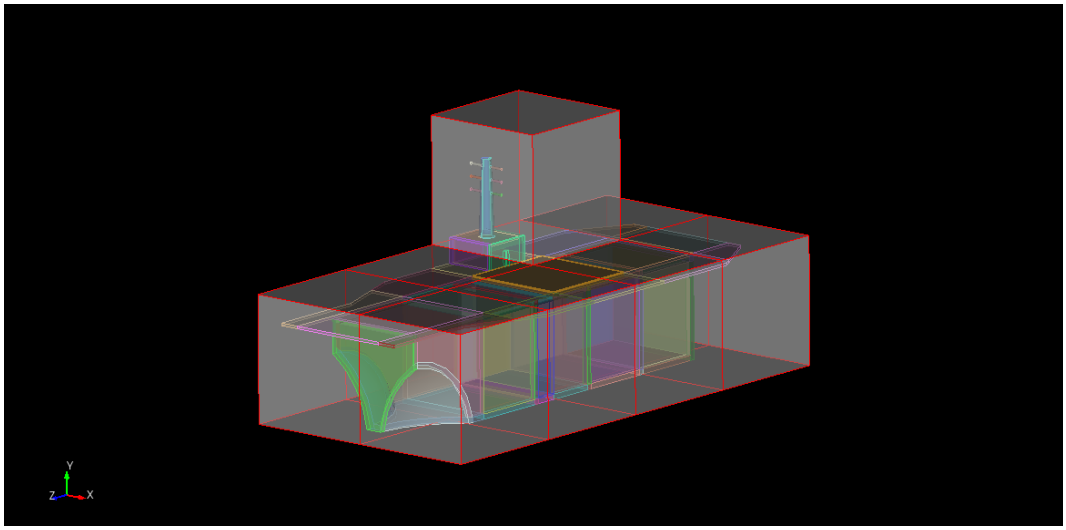


Figure 2.3: Mock aircraft carrier with level-3 octree groups overlaid.

cube seen in these figures is considered a spatial "group" of mesh cell points at that level of the octree.

In a Nyström Method based analysis, the degrees of freedom (DOF) can be associated with specific points on the mesh cells, and each DOF acts both as a transmitter and as a receiver. These DOF are often refered to as "source" and "test" functions. "Source" DOF correspond to columns of $Z$, and "field" DOF correspond to rows of $Z$. For the remainder of this document, consider that these DOF are antenna that generate (source DOF) or receive (field DOF) this electromagnetic radiation.

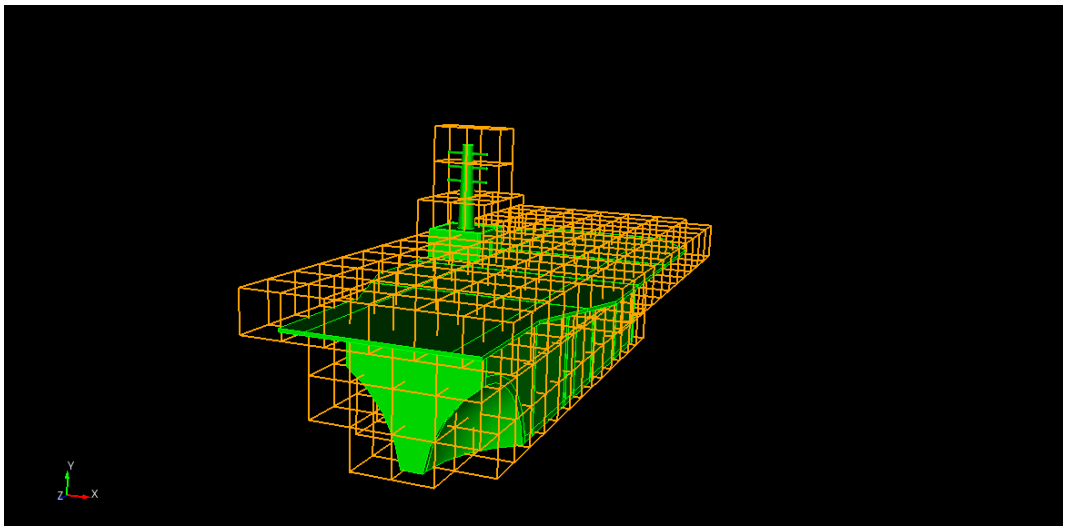


Figure 2.4: Mock aircraft carrier with level-5 octree groups overlaid.

The source and field DOF interact via the electromagnetic field components radiated from a source DOF to a field DOF, and this radiation is represented by an element of the matrix $Z$. Thus, every point in the mesh represents both a transmitter and receiver of electromagnetic fields. These fields are transmitted in a simple (usually homogeneous) background, and this is why the matrix $Z$ is dense.

Depending on the problem and the integral equation formulation used, electromagnetic radiation decays at a rate of either $1/r$, $1/r^2$, or, in some near field cases even, $1/r^3$ where r is the distance from the radiating source. Therefore, degrees of freedom that are respectively far from each other are interacting relatively weakly. This interaction can casually be interpreted as indicating that groups of DOF are separated from one another can convey less 'information' with each other than when those same groups of DOF are relatively closer together. In fact, degrees of freedom that are so close to each other that they are practically touching interact immeasurably strong because $\lim_{r\to 0} 1/r = \infty$. It is this difference in behavior between "near" and "far" interactions that makes the octree so useful in constructing efficient algorithms for CEM applications.

The presentation of the octree overlays in Figures 2.3 and 2.4 allow for the visualization of this notable far/near spatial behavior applied to groups rather than to individual degrees of freedom. Consider, for a given group cube, that all different group cubes that are touching the given cube are called near groups. Then, with respect to the same given group cube, every group that is not a near group is called a far group. Note from observing Figures 2.3 and 2.4 that, with an increase in level, groups have an increasing amount of groups that they consider to be far groups. Yet, for any given level, a group can only have a maximum number of 27 "near" groups in 3-dimensional space. It can also be assumed that, in general, a group considers many more groups to be far than it considers to be near.

The idea of far/weak and near/strong interactions will become important in the upcoming discussion of $H^2$ representations of $Z$. For now, it is mainly important to

understand that the octree's purpose is to indicate how groups are organized with respect to each other at each octree level. It is important to know which groups are close to and which groups are far from given groups at every level of the decomposition. These groups given by the octree decomposition ultimately correspond to columns in $Z$ when the groups are considered as sources and rows in $Z$ when the groups are considered as receivers. The discussion of the $H^2$ structure (Section 2.4) will explain how this information is used.

### 2.3 Block Matrices

Before the $H^2$ structure is discussed, three terms describing specific block matrices that are used throughout this document will be defined. The standard definition of a block matrix is a matrix that is defined using smaller matrices. A basic block matrix, $T$, composed of the matrices $A$ with size $M \times Q$, $B$ with size $M \times P$, $C$ with size $L \times Q$, and $D$ with size $L \times P$ may look like the following:

$$T = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) = \left(\begin{array}{cccc|cccc} a_{1,1} & a_{1,2} & ... & a_{1,Q} & b_{1,1} & b_{1,2} & ... & b_{1,P} \\ a_{2,1} & a_{2,2} & ... & a_{2,Q} & b_{2,1} & b_{2,2} & ... & b_{2,P} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{M,1} & a_{M,2} & ... & a_{M,Q} & b_{M,1} & b_{M,2} & ... & b_{M,P} \\ \hline c_{1,1} & c_{1,2} & ... & c_{1,Q} & d_{1,1} & d_{1,2} & ... & d_{1,P} \\ c_{2,1} & c_{2,2} & ... & c_{2,Q} & d_{2,1} & d_{2,2} & ... & d_{2,P} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{L,1} & c_{L,2} & ... & c_{L,Q} & d_{L,1} & d_{L,2} & ... & d_{L,P} \end{array}\right)$$

This shows that $T$ is a $(M+L) \times (Q+P)$ matrix made up of 4 smaller matrices. The smaller matrices that make up a block matrix are commonly referred to as the *blocks* of the block matrix. In the example above, $A$, $B$, $C$, and $D$ are the *blocks* of $T$. For the sake of clarity, the terms *block-row* and *block-column* will be used to describe the rows of blocks in a block matrix and columns of a block matrix, respectively, for the remainder of this document. So, $T$ has 2 block-rows and 2 block-columns. Therefore, block-row 1 of $T$ is be made up of blocks $A$ and $B$, block-column 1 is made up of $A$ and $C$, and so on. As seen above, the definition of block matrices requires that every block within a single block-row must have the same number of rows and every block within a single block-column must have the same number of columns. For example, it is seen that block-row 1 is composed of blocks with $M$ rows and block-column 1 is composed of blocks with $Q$ columns.

Three block matrices are referenced throughout this document that each describe a different organization of the basic block matrix structure seen above. First, a *block diagonal matrix* is a block matrix where the matrices along its diagonal are dense and the off-diagonal blocks are only matrices of all zeroes (or sparse). A block diagonal matrix containing the same $A$ and $B$ matrices as above along its diagonal forms a block matrix with 2 block-rows and 2 block-columns that represents

a $(M+M)\times(Q+P)$ matrix. This block diagonal matrix would look like the following:

$$\left(\begin{array}{c|c} A & 0 \\ \hline 0 & B \end{array}\right) = \left(\begin{array}{ccc|ccc} a_{1,1} & \dots & a_{1,Q} & 0_{1,1} & \dots & 0_{1,P} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{M,1} & \dots & a_{M,Q} & 0_{M,1} & \dots & 0_{M,P} \\ \hline 0_{1,1} & \dots & 0_{1,Q} & b_{1,1} & \dots & b_{1,P} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0_{M,1} & \dots & 0_{M,Q} & b_{M,1} & \dots & b_{M,P} \end{array}\right)$$

Second, a *block column matrix* is a block matrix with $N$ block-rows and 1 block-column. A block column matrix built from the same $A$ and $C$ as above and a third $R \times Q$ matrix $E$ would be a block matrix with 3 block-rows and 1 block-column that represents a $(M + L + R) \times (Q)$ matrix. This block column matrix may look like the following:

$$\left(\begin{array}{c} A \\ \hline C \\ \hline E \end{array}\right) = \left(\begin{array}{ccc} a_{1,1} & \dots & a_{1,Q} \\ \vdots & \ddots & \vdots \\ a_{M,1} & \dots & a_{M,Q} \\ \hline c_{1,1} & \dots & c_{1,Q} \\ \vdots & \ddots & \vdots \\ c_{L,1} & \dots & c_{L,Q} \\ \hline e_{1,1} & \dots & e_{1,Q} \\ \vdots & \ddots & \vdots \\ e_{R,1} & \dots & e_{R,Q} \end{array}\right)$$

Finally, the third organization of a block matrix used in this document is referred to as a *block row matrix*. A block row matrix is a block matrix with 1 block-row and $N$ block-columns. A block column matrix built from the same $A$ and $B$ as above and a third $M \times R$ matrix $F$ would be a block matrix with 1 block-row and 3 block-columns that represents a $(M) \times (Q + P + R)$ matrix. This block row matrix may look like the following:

$$\left(\begin{array}{c|c|c} A & C & F \end{array}\right) = \left(\begin{array}{ccc|ccc|ccc} a_{1,1} & \dots & a_{1,Q} & b_{1,1} & \dots & b_{1,P} & f_{1,1} & \dots & f_{1,R} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{M,1} & \dots & a_{M,Q} & b_{M,1} & \dots & b_{M,P} & e_{M,1} & \dots & e_{M,R} \end{array}\right)$$

### 2.4 $H^2$ Data Structure

It was previously mentioned that far interactions are "weak" while near interactions are "strong". This description of the physical problem is easily translated into direct mathematical terms. Performing a rank-revealing LU decomposition on submatrices of $Z$ corresponding to far and near interactions will show that far interaction submatrices are rank-deficient, whereas near interaction submatrices are full-rank (or nearly full-rank). For this reason, near-interaction blocks within a problem have the greatest influence on determining the solution of $Zx = b$. This occurs because

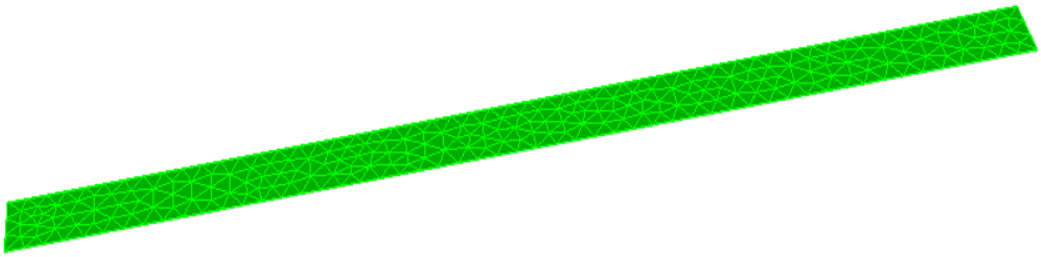


Figure 2.5: 1-D PEC Strip [17]

those blocks are transmitting a relatively stronger field level that provides a stronger constraint on the unknowns.

It was also previously noted that, apart from levels 1 and 2, the majority of the interactions at each level of the octree decomposition consist of far interactions. If these far interaction blocks are computed and stored in a dense manner, then large amounts of computation time and memory are consumed through constructing many dense representations of interactions that will not significantly influence the accuracy of the final solution. Therefore, significant computational savings can be had by approximating long-range interactions between groups of sources and receivers using lossy compression methods. In addition to reducing computational costs, this approach can be performed in a manner that controls the error in the approximation to a level that is acceptable to an end-user. This controllably accurate approximation of the interaction blocks is discussed in the next section. Before getting to that, we complete the current section by indicating how the $H^2$ representation utilizes the octree to specify a multilevel, nested data structure.

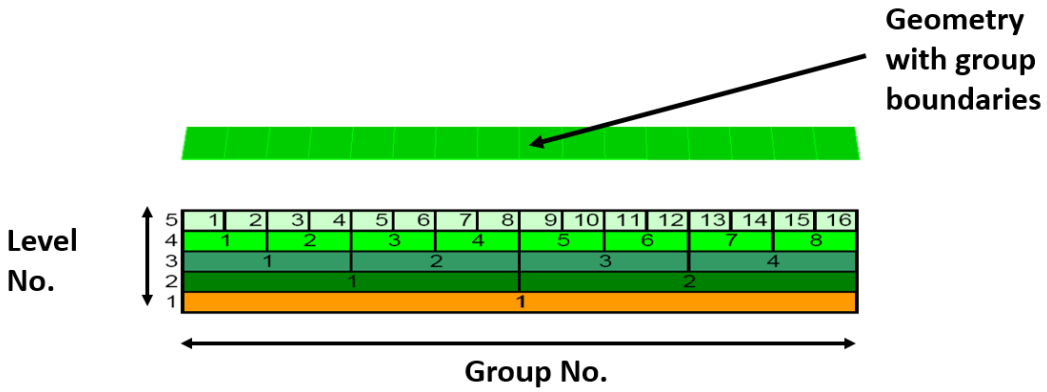



Figure 2.6: Binary tree decomposition of 1-D PEC strip [17]

The $H^2$ representation for $Z$ was originally presented by Wolfgang Hackbusch and has proven to provide effective data sparse representations of the dense matrices that result from discrete approximations of integral operators with smooth kernels [4]. Within the $H^2$ representation of a dense matrix, near interaction blocks at the finest level of the octree are stored in a dense form, while a controllably accurate, lossy compression is used for submatrices representing far interactions at each level

of the spatial decomposition. The lossy compression is obtained using singular value decompositions to determine the minimum number of degrees of freedom required to represent interactions between separated sources and observers [18][19][10] for a given level of desired accuracy. The specific details of these computations are not the subject of this research effort and are not detailed in this thesis. Instead, we summarize the basic structure of the resulting $H^2$ matrix, which is important to the following discussions.

The $H^2$ matrix has the following nested form for an $L$-level octree:

$$Z_m = \hat{Z}_m + U_m Z_{m-1} V_m^H, \ for\ m = 2, 3, ..., L\ where\ Z_2 = \hat{Z}_2 \tag{2.2}$$

In this equation, $m$ indicates the octree level of the $H^2$ decomposition. $\hat{Z}_m$ is the sparse matrix storing the near interactions at level $m$ that were not accounted for at finer levels. $U_m$ and $V_m^H$ are rectangular, orthogonal, block-diagonal matrices which compress far interactions via rank reduction at level $m$. It is also useful to note that $Z_L$ recovers the original interaction matrix $Z$ (*i.e.*, $Z = Z_L$). For the example illustrated in Figure 2.6, the expanded version of (2.2) is,

$$Z \approx \hat{Z}_5 + U_5 \hat{Z}_4 V_5^H + U_5 U_4 \hat{Z}_3 V_4^H V_5^H + U_5 U_4 U_3 \hat{Z}_2 V_3^H V_4^H V_5^H. \tag{2.3}$$

The spatial partitioning induced by the 5-level octree indicated in Figure 2.6 yields a corresponding partitioning of the rows and columns of the matrix, $Z$, which is indicated in Figure 2.7. In Figure 2.7, the red blocks include near interactions that are filled with a standard dense method at the finest level of the octree (level-5 in this example), and the green blocks indicate far interactions that are filled using the lossy compression provided by an SVD operation, which is in turn expedited by the adaptive cross approximation (ACA) discussed in Section 2.4. The resulting compression of far interactions from level 5 to 3 can be seen in Figure 2.8. Only near interactions (in red) and the outer product representation of the far interaction blocks for each level are stored, which provides the desired compression.

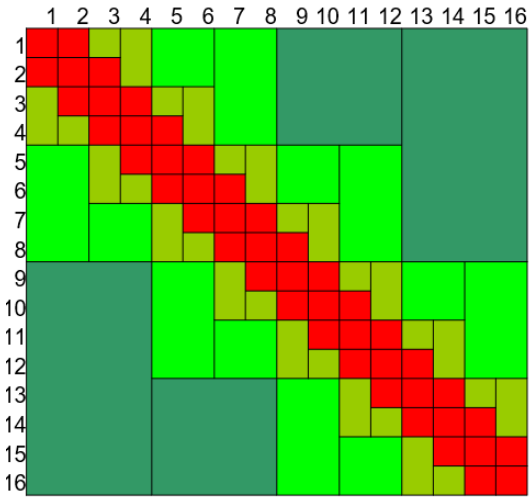
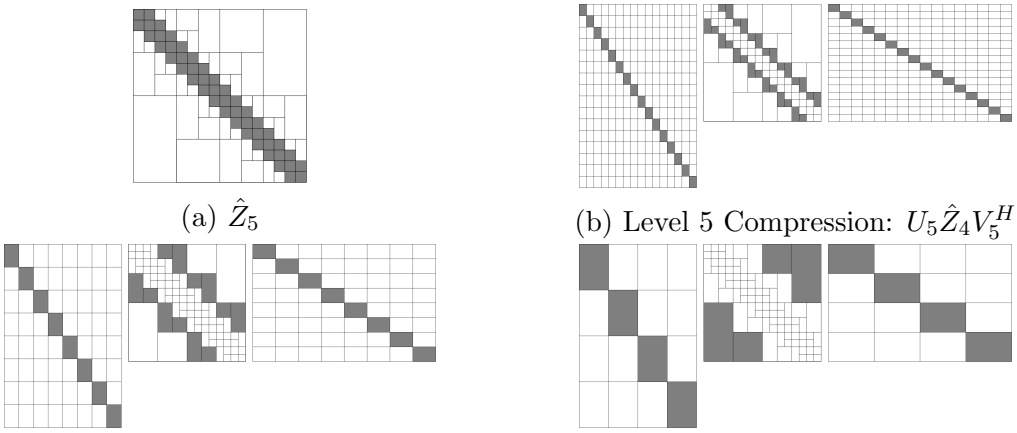



Figure 2.7: Filled system matrix 1-D PEC strip partitioned at level 5 ($\hat{Z}_5$) [17]



(a) $\hat{Z}_5$

(b) Level 5 Compression: $U_5\hat{Z}_4V_5^H$

(c) Level 4 Compression: $U_4\hat{Z}_3V_4^H$

(d) Level 3 Compression: $U_3\hat{Z}_2V_3^H$

Figure 2.8: Multilevel Compression (Only shaded blocks are stored)

## 2.5 Filling the $H^2$ using the Adaptive Cross Approximation

In order to fill the $H^2$ data structure from scratch, the submatrices of $Z$ representing near and far interactions must be filled. As noted above, near interactions are filled using standard, dense-matrix operations. This is done because near interactions are generally not compressible due to the rapid variation of the underlying kernel for short-range interactions. Sections of $Z$ that correspond to far interactions, on the other hand, are compressed using a singular value decomposition. Importantly, this SVD is obtained efficiently (that is, without sampling all elements in the submatrix) via the adaptive cross approximation (ACA) [7][20]. For a given source/field group pairing, the ACA builds a controllably accurate representation of the block by sampling only a subset of all rows and columns in the block. The singular value decompositions (SVDs) of the resulting representation of the block is then performed to obtain a controllably accurate, compressed representation. Next, these compressed

representations are assembled into the multilevel structure indicated in (2.2); details of this procedure are presented in [9]. In this way, a full representation of $Z$ is obtained by $i$) densely sampling the near-neighbor matrix at the finest level of the tree, and $ii$) sparsely sampling the interaction blocks at all coarser levels of the tree using the ACA.

### 2.6 Upper Triangular Factorization of Integral Equation Matrices

Later in this document, work done prior to the presented investigations in RandNLA will be detailed with respect to developments in factorization methods for integral equation matrices. The work that will be discussed in Chapter 8 is a modification applied to a previously developed upper-triangular factorization [21][22].

The upper triangular factorization is a multilevel, localizing factorization of $Z$ using localizing basis functions [11][21][23][24],

$$Z_{l+1}^{NN} = P_l P_l^H Z_{l+1}^{NN} \Lambda_l \Lambda_l^{-1} \approx \begin{bmatrix} P_l^{(L)} & P_l^{(N)} \end{bmatrix} \begin{bmatrix} I & Z_l^{(LN)} \\ 0 & Z_l^{(NN)} \end{bmatrix} \begin{bmatrix} \Lambda^{(L)} & \Lambda^{(N)} \end{bmatrix}^{-1} \tag{2.4}$$

In (2.4), $l$ = 2, 3, ..., L and $Z_{L+1}^{NN}$ recovers the original system matrix $Z$. $\Lambda_l$ is a permuted block diagonal matrix that contains localizing and non-localizing sources. $P_l$ is a unitary matrix of the same structure as $\Lambda_l$. A localizing source group radiates zero field external to the group. The fields radiated by these localizing degrees of freedom are orthogonal to the receiving vectors of $P_l^{(N)}$. This orthogonality results in the zero block seen in (2.4). The upper triangular structure allows for (2.4) to be easily inverted recursively up to level 2 of the tree where an LU decomposition must be used to factor the final $Z_2^{(NN)}$. The most important quality of this factorization is that $Z^{(NL)} = (P^{(N)})^H Z \Lambda^{(L)} \approx 0$ is controllably small and allows for the ability for easy inversion of (2.4) [9] without fill-in.

### 2.7 Problem Background

As discussed above, in most applications, the $M \times N$ system matrix $Z$ has many low-rank submatrices, which can be identified using an octree partitioning. In other words, $Z$ has $N^2$ non-zero entries, but only about $O(kN)$ of them are independent within some tolerance, $\tau$, with $k << N$. Additionally, $Z$ is large, so operating on or even building its full dense form is not feasible. $Z$'s large size ultimately motivates the advancements detailed regarding fill and factorization methods in this document. Currently, the ACA fill method is used to build far groups of $Z$ and the SVD of those sections are used to build the $H^2$ representation of $Z$.

The RandLNA work presented herein was initially motivated by the desire to build an $H^2$ representation of a preconditioned system matrix of the form $CZ$ where $C$ is a preconditioning matrix having a footprint restricted to near-neighbor interactions. That is, given an $H^2$ representation of $Z$ and a near-neighbor matrix $C$, how can one build a single $H^2$ representation of the product of these matrices, $CZ$? In this regard, it is important to understand that the primary challenge lies in finding an adequate

replacement for the ACA algorithm summarized above. The ACA, usually, provides an efficient method for building low-rank representations of interaction blocks in $Z$ because the underlying Green function has a closed-form representation that is relatively cheap to evaluate. In contrast, the submatrices of $CZ$ do not have a closed analytical form that can be used to compute rows and/or columns. For this reason, while the ACA method can be used to build submatrices of the product $CZ$, the resulting procedure to determine an $H^2$ representation of $CZ$ will be too computationally expensive to be useful.

Rather than using the ACA to build low-rank submatrices, a method is needed that can utilize the fact that each of the matrices $C$ and $Z$ are themselves sparse. For example, it is possible to perform fast matrix-vector products using sparse matrices. And this is where RandLNA methods enter – it is now fairly well-known that randomized methods can be used to efficiently perform SVDs of low rank matrices if we can efficiently perform matrix-vector products using those matrices [1][2][25]. (It is also worth nothing that RandLNA does not provide a replacement for the ACA method in filling the original representation of $Z$, since one cannot perform efficient matrix-vector products until the $H^2$ of $Z$ is already formed. This is why the RandLNA methods discussed herein are complementary to, rather than replacements of, the ACA.)

As the RandLNA work discussed herein progressed, it became apparent that, if one could properly leverage the dimensionality reduction and sampling properties offered by randomized versions of the SVD to target preexisting $H^2$ structures, then it would be possible to implement additional operations beyond simple preconditioning. For example, the proposed RandLNA implementation allows for not just preconditioning of a matrix, but also algebraic operations, such as the addition of multiple $H^2$ matrices without ever needing to decompress the individual $H^2$ structures. Additionally, because the overarching mechanic is an SVD, then the results could be used in creating a new $H^2$ fill method that can fill a new $H^2$ structure using the results acquired from performing efficient RandNLA based algebra on preexisting $H^2$ structures.

This functionality already had an application in the UKCEM's preexisting research where the nonlinear relationship between current and voltage in a problem requires algebra on multiple $H^2$ data structures [26]. This elevated functionality built into a randomized SVD would allow the components of the Schur Complement to also be compressed into their own individual $H^2$ structures and then be manipulated as necessary while substantially improving the memory efficiency of the required Schur Complement process. In addition to this preexisting application, the generality of having efficient RandNLA based $H^2$ algebra is undoubtedly a powerful capability. The algorithm that was developed to make these capabilities a reality was coined as the *random sampling method* (RSM). One of the most novel properties of this method, is that it was purposefully designed and developed to mimic ACA fill process's input and output paradigms exactly. This allowed for the incorporation of the RSM's capabilities within the UKCEM's Modular Fast Direct code base in a very non-invasive manner.

# Chapter 3 Randomized Singular Value Decomposition

The core operation used in assembling an $H^2$ representation of a matrix is the SVD of submatrices of the matrix. For the case of the matrix $Z$, this operation is expedited using the ACA. However, when performing $H^2$ algebra, we will use RandLNA methods to replace the ACA-enabled SVD with a RandLNA-enabled SVD (rSVD). The core operation that is used to perform the rSVD is the random sampling of a given submatrix. For this reason, we refer to the randomized methods used herein to build $H^2$ matrices as random sampling methods (RSMs).

Randomized SVDs take advantage of the following properties:

- RandNLA's dimensionality reduction (that is, randomly chosen basis vectors are highly likely to span the desired subspace of low-rank matrices)
- rSVD's ability to compress submatrices in the $H^2$ data structure (with the assistance of the work in Chapter 4)
- The ease with which applying intermediate algebraic operations on the data can be done within a rSVD algorithm
- The fact that the after applying intermediate algebraic operations to the data, the results are presented in the standard SVD $U$,$\Sigma$, and $V^H$ matrix set; this is the same matrix format obtained using ACA expedited fill methods, and thus enables easy integration with an existing, ACA-based $H^2$ fill code.

An SVD is appealing because the resulting $U$,$\Sigma$, and $V^H$ matrices that are built from the sampled $H^2$ data, which was subject to algebraic operations, can immediately be used to fill a new $H^2$ data structure. Therefore, this newly constructed $H^2$ will reflect the intermediate algebraic operations that were applied during the rSVD. The output of a $U$,$\Sigma$, and $V^H$ matrix set mimics the ACA's output, which is essential for developing a non-invasive fill method alternative. Therefore, before addressing the RSM, discussing a basic randomized SVD that operated on the $H^2$ structure will cover all of the RandNLA present in the RSM without the RSM's additional functionality obscuring the details of the RandNLA principles. Yet, before discussing a simplified $H^2$ randomized SVD, a basic randomized SVD needs to be described in order to identify the challenges that come with a $H^2$ randomized SVD.

## 3.1 Basic Randomized SVD on a dense, full matrix $Z$

A basic randomized SVD on a dense $M \times N$ matrix $Z$ can be described as follows [27]:

Step 1: Generate $\Omega$ which is a set of $K$ random vectors, forming an $N \times K$ matrix

Step 2: Calculate $Y = Z\Omega$ ($Y$ will be $M \times K$)

Step 3: Find an orthogonal basis, $Q$, that spans the range of $Y$ ($Q$ will be $M \times q$)

Step 4: Form a temporary matrix, $B$, by $B = Q^H Z$ ($B$ will be $q \times N$)

Step 5: Get SVD of $B$, giving $B = \bar{U}\Sigma V^H$

Step 6: Finally, $U$ can be found by applying the orthogonal basis to $\bar{U}$. Giving $U = Q\bar{U}$

We briefly justify this six-step procedure for estimating the SVD of matrix $Z$ as follows. A required component to assure this randomized algorithm's proper functionality is to build a orthogonal matrix, $Q$, such that $Z \approx QQ^H Z$. This can be done by first estimating the range of $Z$ through randomly sampling the column space of $Z$ by multiplying it against a set of uniformly or normally distributed random vectors $\Omega$. Then, an orthogonal basis that spans the range of the resulting subspace from $Z\Omega$ needs to be found. Finding an orthogonal basis that spans the range of the $Z\Omega$ subspace can simply be found by taking the QR decomposition of the subspace. This is because, if $Z\Omega$ has $n$ linearly independent columns, then the first $n$ columns of the $Q$ matrix resulting from $Z\Omega = QR$ will form an orthogonal basis for the column space of $Z\Omega$. As long as $Z$'s structure is well sampled by using a sufficient number of random vectors (more on determining this number in Chapter 6), then $Q$ should span the main range of $Z$ and be a near-optimal basis. This connection between $Q$ and $Z$ is known because $\mathcal{R}(Q) = \mathcal{R}(Z\Omega) \subseteq \mathcal{R}(Z)$ where $\mathcal{R}(\cdot)$ denotes the column space or range of a matrix [28].

One fundamental definition of the singular value decomposition is that the columns of $U$ from $Z = U\Sigma V^H$ are the left-singular vectors of $Z$ and one fundamental property of the singular value decomposition is that the left-singular vectors that correspond to the non-zero singular values of $Z$ provide an orthonormal basis that spans the range of $Z$. This gives us $\mathcal{R}(Z) = \mathcal{R}(U)$. It is now seen that $\mathcal{R}(Q) = \mathcal{R}(Z\Omega) \subseteq \mathcal{R}(Z) = \mathcal{R}(U)$ and resolving the equality gives $\mathcal{R}(Q) \subseteq \mathcal{R}(U)$. Therefore, the factor $Q$ must capture the dominant left singular vectors of $Z$ [28].

In step 4, $Q$ can be used to apply dimensionality reduction on $Z$ by forming $B = Q^H Z$. Conveniently, because $Z^H = V\Sigma^H U^H$, it is true that $\mathcal{R}(Z^H) = \mathcal{R}(V)$.

Additionally, that means that $\mathcal{R}(B^H) \subseteq \mathcal{R}(Z^H) = \mathcal{R}(V)$ and that $B$ must provide information on the dominant right singular vectors of $Z$ [28].

In Step 5 above, with the information provided by $B$ and the RandNLA dimensionality reduction provided by $Q^H$, the deterministic SVD of a $q \times N$ matrix is performed to acquire $B = \bar{U}\Sigma V^H$. Then, because $Q$ was built to achieve $Z \approx QQ^HZ$, then $Z \approx QQ^HZ = QB = Q\bar{U}\Sigma V^H$. Finally, after considering $Z = U\Sigma V^H$ we can derive that $Z = U\Sigma V^H \approx Q\bar{U}\Sigma V^H$. It is now shown that all is needed is to apply the found orthogonal basis, $Q$ to $\bar{U}$ in order to recover the excluded singular vectors that are not present in $\bar{U}$ but were previously proven to be captured in $Q$. This leaves approximately equivalent $U$,$\Sigma$, and $V^H$ matrix sets for the SVD and rSVD of $Z$. At this point it is perhaps worth emphasizing that *the only operations required involving $Z$ are left and right matrix-vector multiplications.* Other operations, such as QR and SVD decompositions, are performed only on results of these products and not on $Z$ itself. This is why, provided the number of columns in $\Omega$ is small compared to the number of columns in the matrix $Z$, randomized methods provide an efficient replacement of the ACA algorithm when performing $H^2$ preconditioning and algebraic operations.

Of course in all RandNLA methods and algorithms, error can be introduced as a result of the random sampling of the column space, especially if the vectors of $\Omega$ have little orthogonality. This error will be discussed further alongside the RSM in Chapter 6. For now, it is only important to notice the trivial nature of the algorithm above given that it is operating on a dense matrix. Additionally, if it was necessary to take a randomized SVD of the full $H^2$ structure, then the algorithm would not have a significantly different algorithmic pattern of that found in the trivial example above. Therefore, a far more useful form of a $H^2$-specific randomized SVD will be described.

# Chapter 4 Group Specific Matrix-$H^2$ Left And Right Multiplies

It has been previously established that the initial, ACA-based $H^2$ fill process iteratively samples rows and columns of a submatrix of $Z$ (associated with a given set of source and field groups), and the SVD is then taken of each of the row/column sample sets. So, describing a basic randomized SVD that takes the SVD of the full $H^2$ is not very useful as that bulk operation is never used in the fill or factorization processes that are being discussed in this document. Instead, it is more useful to discuss a randomized SVD that can target specific source groups and field groups at specific levels within the $H^2$ data structure (that is, a submatrix of the desired $H^2$ structure).

There are two multiplies against the $H^2$ structure that occur in an $H^2$-specific randomized SVD (see Section 3.1). First, a right multiply would be required for $(H^2)\Omega$. Second, a left multiply would be required for $Q^H(H^2)$. In this regard, it is important to observe that these matrix-multiply operations occur on vectors that do not span the entire domain/range of the $H^2$ matrix. For example, depending on the submatrix of the $H^2$ being constructed, $\Omega$ may span only a single group at the finest level of the octree. However, prior to the work discussed in this thesis, the only multiplies that existed in the UKCEM tool set for $H^2$ matrices were those to multiply an $H^2$ matrix on the left or right by a matrix, $B$, that contained *all* degrees of freedom represented in the $H^2$. Therefore, the entire $H^2$ structure would be recursed by the multiply giving (4.1) and (4.2).

$$BZ = B\hat{Z}_{Near} + \sum_{n=1}^{L-2} BU_L ... U_{L-(n-1)} \hat{Z}_{L-n} V_{L-(n-1)}^H ... V_L^H \tag{4.1}$$

$$ZB = \hat{Z}_{Near} B + \sum_{n=1}^{L-2} U_L ... U_{L-(n-1)} \hat{Z}_{L-n} V_{L-(n-1)}^H ... V_L^H B \tag{4.2}$$

For each column in $B$, each of these is an $\mathcal{O}(N)$ operation.

In (4.1) and (4.2), $\hat{Z}_{Near}$ is just $\hat{Z}_L$ when $L$ is the finest level of the problem's octree decomposition. These multiplies are useful, but if multiplies against only specific groups at a certain level are desired, which is common during the far group build process, then this full manner of multiplying with the $H^2$ is very wasteful. Therefore, adaptations were applied to the existing $H^2$ multiplies in order to make group targeting possible. Developing these multiplies within the UKCEM tool set constituted a primary component of the effort.

## 4.1 Operand Matrix Partitioning by Target Groups

Consider that a given matrix, $B$, is being right multiplied onto $Z$ (i.e. $ZB$). It has already been established that $Z$ could be massive and should not be operated on

via its dense representation. Therefore, $B$ must be multiplied by the right of the $H^2$ representation of $Z$ (i.e. $H^2B$). For this example, consider the $H^2$ and octree decomposition given in Figure 2.6 and that $B$ is to only operate on the source groups 1 and 3 at the finest level (i.e., level-5). Consider that group 1 contains degrees of freedom 2, 5, and 7 and group 3 contains degrees of freedom 1, 3, 8, 9. Because each degree of freedom represents a row and column of the full, dense representation of $Z$, degrees of freedom are organized on $Z$ continuously and sequentially from 1 to N with N being the total number of degrees of freedom in $Z$. In the context of this document, $B$ will always be organized and indexed globally by degrees of freedom, because $B$ is constructed to operate on $Z$ rather than $Z$'s compressed representation via the $H^2$. Yet, the $H^2$ is organized and indexed by octree groups, not degrees of freedom. Additionally, it is unlikely that the degrees of freedom are partitioned into octree groups in such a way that indexing by groups is equivalent to indexing globally by degrees of freedom like how $Z$ is indexed. Therefore, because of this difference in organization and indexing between $H^2$ and $B$, it is unlikely that the operand $B$ is organized in a way that it could operate on the $H^2$ without mapping to the $H^2$'s indexing space. So, if $B$ operates on $Z$ and only on groups 1 and 3 at the finest level, $B$ may look something like

$$\begin{pmatrix} a1 & a2 & ... & aN \\ b1 & b2 & ... & bN \\ c1 & c2 & ... & cN \\ d1 & d2 & ... & dN \\ e1 & e2 & ... & eN \\ f1 & f2 & ... & fN \\ g1 & g2 & ... & gN \end{pmatrix} \tag{4.3}$$

where rows $b$, $d$, and $e$ would need to operate on group 1 and rows $a$, $c$, $f$, and $g$ would need to operate on group 3. $B$ could be indexed by rows in order to extract the correct rows to multiply against the correct blocks in the $H^2$ structure, but it is more convenient to preprocess the matrices operating on the $H^2$ and format them into a block matrix that is organized by groups like the $H^2$. So, before operating on source or receiver groups, $B$ is converted into a block column or block row matrix, respectively. This would make the current hypothetical $B$ to be organized in the following manner:

$$\left(\begin{array}{cccc} b1 & b2 & ... & bN \\ d1 & d2 & ... & dN \\ e1 & e2 & ... & eN \\ \hline a1 & a2 & ... & aN \\ c1 & c2 & ... & cN \\ f1 & f2 & ... & fN \\ g1 & g2 & ... & gN \end{array}\right) \tag{4.4}$$

The operation of mapping from the full (or, non-grouped) representation of $B$ in (4.3) to the group-wise partitioned representation of $B$ in (4.4) is referred to as *operand partitioning by target groups*. The "target groups" are the finest-level groups of the $H^2$ structure that are spanned by $B$.

## 4.2 Basic $H^2$ Right Multiply Against Matrix

Now that the purpose of the initial structure change of the matrix operand has been described, the group specific $H^2$ right multiply process is detailed. In general, in order to multiply the $H^2$ by a matrix, one must recurse through all of the layers that are formed from the $H^2$'s nested structure. (Depending on the structure of $B$ in relation to the octree group boundaries, in some cases it may not be necessary to incorporate all layers.)

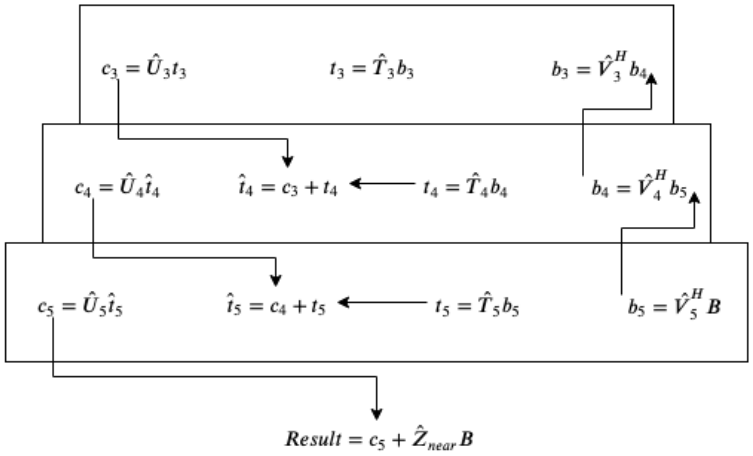



Figure 4.1: Group Specific $H^2$ Right Multiply Matrix

In Figure 4.1, starting at the bottom right hand side of the pyramid and following counterclockwise around the pyramid, the translation of $B$'s contributions across the $H^2$ layers is seen. In Figure 4.1, $\hat{U}_m$, $\hat{V}_m^H$, and $\hat{T}_m$ represent a block sample corresponding to $U_m(rcvGrps_m, rcvGrps_m)$, $V_m^H(srcGrps_m, srcGrps_m)$, and $T_m(rcvGrps_m, srcGrps_m)$ block matrices, where $T$ is equivalent to $\hat{Z}_m$ when $m$ is not equal to the finest level, $rcvGrps_m$ are the targeted receiver groups at level $m$, and $srcGrps_m$ are the targeted source groups at level $m$. Other than the sampling of $U_m$, $V_m^H$, and $T_m$, the activity shown in Figure 4.1 is executing the exact same operations seen in equation (4.2).

Because $B$ is being right multiplied against the $H^2$, $B$ must start its translation process by being multiplied by the finest level's $V^H$ blocks that correspond to groups 1 and 3. A very important property of the compressing outer product matrices can now be clearly seen through this example. The outer product matrices at each level act as a translating matrix to its parent level's information. In other words, in order to operate on level 3's information, one must translate through level 4's and 5's compressing outer product matrices. This translation is required for numerically mapping a given operand into coarser level's compressed organization. Therefore, the operand $B$ must be initially organized with respect to the finest level, regardless of whether groups at the finest level are the explicit target or not. For example, as part of building an $H^2$ representation, one will of course need a level-3 block of the underlying matrix for which the $H^2$ is being constructed. However, due to the nested

nature of the $H^2$, any $H^2$ matrix-vector products that are used as part of the process will have to start with vectors that are partitioned in terms of finest level groupings.

This simple initial reorganization with respect to the finest level can be illustrated using the example of Figure 2.5. Consider having $B$ target group 1 at level 3 of the geometry seen in Figure 2.6. In order for $B$ to be reorganized at the finest level, $B$ just needs to be organized with respect to group 1 at level 3's finest level descendants, which are groups 1, 2, 3, and 4. So, $B$ will be initially restructured to be a block column matrix with 4 block-rows. $B$'s rows will be properly distributed throughout the four new block matrices so that the rows are still operating on the same degrees of freedom they would have corresponded to in a dense multiply with $Z$.

Once initially reorganized, $B$ can begin to be applied to form the $b_m$ matrices on the upward pass. $b_m$ must also be reorganized before continuing on to operate on $\hat{V}^H_{m-1}$. $\hat{V}^H_{m-1}$ is going to be organize with respect to level *m-1* groups, yet $b_m$ will initially be organized with respect to level $m$ groups. This reorganization step has been called *ascension*. The opposite of ascension, being reorganized with respect to a finer level's groups, is called *descension*. In Figure 4.1, the ascension and descension steps are depicted by arrows that cross over into a different level. The ascension of $b_m$ utilizes the tree to form a new version of the block column matrix that is organized by its parent groups. If the parent contains more than one of the groups being targeted in $b_m$ as children, then those blocks are simply stacked together. If the parent encapsulated degrees of freedom that are not being operated on by $B$, then the corresponding rows of $b_{m-1}$ are padded with zeros. For example, consider applying ascension to just the initial $B$ operand. Going back to the original example of targeting groups 1 and 3 at level 5 with $B$, it can be seen in Figure 2.6 that group 1 at level 5 would ascend into group 1 at level 4 and group 3 at level 5 would ascend into group 2 at level 4. Consider that group 2 at level 5 contains degrees of freedom 4 and 6, and group 4 at level 5 contains degrees of freedom 10 and 11. Then, because groups 1 and 2 at level 5 fall under group 1 at level 4, and groups 3 and 4 at level 5 fall under group 2 at level 4, the ascended form of $B$ would be

$$\left(\begin{array}{cccc} b1 & b2 & ... & bN \\ d1 & d2 & ... & dN \\ e1 & e2 & ... & eN \\ 0 & 0 & ... & 0 \\ 0 & 0 & ... & 0 \\ \hline a1 & a2 & ... & aN \\ c1 & c2 & ... & cN \\ f1 & f2 & ... & fN \\ g1 & g2 & ... & gN \\ 0 & 0 & ... & 0 \\ 0 & 0 & ... & 0 \end{array}\right)$$

Because $B$ still only operates on the degrees of freedom that fall under groups 1 and 3 at level 5, it can be seen that zero padding is inserted into rows that correspond to degrees of freedom 4, 6 in group 2 of level 5 and to degree of freedom 10 and 11 in group 4 at level 4. It is important to remember that the $H^2$'s levels are always organized by

groups. Yet, reorganizing rows in order to match them to their corresponding degrees of freedom is only required at the finest level. Once the rows are reorganized with respect to the finest level, only stacking operations need to occur at all coarser levels because within the UKCEM toolset parents construct their own degree of freedom lists by simply concatenating their children's DOF lists. Alternatively, if group 1 and 3 at level 5 were to both be the only children of group 1 at level 4, then the two child blocks would just be stacked. This would make the ascended form of $B$ be

$$\begin{pmatrix} b1 & b2 & ... & bN \\ d1 & d2 & ... & dN \\ e1 & e2 & ... & eN \\ a1 & a2 & ... & aN \\ c1 & c2 & ... & cN \\ f1 & f2 & ... & fN \\ g1 & g2 & ... & gN \end{pmatrix}$$

In Figure 4.1, the ascension process is applied to every $b_m$. Additionally, the descension process is intuitively the reverse process of ascension. Row blocks of the block column matrix being descended are just split apart out of the parent block. Because parents construct their own degree of freedom lists by simply concatenating their children's degrees of freedom lists, the proper rows to split out of the parent block for a given child to form its own block is known by examining the number of degrees of freedom that the child encapsulates and the order at which the parent has its children stored. The parent group will store a list of child groups in an order that is the same as the order at which the child's degree of freedom lists were concatenated to create the parent's degree of freedom list. So, starting with the first child in the parent's child list, the rows 1 to F, where F is the number of degrees of freedom under the first child group, will be extracted from the parent block to form that child's descended block. Then, the second group in the parent's child list will extract rows F+1 to F+1+G, where G is the number of degrees of freedom under the second child group, from the parent block to form its descended child block. This process is repeated for every child group under the parent.

### 4.3 Left and Right $H^2$ Multiplies with Truncation

The left multiply is implemented as the transpose of the right multiply for the ease of development. Consider the right and left multiply examples in Figures 4.2 and 4.3, respectively. The multiplies in Figures 4.2 and 4.3 are operating on an $H^2$ with more levels than our previous example in order to highlight some interesting spatial behavior that allows for cost saving decisions to be made in the $H^2$ multiplies with sufficiently deep trees. Observe the behavior in Figure 2.8. The shaded blocks of $\hat{Z}_m$ never touch any blocks past one level above $m$ and one level below $m$. The bandwidth of a given $\hat{Z}_m$ is isolated to only levels *m-1*, *m*, and *m+1*. Therefore, the near-neighbor interactions at any level that are not being touched by a near-neighbor at a specific level will not contribute to operations on those near-neighbor groups.

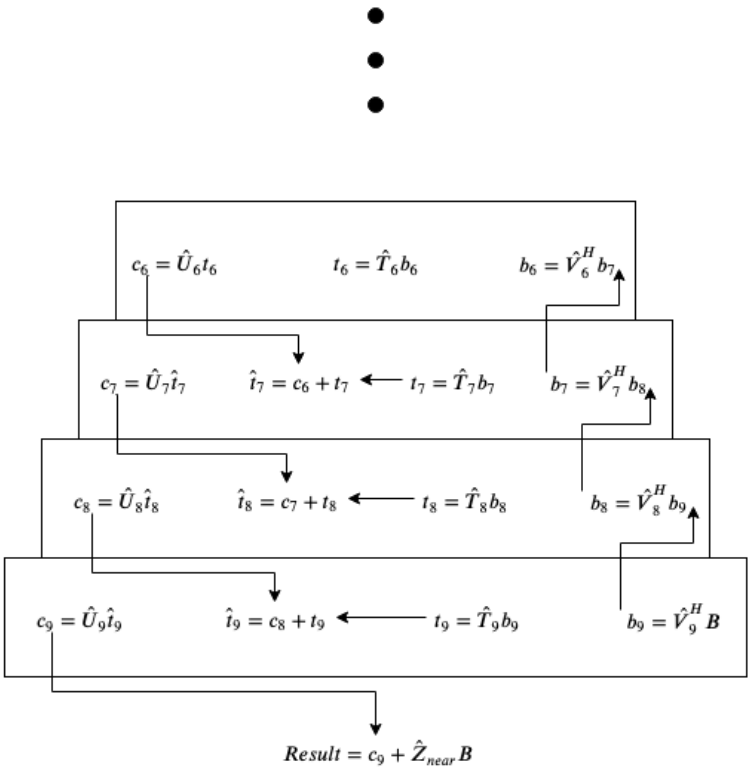



Figure 4.2: Group Specific $H^2$ Right Multiply Matrix

Therefore, if the $H^2$ multiply is targeting groups at level 7 of the $H^2$ depicted in Figures 4.2 and 4.3, the $T$ blocks finer than level 8 and coarser than level 6 and $Z_{near}$ can be ignored. Therefore, the multiplies seen in Figures 4.2 and 4.3 respectively show what the multiplies in Figures 4.4 and 4.5 would look like when the ability to to ignore some of these near-neighbor interactions is utilized. Ignoring these blocks allow for some operations savings *with no error introduction* because of those block's lack of contributions. This feature of the $H^2$ multiplication is referred to as *truncation*.

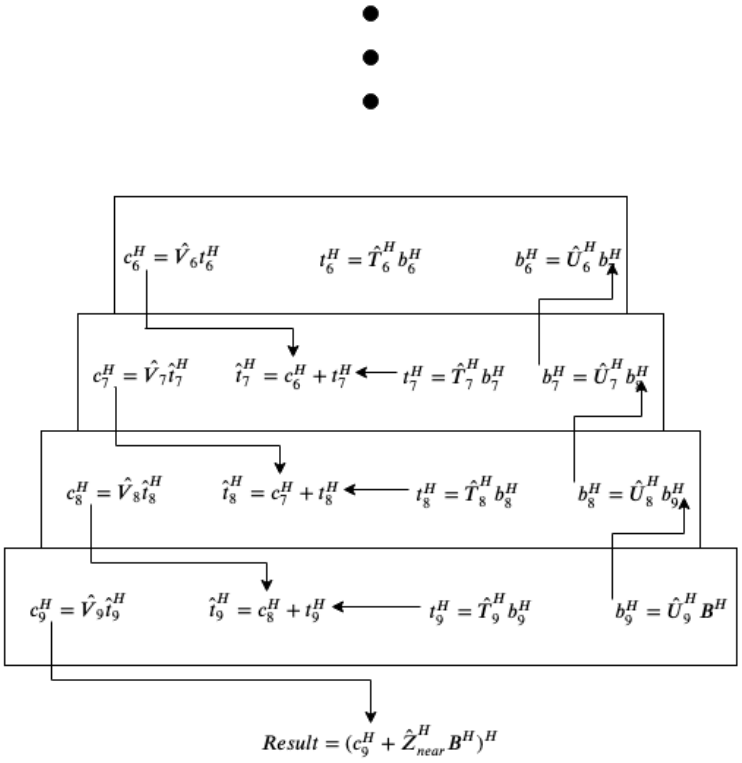



Figure 4.3: Group Specific $H^2$ Left Multiply Matrix

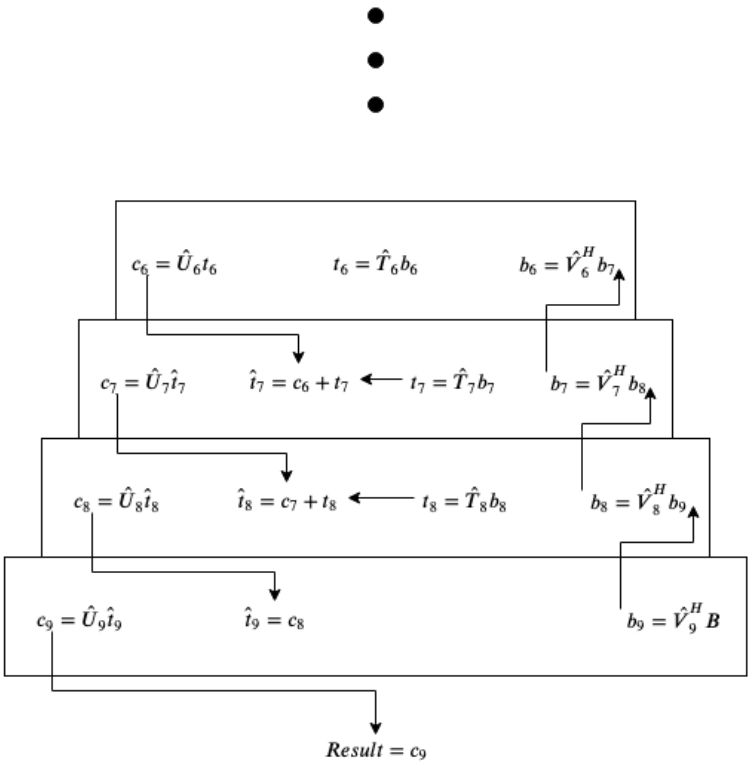



Figure 4.4: Group Specific $H^2$ Right Multiply Matrix targeting groups at level 7 showing truncation

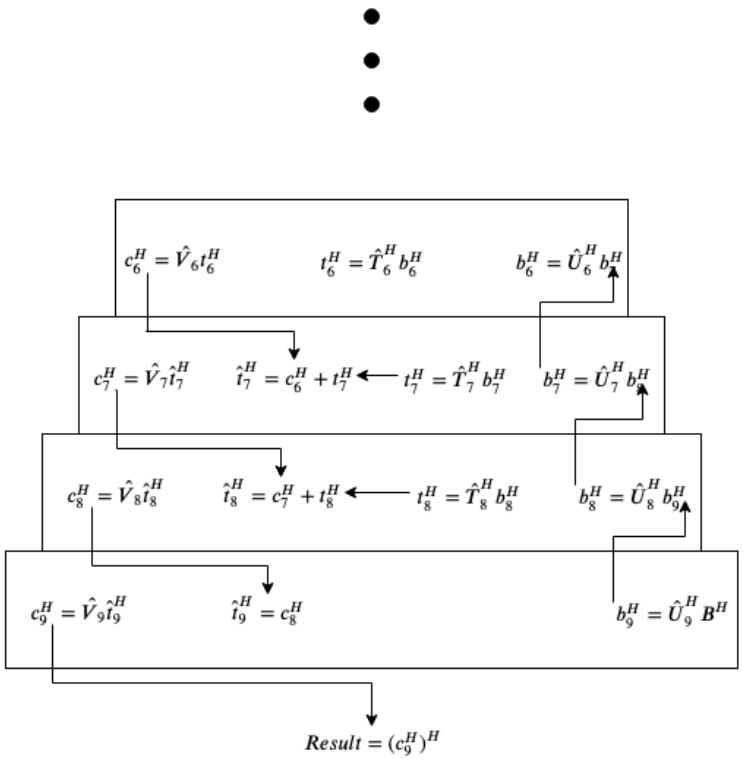



Figure 4.5: Group Specific $H^2$ Left Multiply Matrix targeting groups at level 7 showing truncation

## Chapter 5 Randomized SVD for $H^2$ Matrices

With proper group targeting $H^2$ multiplies having been defined above, it is now possible to describe a randomized SVD for $H^2$ matrices. It is quickly noticed by observing Algorithm 1 that the general flow is almost identical to that seen in section 3.1. The primary differences are in the details that come with converting a randomized SVD to operate on block matrices and the context that is introduced by operating on the $H^2$ structure.

**input :** $H^2$, tree, srcGrps, rcvGrps, level, tolerance
**output:** $U$,$\Sigma$, and $V^H$
1: Determine number of random vectors, K //will be discussed in Section 6.3;
2: $\Omega$ = random(numOfDofs(srcGrps), K) //uniform or normal distribution;
3: $H\Omega = H^2_{level}$(rcvGrps, srcGrps)$\Omega$;
4: $[\hat{u}, \hat{s}, \hat{v^H}]$ = truncated-svd($H\Omega$, tolerance);
5: $Q = \hat{u}$;
6: $B = Q^H H^2_{level}$(rcvGrps, srcGrps);
7: $[\bar{U}, \Sigma, V^H]$ = svd($B$);
8: $U = Q\bar{U}$;
9: return $U$,$\Sigma$, and $V^H$;

**Algorithm 1:** $H^2$-based randomized SVD

In dense randomized SVD's, $\Omega$ is a uniformly distribution set of $K$ random vectors of length $N$ that is employed in order to apply the initial dimensionality reduction and random sampling steps of the RandNLA algorithm (determining $K$'s value is discussed in Section 6.3). $\Omega$ fills the same role here but is also used to perform sampling of $Z$ via operations on the $H^2$ structure. This sampling is a very useful utility that comes almost naturally when applying a randomized SVD to $H^2$ matrices in this group specific manner. The randomized SVD is designed to take in a set of source groups, a set of receiver groups and a level at which those groups reside so that the randomized SVD can be applied to the matrix block for a specific set of source and receiver groups at a given level. This design of the randomized SVD is meant to mimic the ACA's input and output paradigm.

During the ACA-based build method, the ACA is called to build, from scratch, a specific section of $Z$ that corresponds to a given set of source and receiver groups. The ACA relies on a row/column sampling method along with calling to a method of moments or Nyström Method backend in order to build a far block of $Z$ [13][20][29][30]. After building a far block of $Z$, the SVD of the built block is provided as output for use in constructing the $H^2$ structure.

Because the randomized SVD's described in Algorithm 1 are used to operate on existing $H^2$s (likely provided by a prior ACA build call) and not on $Z$ directly, the randomized SVD can be used to efficiently sample $Z$ for a specific far block via its $H^2$

representation and provide that block's SVD representation. To perform this same block-specific building seen in the ACA, the RSM utilizes existing $H^2$s and multiplies them by $\Omega$ through the use of the adapted $H^2$ multiplies discussed in Chapter 4. Using a $\Omega$ that is organized in the form discussed in Section 4.1, the multiplies, as described in Section 4.2, operate only the desired groups during its execution. The results of these multiplies are in the form of a block column matrix that is also organized in the same group-based manner as $\Omega$. This resulting block column matrix have row-blocks that are organized to operate on the desired target receiver groups passed to the multiplies as an argument and the columns of the block column matrices operate on a dimensionally reduced basis that was applied by using the random sampling matrix to estimate the range of the target source groups. Therefore, through leveraging the indexing paradigms and multiplies discussed in Chapter 4, the $H^2$-based randomized SVD can be used to efficiently sample $Z$ for specific blocks via operations on its $H^2$ representation without the risk of abundant, wasteful operations and proceed to produce the blocks' SVD representations.

Also note that if $\Omega$ were to be an identity of appropriate size, then the matrix returned by the adapted $H^2$ multiplies would be the exact same matrix that would result from sampling a dense representation of $Z$ for the rows and columns that correspond to the target groups at the target level. This observation can be used to extract sections of the matrix without performing an SVD-based truncation/approximation.

# Chapter 6 Random Sampling Method

The random sampling method (RSM) is a modified $H^2$ randomized SVD that can be used to build a new $H^2$ structure that results from the intermediate application of algebra on preexisting $H^2$ matrices. The RSM relies on the randomized SVD as its RandNLA foundation for three major reasons. First, the multiplication against $\Omega$ allows for the extraction of dense samples of the system matrix $Z$ that is represented by the $H^2$ that correspond to specific groups at any level of the decomposition. Second, these samples of $Z$ are dimensionally reduced, which allows for efficient manipulation of the extracted samples. Finally, the algorithm presents many intermediate opportunities for added operations in the dimensionally reduced space and, as seen in Section 3.1, through dimensional expansion is able to still produce an equivalent $U$, $\Sigma$ and $V^H$ decomposition to the decomposition that a more expensive deterministic SVD would produce.

The RSM has been designed to seamlessly fit in to the place of the ACA fill method in an $H^2$ fill, but it does not replace the ACA fill. This is because the RSM and ACA fill methods have completely separate use cases, both of which result in the construction of an $H^2$ data structure. So, the same code base is now able to fill an $H^2$ from scratch by using the ACA, and it is able to fill an $H^2$ that is obtained as the result of algebraic operations on existing $H^2$s using the RSM.

The RSM build method's noninvasive design that allows it to fit in the same code flow as ACA fill methods allows for the introduction of powerful $H^2$ algebra functionality without alienating developers by requiring them to learn a new, complicated library that performs $H^2$ algebra. Additionally, this ability to perform $H^2$ algebra without invasive changes to the code base in $O(k^2 N \log N)$ time complexity allows for easier integration of $H^2$ algebra in applications. Calling the RSM to perform $H^2$ algebra on existing $H^2$s is as simple as constructing a new, simple data structure called Meta $H^2$, which will be discussed in Section 6.1, and passing the structure to the same $H^2$ fill routine that has always been called to fill a new $H^2$. Now, the existing fill routines identify when a meta $H^2$ structure is present and automatically branch into the RSM-based fill methods. If no meta $H^2$ is passed to the fill routines, then they know that a simulation is building an $H^2$ from scratch using the ACA and the code automatically branches into the appropriate code path.

## 6.1 Meta $H^2$ Structure

The RSM currently has the functionality to multiply existing $H^2$s on the right by diagonal matrices, multiply existing $H^2$s on the left by block diagonal matrices, and add/subtract existing $H^2$s. Commands are administered to the RSM by passing a meta $H^2$ data structure to the standard fill routine call that already existed in the code base. Once passed to the RSM, the meta $H^2$ will be treated as the following

equation:

$$H^2_{RSM} = \sum_{n=1}^{N} P[n]\ H^2[n] A[n] \tag{6.1}$$

Where $H^2_{RSM}$ is the new $H^2$ data structure that is to result from the RSM's execution, $P[n]$ is a block diagonal matrix, $H^2[n]$ is a preexisting $H^2$ data structure one would like the operate on, and $A[n]$ is a diagonal matrix.

$$P[n]\ H^2[n]\ A[n]$$

is a single *component clause* in the meta $H^2$. The meta $H^2$ sums all of the passed component clauses together enabling the addition and subtraction of preexisting $H^2$ data structures.

The meta $H^2$ data structure has the following fields that the developer is required to fill:

```
struct {
    int N //numberOfComponentClauses;
    struct H2* H2[N];
    struct diagMatrix* A[N];
    struct blkDiagMatrix* P[N];
    struct octree* CompTree[N];
} metaH2;
```

**Algorithm 2:** Meta H2

The meta $H^2$ data structure seen in Algorithm 2 holds the number of component clauses (seen as N in (6.1) also) and four length N arrays of pointers. These arrays of pointers reference all the operands needed for the RSM to execute (6.1). These required operands are $H^2$ structures, block diagonal matrices, and diagonal matrices which are seen as $H2[]$, $P[]$, and $A[]$ respectively in (6.1) and Algorithm 2. Additionally, an array of pointers referencing the octrees used for indexing the $H^2$ data structure given through $H2[]$ must be provided in the meta $H^2$ data structure as $CompTree$. Currently, all $H^2$ given to the meta $H^2$ must use identical octrees. Therefore, $CompTree$ can currently be implemented as a single pointer to a common octree.

Now, the simplicity and familiarity that the developer experiences when wanting to leverage the RSM to perform powerful and efficient algebra on $H^2$ data structures that was mentioned in the Chapter 6 introduction can be thorough described. Because of the non-invasive design of the RSM build method, it is not necessary for the developer to learn a new set of libraries, helper routines, or rewrite any of their existing code in order to leverage the new $H^2$ algebra in the form of (6.1). The developer only needs to understand *i*) the general structure of the newly added meta $H^2$ data structure, *ii*) that the build routine they have always used to construct $H^2$ representations from scratch can now efficiently compute (6.1) using existing $H^2$ representations, and *iii*) the functionality to compute (6.1) is executed by passing the meta $H^2$ data structure as an argument to the those familiar build routines. In

other words, to compute (6.1) on a desired set of existing $H^2$ representations, the developer only needs to construct a meta $H^2$ that points to the desired operands for each component clause and pass that meta $H^2$ as an optional argument to the same build routine they have traditionally called to build an $H^2$ representation. By detecting the optional presence of the meta $H^2$, the build process will internally handle all of the intricacies involved with operating on existing $H^2$ data structures to perform $H^2$ algebra in the form of (6.1) via the RSM . Therefore, the developer is now able to leverage the power behind being able to perform efficient $H^2$ algebra by only needing to learn a new data structure, of which is just composed of pointers to a superset of already familiar data structures, and utilize the same build routines that they always use to acquire $H^2$ representations.

### 6.2 Performing $H^2$ Algebra via Random Sampling Method

Algorithm 1 will now be expanded into the full RSM:

**input :** $metaH2$, srcGrps, rcvGrps, level, tolerance
**output:** $U$,$\Sigma$, and $V^H$
;
Determine the number of random vectors, K, by finding the max rank of the rcvGrps U blocks out of all of the $metaH2.CompH2s$ and adding a margin of 5 to that value;
;
$Sum = ZeroMatrix$;
**for** $i \leftarrow 1$ **to** $metaH2.N$ **do**
  $\Omega$ = random(numOfDofs(srcGrps), K);
  ;
  $H2_i = metaH2.H2[i]$;
  $A_i = metaH2.A[i]$;
  $P_i = metaH2.P[i]$;
  //get $H^2$(AO) (multiply AO first because vector - block matrix multiply is quick compared to $H^2$A. Therefore, $H^2$(AO) is faster than ($H^2$A)O) ;
  $Mat_{H^2AO}$ = H2_rightMultiply($H2_i$, ($A_i\Omega$), rcvGrps, srcGrps, level);
  ;
  //get P$H^2$AO;
  $Mat_{PH^2AO} = P_i Mat_{H^2AO}$;
  ;
  //continue to sum all PH2AOs for all metaH2 clauses;
  $Sum = Sum + Mat_{PH^2AO}$;
**end**
;
//get truncated SVD representation so you only need to retain $O(KN)$ independent pieces of information;
[$\hat{u}$, $\hat{s}$, $\hat{v}^H$] = truncatedSVD($Sum$, tolerance);
$Q = \hat{u}$;
;
//Clear out Sum;
$Sum = ZeroMatrix$;
**for** $i \leftarrow 1$ **to** $metaH2.N$ **do**
  $H2_i = metaH2.H2[i]$;
  $A_i = metaH2.A[i]$;
  $P_i = metaH2.P[i]$;
  //sum all (($Q^H$P)$H^2$)A across all metaH2 clauses;
  $Mat_{Q^HPH^2}$ = H2_leftMultiply(($Q^H P_i$), $H2_i$, rcvGrps, srcGrps, level);
  $Sum = Sum + Mat_{Q^HPH^2} A_i$;
**end**
;
[$\bar{U}$, $\Sigma$, $V^H$] = svd($Sum$);
$U = Q\bar{U}$;
;
return $U$,$\Sigma$, and $V^H$;

**Algorithm 3:** RSM Algorithm

One important idea to state about the RSM is that it can be used to build both far and near interaction blocks. In order to build near blocks with the same RSM algorithm (Algorithm 3), it is as simple as using an identity in place of the random sampling matrix $\Omega$. This near block build counterpart to the RSM has been named the *identity sampling method* (ISM). Using an identity allows for basic dense operations on the near blocks of the existing $H^2$, because the blocks of the $H^2$ are being extracted out by multiplying them by identity matrices. Therefore, no dimensionality reduction via random sampling is occurring when operating on near blocks.

The general idea behind the RSM is better understood when it is broken down in steps and explained in plain English. The following list attempts to supply this step by step explanation:

Step 1: Generate $\Omega$ which is a set of $K$ random vectors, forming a $NumOfSrcDofs \times K$ matrix where $NumOfSrcDofs$ is the number of source degrees of freedom that are encapsulated by all the groups operated on at a given level. Determining the value of $K$ is discussed in Section 6.3. As discussed in Section 3.1, $\Omega$'s objective is to randomly sample the column space of targeted groups in order to acquire an estimation of the range of the target groups while also providing some convenient dimensionality reduction. The better the targets' column spaces' minimum spanning bases are extracted during random sampling, the more effective and less erroneous the dimensionality reduction process and the derivation of a range spanning orthogonal basis, $Q$, will be later in the algorithm.

Step 2: Apply the postconditioning diagonal matrix, $A$, that the developer wishes to multiply to Z via $H^2$ operations on $\Omega$ first. Because the matrix is diagonal, applying to $\Omega$ does not influence the random sampling's effectiveness. Call this new matrix $A\Omega$.

Step 3: Convert $A\Omega$ to its proper block column matrix equivalent that was discussed in Section 4.1 in order to then multiply $A\Omega$ on the $H^2$'s right side using the new multiply routines discussion in Chapter 4. This product will now sample the proper groups of degrees of freedom of $Z$ from the $H^2$'s compressed structure. Additionally, dimensionality reduction will be applied to this sampling. Call the result of this product $ZAO$.

Step 4: The $H^2$ multiply will actually supply $ZAO$ in block column form. This form makes the left multiplication of the preconditioning block diagonal matrix, $P$, simple. Apply the preconditioning block diagonal matrix, $P$, that the developer has supplied using a simple block matrix multiply onto $ZAO$ and call this product $PZAO$.

Step 5: Repeat all of the previous steps for all of the component clauses given in the meta $H^2$ and add all of the resulting $PZAO$s together. We have to go ahead and apply the summation before we take the SVD as the SVD lacks the distributive property. This SVD is necessary to find a matrix $Q$ whose columns form an orthonormal basis for the range of the summations of all of the $PZAO$s. A truncated SVD is applied here, because only the most significant singular values are needed for the construction of the new $H^2$.

Step 6: Now starts the process of applying the orthonormal basis for dimensionality reduction as discussed in Section 3.1. This consists of computing $((Q^H P)Z)A$ for every component clause of the meta $H^2$ and adding all of those products together.

Step 7: Finally an SVD can be taken of the sum of the $((Q^H P)Z)A$s. Then the orthonormal basis must be applied to the $U$ resulting from that SVD in order to recover excluded singular vectors as discussed in Section 3.1.

### 6.3 Determining K-Value Dynamically

In order to achieve dimensionality reduction and maintain controllable error introduction, the sampling matrix would have to be either the minimal spanning basis of your target matrix, $Z$ for this discussion, or a set of $K << N$ random, orthogonal vectors. Determining either of those matrices in this setting is far too expensive. Therefore, randomness is relied upon to try to quickly generate a set of vectors that span the basis of the target matrix, $Z$, well enough to only introduce minimal error. The number of random vectors in the random sampling matrix $\Omega$ will determine how likely the minimum spanning bases are captured in the random sampling that occurs when multiplying the target matrix, $Z$, by $\Omega$. The more random vectors generated, the more likely all of the basis of the target matrix will be covered by the sampling, thereby resulting in low error. Yet, a high number of random vectors will also drive up computation costs.

It was found that rather than statically defining the value of K, adding 5 to the maximum rank of the rcvGrps' U blocks, given by the preexisting $H^2$s that the RSM operates on, at the target level served as a great approximation of an ideal number of basis to attempt to generate. The maximum rank of the U blocks give a good idea of the rank of the interaction blocks being targeted. Therefore, by adding a small margin, it is hoped that enough orthogonality will be captured in the random vectors in order to span the basis well enough to achieve low error and reasonable runtime performance.

## Chapter 7 Results and Analysis of Testing the RSM

All of the following tests were executed on an i7-7700K at 4.2GHz using 8 OpenMP threads. In the following sections, different examinations of the RSM's performance will be analyzed by testing it on a locally corrected Nyström discretization of a magnetotstatic volume integral equation [16] solution of the fields in four different geometries: a thin steel strip, an imprinted plate, a steel shell with an outer radius of 0.25m and inner radius of 0.15m, and a large nonconformal steel shell. All of these test cases used a fill tolerance of $10^{-6}$. Therefore, an error close to the order of $10^{-6}$ is satisfactory performance for the RSM fill method. For all of these cases, the RSM was assigned through the meta $H^2$ structure to just multiply the original $H^2$ built by the ACA by identity matrices. The relative RMS errors reported are the relative RMS errors that resulted from comparing the RSM-built $H^2$ to the original ACA-built $H^2$. These results are used to analyze the base behavior of the RSM.

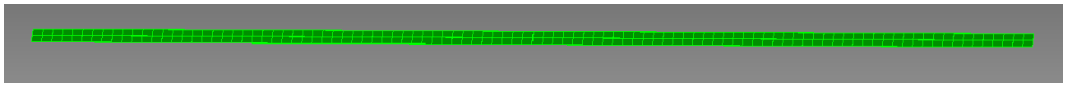


Figure 7.1: Thin Steel Strip

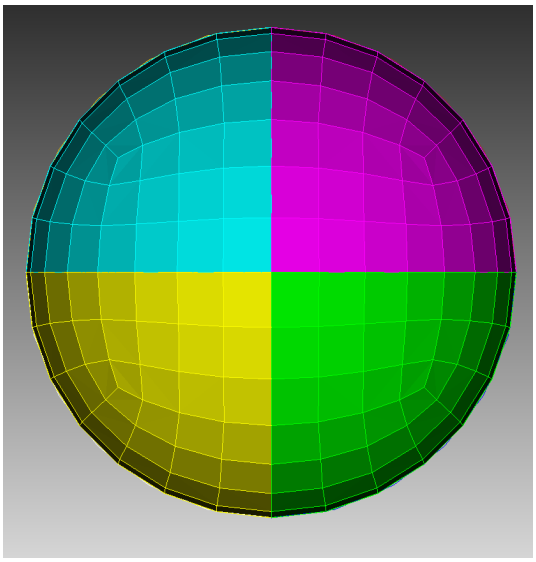


Figure 7.2: Steel Shell with 0.25m radius

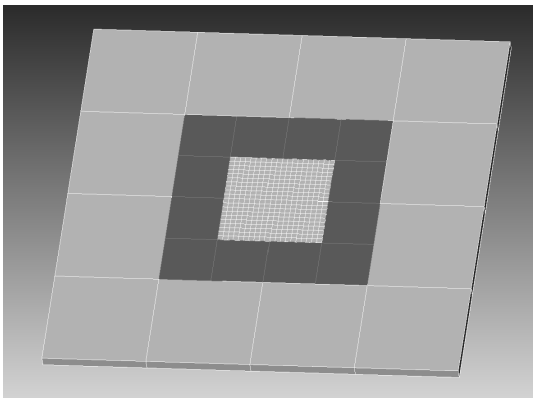


Figure 7.3: Imprinted Plate

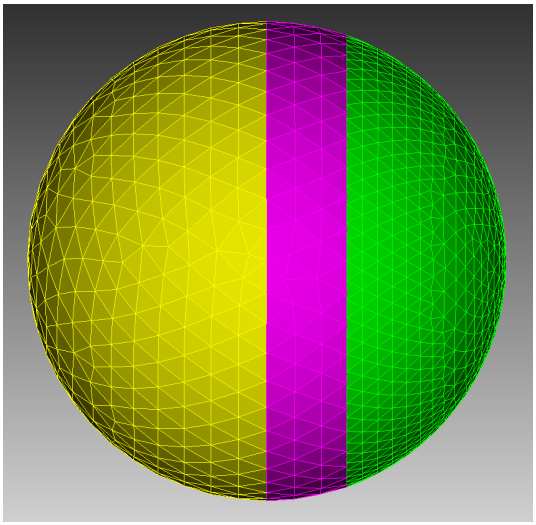


Figure 7.4: Nonconformal Steel Shell

## 7.1 Timing of the RSM and ACA Fills vs Number of Degrees of Freedom

In Figure 7.5, the total fill times of the RSM and the ACA are compared for varying sized test cases using a 5 level tree. It can be seen that the RSM is consistently 1 to 2 orders of magnitude faster than the corresponding ACA fill. This is because the RSM does not have to rely on standard matrix fill methods like method of moments or Nyström Method.

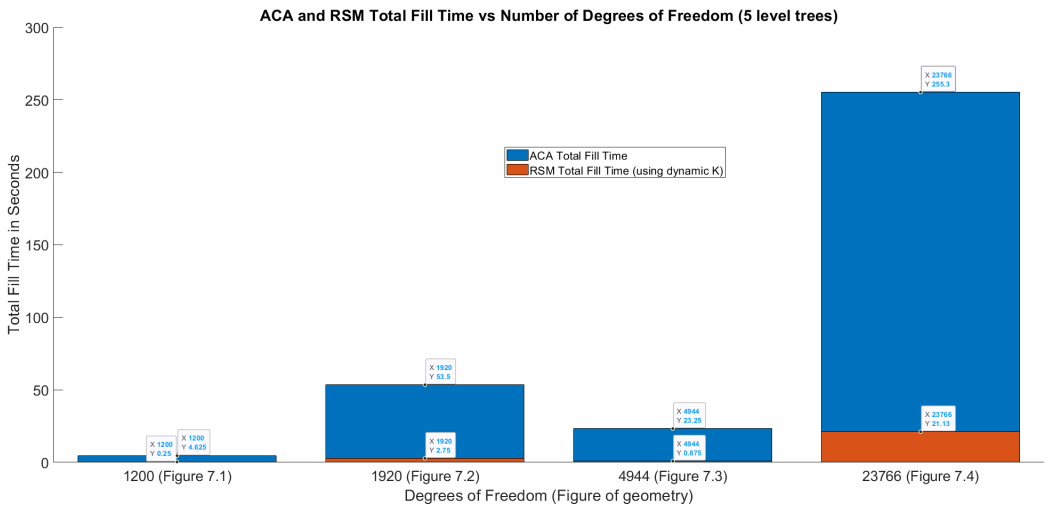



Figure 7.5: ACA and RSM Fill Time vs Degrees of Freedom

## 7.2 Dynamic K Error vs Number of Degrees of Freedom

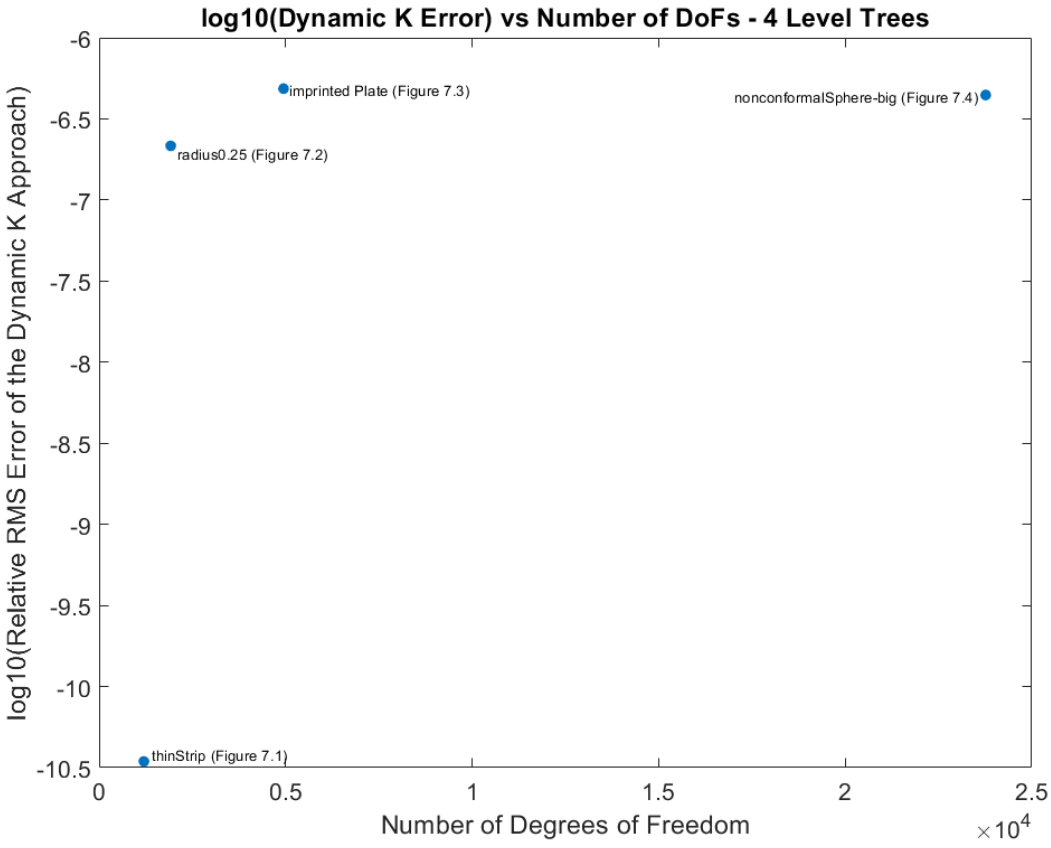



Figure 7.6: Dynamic $K$ error vs number of degrees of freedom for level 4 trees

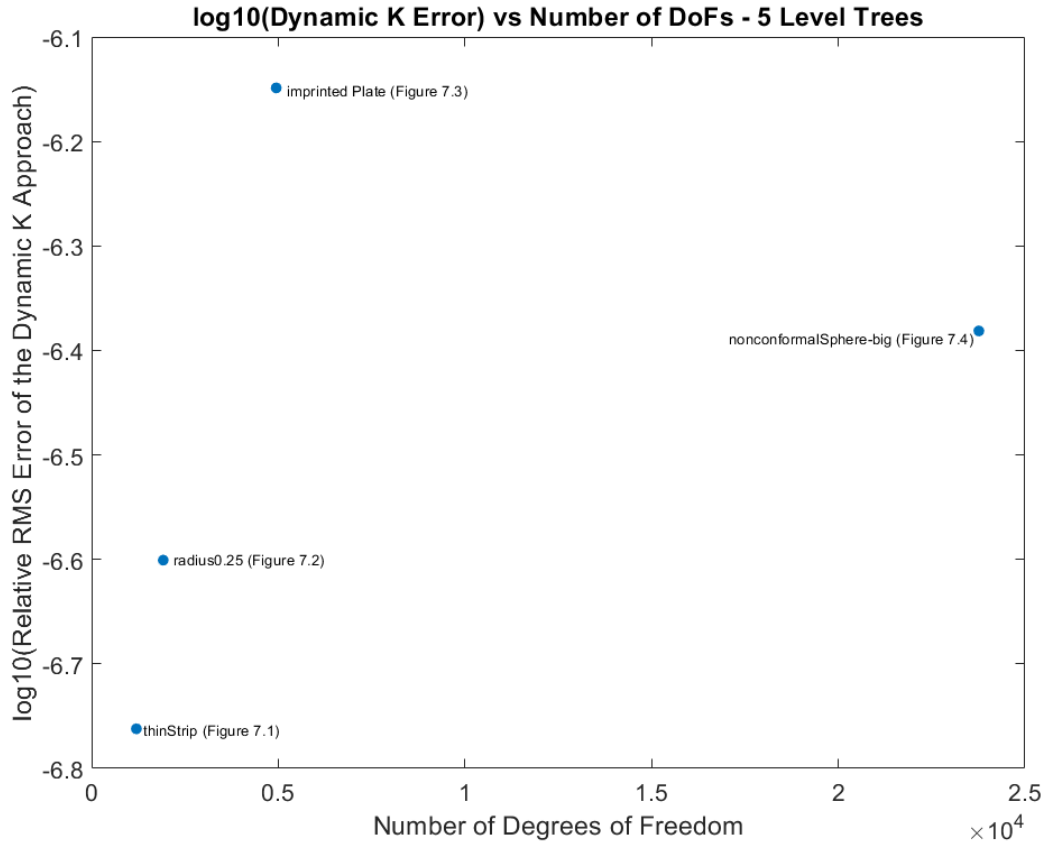



Figure 7.7: Dynamic $K$ error vs number of degrees of freedom for level 5 trees

In Figures 7.6 and 7.7, the reconstruction error under the usage of the dynamic $K$ value found via the method discussed in Section 6.3 vs the number of degrees of freedom in the test case is compared. It can be seen that in all of these cases for both a 4 and 5 level octree, the errors remain smaller in order of magnitude than the acceptable error order of magnitude of $10^{-6}$.

### 7.3 Error of Dynamic K and Static Ks

Figures 7.8 through 7.12 are comparing the error of static set $K$ values versus the dynamically decided $K$ value. The RSM allows for the developer to statically set a $K$ value via software options. When $K$ is statically defined by the developer, the RSM stops determining the $K$ value dynamically and uses the developer-defined $K$ at every level and for every group of the RSM build process. This functionality of being able to define a static $K$ value and stop the usage of a dynamic $K$ value allows for the observation of how well the methodology of determining $K$ dynamically (discussed in Section 6.3) is performing. In all of these plots, it is important to note that the error when using the dynamic $K$ value is roughly the same order of magnitude as the error for a statically set $K$=1000, but because of the size of these test cases, we know that

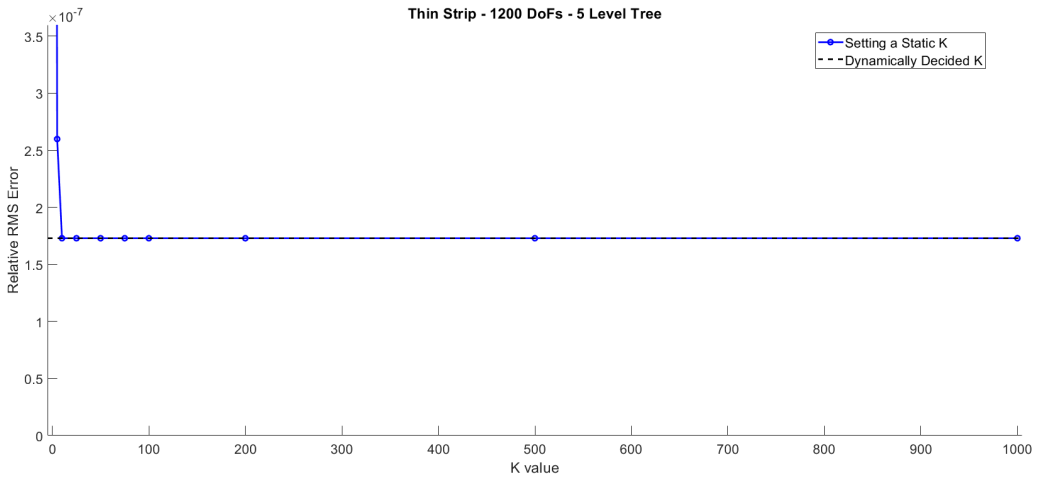



Figure 7.8: Comparing static $K$ errors and dynamic $K$ error

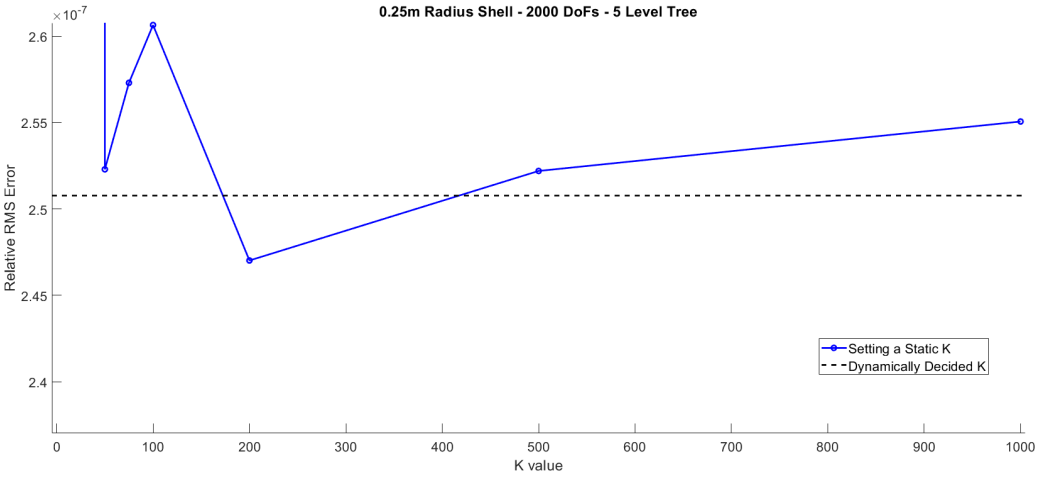



Figure 7.9: Comparing static $K$ errors and dynamic $K$ error

the dynamic $K$ values found are certainly less than 1000. Therefore, the method used to determine the dynamic $K$ seems to be providing $K$ values that perform adequate sampling of the target matrix without wastefully oversampling the target matrix. This claim is strongly supported by the results discussed later in Section 7.5.

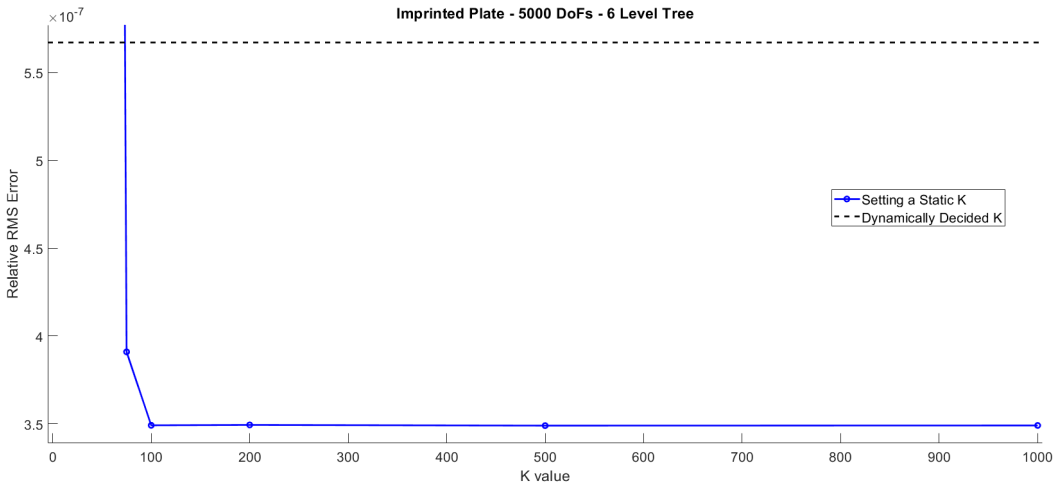



Figure 7.10: Comparing static $K$ errors and dynamic $K$ error

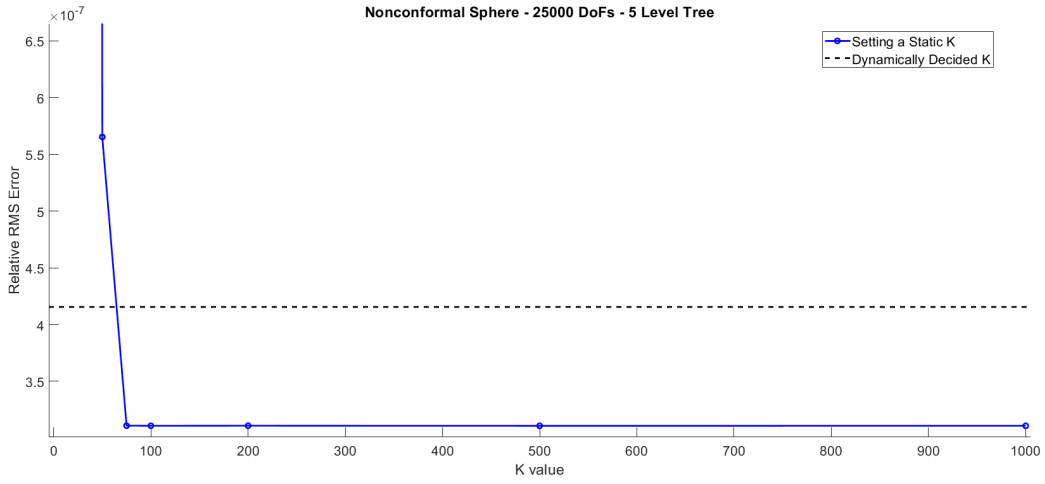



Figure 7.11: Comparing static $K$ errors and dynamic $K$ error

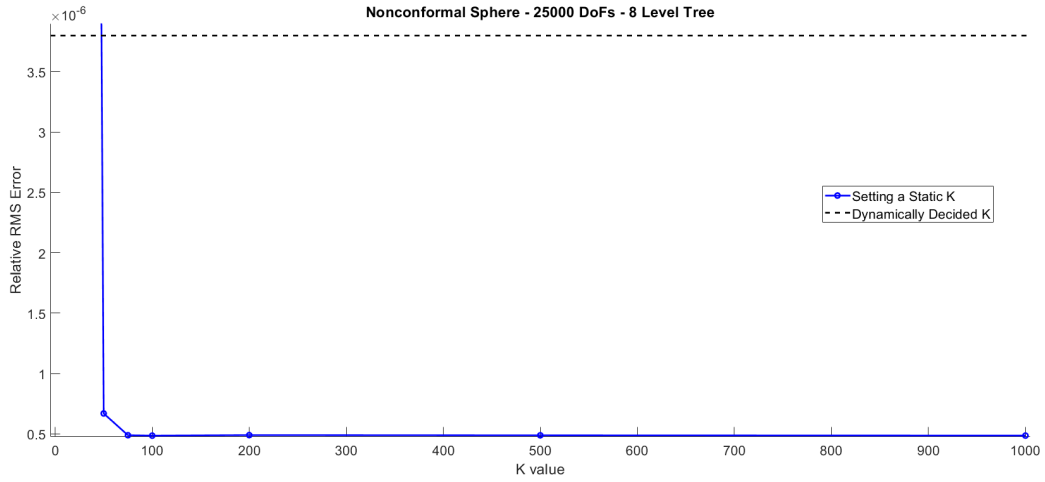



Figure 7.12: Comparing static $K$ errors and dynamic $K$ error

## 7.4 Dynamic K Error vs Tree Depth

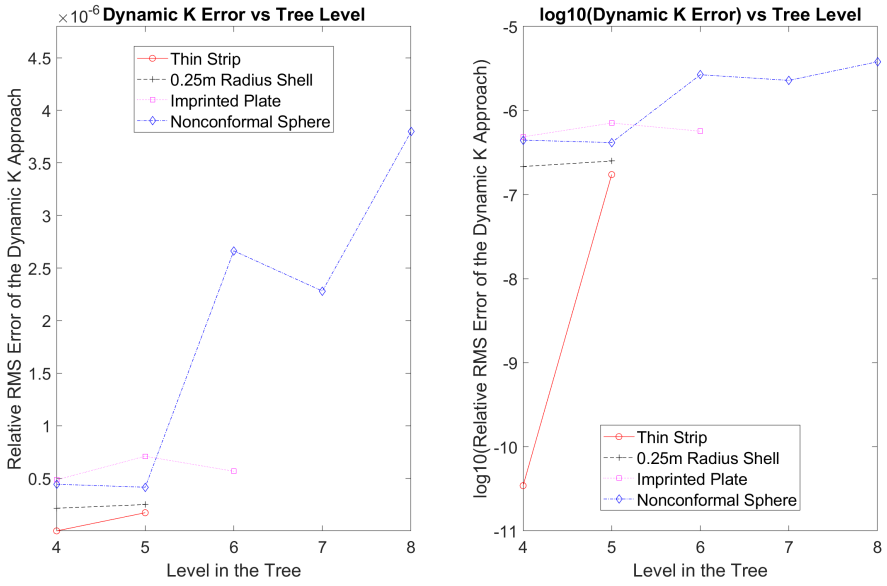



Figure 7.13: Dynamic K vs Tree Level

In Figure 7.13, the behavior of the error introduced when using the dynamic K value as tree level increased can be seen. It can be seen that the error for all of the test cases hovers around an order of magnitude of $10^{-6}$. This plot does suggest that there is some error being introduced in the level 8 nonconformal sphere test. This error is not high enough to suggest any issues in the RSM. Numerical round off is a likely culprit for this increase in error.

## 7.5 Comparison between Error and Time Savings of K-Values

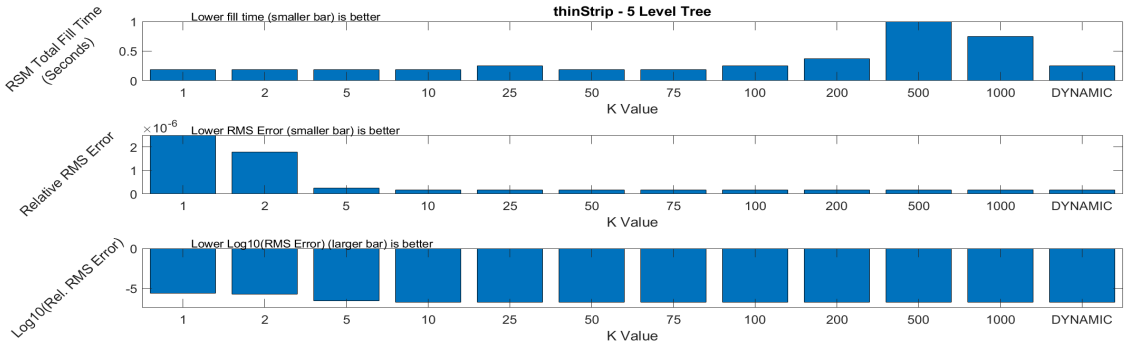



Figure 7.14: Runtime - error tradeoff for thin strip with a 5 level tree

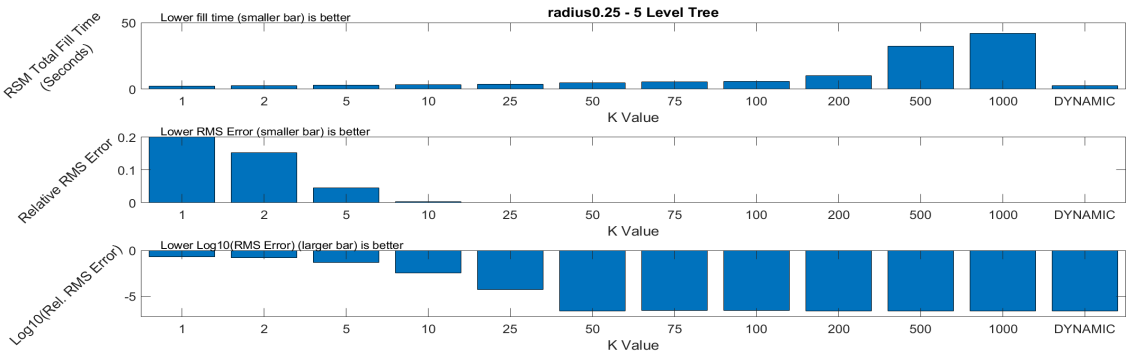



Figure 7.15: Runtime - error tradeoff for steel shell of radius 0.25m with a 5 level tree

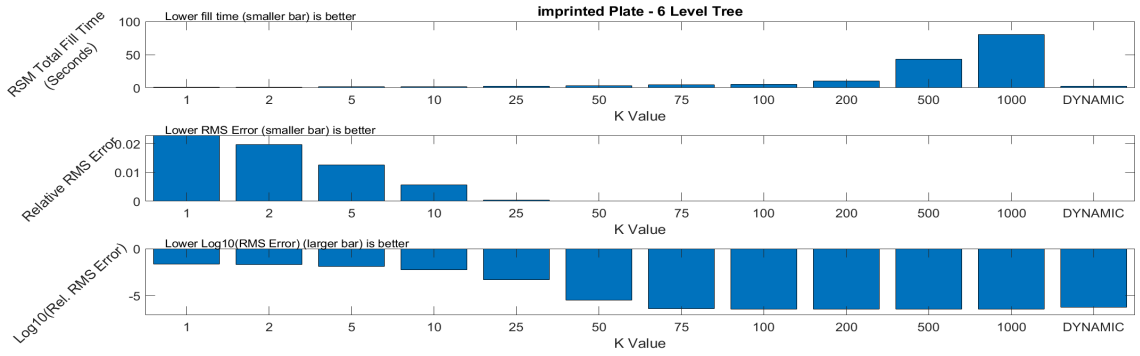



Figure 7.16: Runtime - error tradeoff for imprinted plate with a 6 level tree

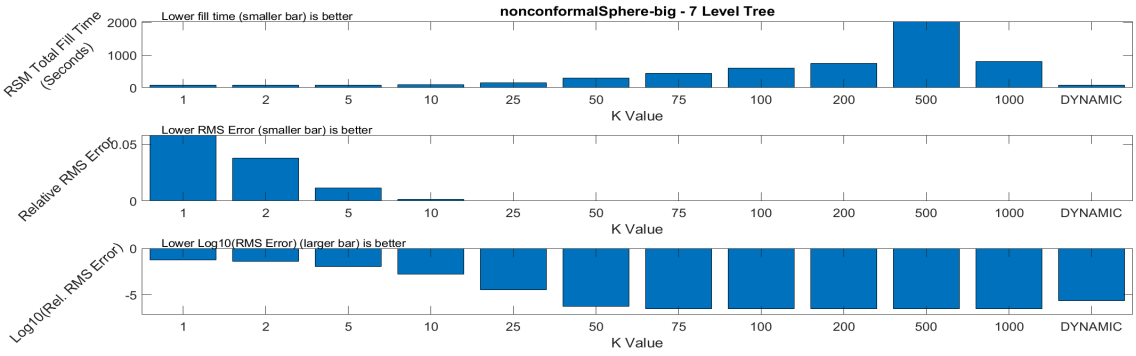



Figure 7.17: Runtime - error tradeoff for nonconformal sphere with a 7 level tree

The collection of figures in this section share some of the most important knowledge acquired from the analysis. On each plot, $K$ values (static and dynamic) are plotted on the x-axis, the RSM total fill time for each $K$ value is plotted on the top y-axis, the relative RMS error introduced by each $K$ values is plotted on the middle y-axis, and the relative RMS error is plotted again on the bottom y-axis on a log scale. For all of the statically defined $K$ values, it can be seen that a $K$ value that is small may have low fill time, but accomplishes this low fill time by risking undersam-

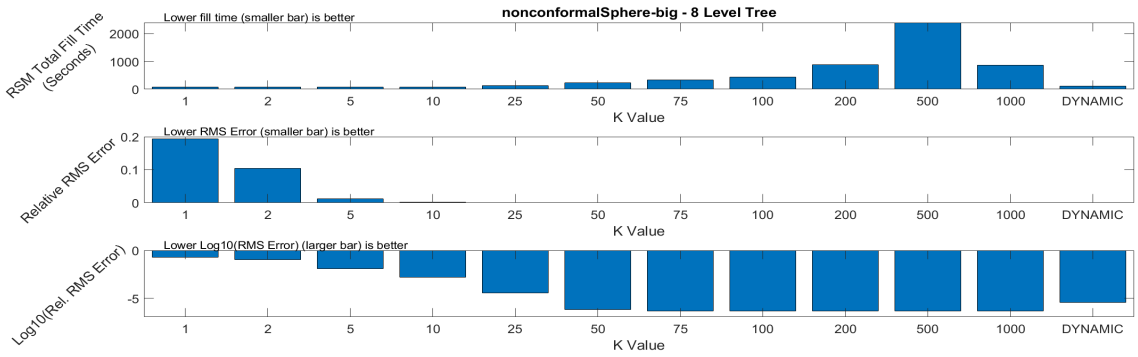



Figure 7.18: Runtime - error tradeoff for nonconformal sphere with a 8 level tree

pling of the target matrix. *Undersampling* means that there were not an adequate number of random vectors used to accurately estimate the range of $Z$ via random sampling, therefore causing high error introduction for the RSM. Additionally, a $K$ value that is large may have low error introduction, but accomplishes this low error by requiring more operations and risking oversampling of the target matrix. A high $K$ value already harms fill time because more operations are required by the multiplies within the RSM to apply the random sampling of $Z$ using a large number of random vectors. Yet, a large number of random vectors may also be sampling redundant basis when randomly sampling $Z$, this very wasteful behavior is called *oversampling*. So, the goal of the dynamic $K$ is to not only avoid undersampling and oversampling at all cost, but also to attempt to maintain a favorable, if not optimal, ratio of total fill time and error introduction. In all of these plots, it is seen that the usage of dynamic $K$ corresponds consistently to the lowest error AND lower total fill time when compared to any of the statically set $K$ values. This shows that the presented approach in Section 6.3 for determining $K$ dynamically is providing a good balance between fast execution and low error levels.

## 7.6 RSM Truncation Tolerance Error Control

Additional tests were run to ensure that the error introduced by the RSM can be controlled to a saturation point. To test this, different multipliers were applied to the SVD truncation tolerance that the RSM is using. 13 multipliers ranging from 0.001 to 10000 were tested, the multiplier value for each point in Figure 7.19 can be seen on as a label next to the data point. The curve seen in Figure 7.19 is the expected behavior. As the tolerance is loosened, more singular values are being dropped when forming the truncated representation within the RSM. Therefore, singular values that represent significant information are being dropped and not being used in the RSM constructed $H^2$ and causing the RSM $H^2$ to deviate from the original ACA built $H^2$.

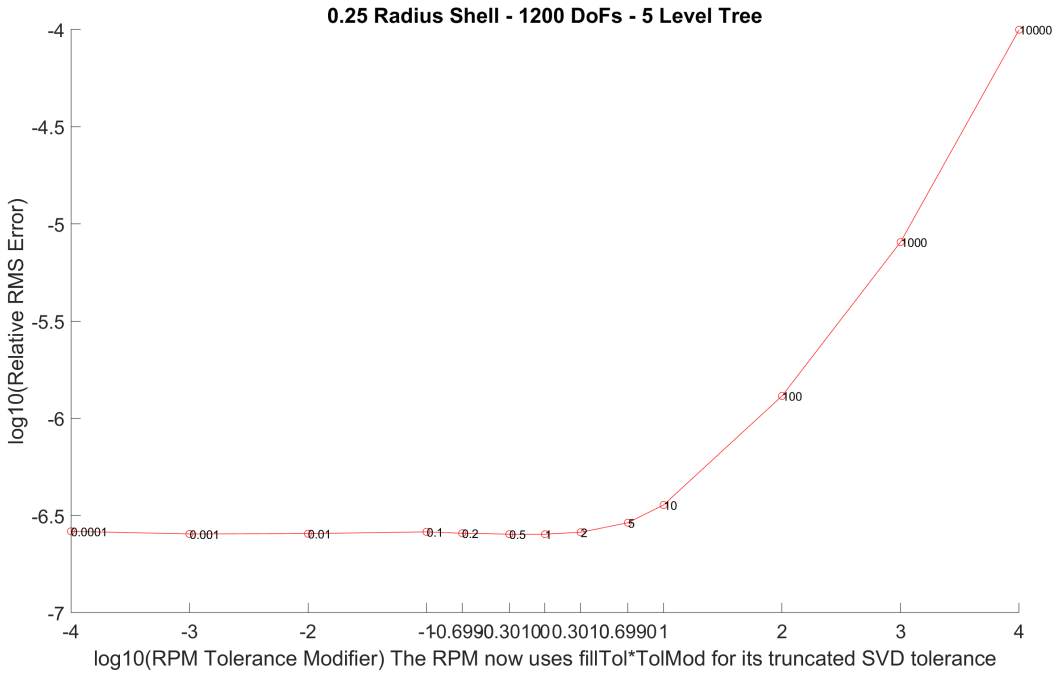



Figure 7.19: Error controllability through RSM truncation tolerance

## Chapter 8 Diagonal Factorization

In the pursuit of solving large problems in CEM, efficiently representing and storing the system matrix is only one component of the challenge. The resulting linear systems are often very difficult to solve. Applying a basic LU factorization or any other fundamental factorization to $Z$ in order to attempt to solve the system would be, in most cases, impossible. A system with a million degrees of freedom, which is a feasible size for modern problems, would take 14TB just to store $Z$. Additionally, an LU factorization has a runtime complexity of $O(N^3)$. Therefore, it would take a couple of months to perform an LU on a 1 million unknown system on a nice desktop PC, if you had the memory. Luckily, the $H^2$ data structure allows for the efficient storing of a 1 million degrees of freedom system using $\sim O(N^{1.5})$ floating-point numbers rather than the $O(N^2)$ required by the dense representation. And now, it is even possible to easily perform $H^2$ algebra in $O(k^2 N \log N)$ time complexity. Yet the system still needs to be solved, and before solving can be performed, factoring of the $H^2$ still needs to be done in a computationally efficient manner. Additionally, the approach used in the factorization will greatly impact performance in the solve.

UKCEM has developed $O(N)$ sparse direct methods to directly factor compressed representations of $Z$ using localizing sources, see section 2.6. This factorization method uses the physical concept of localization to develop an efficient linear algebraic factorization for $Z$ which maintains the sparseness of $Z$. As mentioned in section 2.6, the submatrix $Z_l^{(NL)} = (P_l^{(N)})^H Z_{l+1}^{(NN)} \Lambda_l^{(L)}$ is controllably small. This allows for a upper triangular form to be achieved, which allows an approximate matrix inverse to be recursively applied to the factored structure until only an LU or QR decomposition needs to be performed on $Z_2^{(NN)}$, which is computationally reasonable [22].

This upper triangular factorization can be further evolved. For the same reason that $Z_l^{(NL)}$ is approximately 0 in the upper triangular form, $Z_l^{(LN)}$ could also become approximately 0 if $\Lambda^{(N)}$ were to be replaced with $conj(P^{(N)})$. Using $\hat{\Lambda}_l = \begin{bmatrix} \Lambda_l^{(L)} & conj(P_l^{(N)}) \end{bmatrix}$ would yield

$$\begin{aligned} Z_{l+1}^{(NN)} &= (\hat{\Lambda}_l^T)^{-1} \hat{\Lambda}_l^T Z_{l+1}^{(NN)} \hat{\Lambda}_l \hat{\Lambda}_l^{-1} \\ &= (\hat{\Lambda}_l^T)^{-1} \begin{bmatrix} Z_l^{(LL)} & Z_l^{(LN)} \\ Z_l^{(NL)} & Z_l^{(NN)} \end{bmatrix} \hat{\Lambda}_l^{-1} \\ &\approx (\hat{\Lambda}_l^T)^{-1} \begin{bmatrix} B_l & 0 \\ 0 & Z_l^{(NN)} \end{bmatrix} \hat{\Lambda}_l^{-1} \end{aligned} \tag{8.1}$$

Yet, the addition of $conj(P^{(N)})$ as the orthogonal complement of $\Lambda^{(L)}$ now causes $\hat{\Lambda}$ to be ill conditioned in some cases. Therefore, this may cause error in reconstructing $Z$ from this new factored form. Conveniently, $B$ can be used to indirectly control the conditioning of $\hat{\Lambda}$. Because $B$ is determined by the localizing part of $\hat{\Lambda}$, then by taking the SVD of $B$ one can determine the parts of $\Lambda^{(L)}$ that should be kept to

control $\hat{\Lambda}$'s conditioning. Sampling $\Lambda^{(L)}$ for the subspace that corresponds to the well-conditioned parts of $B$ is as simple as postconditioning $\Lambda^{(L)}$ by $v_l$, where $v_l$ is found from $\text{svd}(B_l) = u_l \sigma_l v_l^H$. This postconditioning forms $\hat{\Lambda}_{B,l} = \left[\Lambda_l^{(L)} v_l \quad conj(P_{B,l}^{(N)})\right]$. The factorization then becomes

$$Z_{l+1}^{(NN)} \approx (\hat{\Lambda}_{B,l}^T)^{-1} \begin{bmatrix} \hat{B}_l & 0 \\ 0 & Z_l^{(NN)} \end{bmatrix} \hat{\Lambda}_{B,l}^{-1} \tag{8.2}$$

where $\hat{B}_l$ is controllably well conditioned.

Yet, there is one more convenience that allows for this method to be improved even further. If $Z$ is symmetric, $\hat{B}_l$ is also symmetric. Therefore, the factorization can be simplified one step further by performing a symmetric SVD on $\hat{B}_l$, which results in $\hat{B}_l = Q_l^H \Sigma_l Q_l = Q_l^H S_l S_l Q_l$ where $S_l$ is the $sqrt(diag(\Sigma_l))$. The factorization can then be completely diagonalized by using $\bar{\Lambda}_l = \left[\Lambda_l^{(L)} v_l Q_l^H S_l^{-1} \quad conj(P_{B,l}^{(N)})\right]$. This yields the block diagonal factorization of

$$Z_{l+1}^{(NN)} \approx (\bar{\Lambda}_l^T)^{-1} \begin{bmatrix} I & 0 \\ 0 & Z_l^{(NN)} \end{bmatrix} \bar{\Lambda}_l^{-1} \tag{8.3}$$

## 8.1 Test Case Comparing Diagonal Factorization and Upper Triangular Factorization

The upper triangular factorization and diagonal factorization have been tested and compared abundantly in [22], [31], and [18]. Here, the numerical results for a single test case that has been under investigation in some current work will be reviewed. The test case geometry is shown in Figure 8.1. Tables 8.1 and 8.2 show time reported in seconds and memory reported in GB. For this discussion, the $N_{it}$ and RR columns can be disregarded.

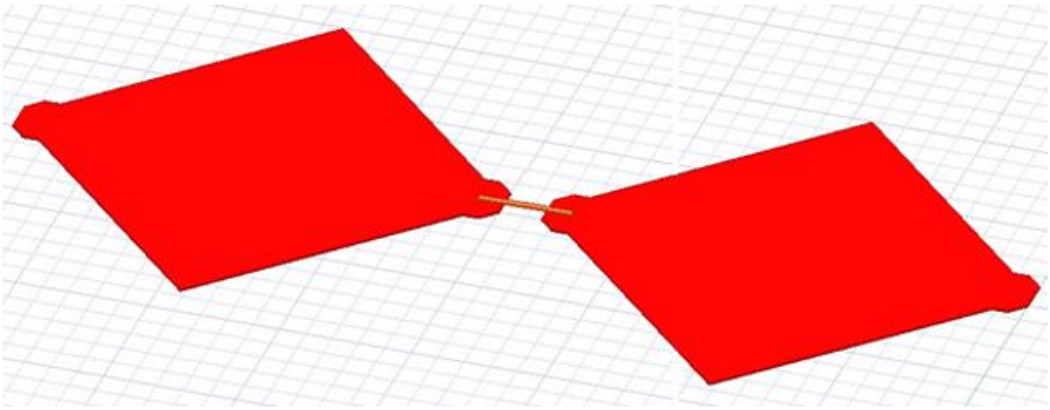


Figure 8.1: Two electrodes with a connecting bridge

It can be seen in Tables 8.1 and 8.2 that the diagonal factorization, although having a double factor time, consistently has a 4x speed up is solve time compared

| $\tau_{fac}$ | factor time | factor mem | $N_{it}$ | solve time | RR |
|---|---|---|---|---|---|
| 1e-1 | 1.2 | 0.04 | 25 | 4.4 | 3e-10 |
| 1e-2 | 3.8 | 0.15 | 9 | 1.8 | 2e-13 |
| 1e-3 | 9.2 | 0.34 | 9 | 2.0 | 8e-14 |
| 1e-4 | 17 | 0.63 | 5 | 1.3 | 1e-13 |

Table 8.1: Triangular factorization of circuit test case from Figure 8.1

| $\tau_{fac}$ | factor time | factor mem | $N_{it}$ | solve time | RR |
|---|---|---|---|---|---|
| 1e-1 | 3.9 | 0.12 | 26 | 0.98 | 1e-12 |
| 1e-2 | 8.0 | 0.17 | 13 | 0.6 | 1e-13 |
| 1e-3 | 19 | 0.37 | 9 | 0.5 | 7e-14 |
| 1e-4 | 34 | 0.67 | 5 | 0.38 | 9e-14 |

Table 8.2: Diagonal factorization of circuit test case from Figure 8.1

to the triangular factorization. The triangular factorization's increase in solve time is because of its need to invert via back-substitution. The diagonal factorization does not require any inversion or back-substitution until the coarsest level.

## 8.2 Implementation of the Diagonal Factorization

A diagonal factorization development map was provided in the appendix that was used during the implementation of the diagonal factorization into UKCEM's main modular fast direct (MFD) code base. Aside from setting up the needed surrounding code framework for the diagonal factorization, a significant amount of work was put in to transposing the ΘR algorithm [32]. The ΘR is a multilevel QR-like factorization that operates on the $H^2$ to ultimately take the QR of the column groups of $Z$. The ΘR is used to determine the localizing source degrees of freedom $\Lambda_l^{(L)}$ used in both the triangular and diagonal factorizations. Localizing receiver degrees of freedom can be used to achieve the diagonal factorization results by calculating the preconditioning factoring blocks of the diagonal factorization form rather than computing $(\hat{\Lambda}_l^T)^{-1}$ and $\hat{\Lambda}_l^T$. In order to get these localizing receiver degrees of freedom, a row based ΘR was needed. This was achieved by developing the RΘ, which is a ΘR that operates in the transpose space.

## Chapter 9 Summary

A randomized numerical linear algebra approach for applying $H^2$ algebra in a non-invasive manner and a diagonal factorization for $H^2$ representations have been detailed. It has been observed that RandNLA can be adapted to operate on $H^2$ matrices, which enables efficient $H^2$ algebra to be performed. It is also important to emphasize that the RandNLA methods outlined here can be non-invasively incorporated in an ACA-based $H^2$ code. This design choice allows the easy incorporation of powerful $H^2$ algebra into code bases without the need for major refactoring and maintains a familiar developer experience when leveraging this new functionality.

Although not explored herein due to time limitations, it is expected that the computational costs of the proposed RSM-based $H^2$ algebra methods will scale with a complexity of $O(k^2 N \log N)$, and reduction to $O(k^2 N)$ costs is likely possible. To achieve this complexity reduction, modification to the multiplies discussed in Chapter 4 would need to be done to enable the caching of finer level's far interaction sampling which is mathematically seen as operation on $\hat{V}^H$ and $\hat{U}^H$ in Figures 4.4 to 4.5. The build process constructs the $H^2$ representation from the finest decomposition level to the coarsest decomposition level. Additionally, the multiplies that the RSM relies on must decompose the target groups down to the finest level decomposition as operating on all finer level's $\hat{V}^H s$ and $\hat{U} s$ is required as discussed in Chapter 4. Therefore, it is likely that the same groups of finer level's $\hat{V}^H s$ and $\hat{U} s$ will be sampled multiple times through the RSM-based fill process. Therefore, caching these repeatedly sampled far interactions is likely to reduce the overall complexity for a factor of $O(\log N)$.

Additionally, a method to dynamically determine the number of random vectors required to apply random sampling and dimensionality reduction through random sampling in competitive time and competitive error was described. The success of this decision making rule was clearly observed. Yet, the context of this paper allowed for a convenient method for determining the dynamic number of random vectors by observing the rank of the preexisting $H^2$ matrices. The ability to determine a well-performing K-value is unlikely in most other applications. Most other applications wishing to determine an adaptive K-value to leverage in a random sampling must pursue a probabilistic approach, as described in [33].

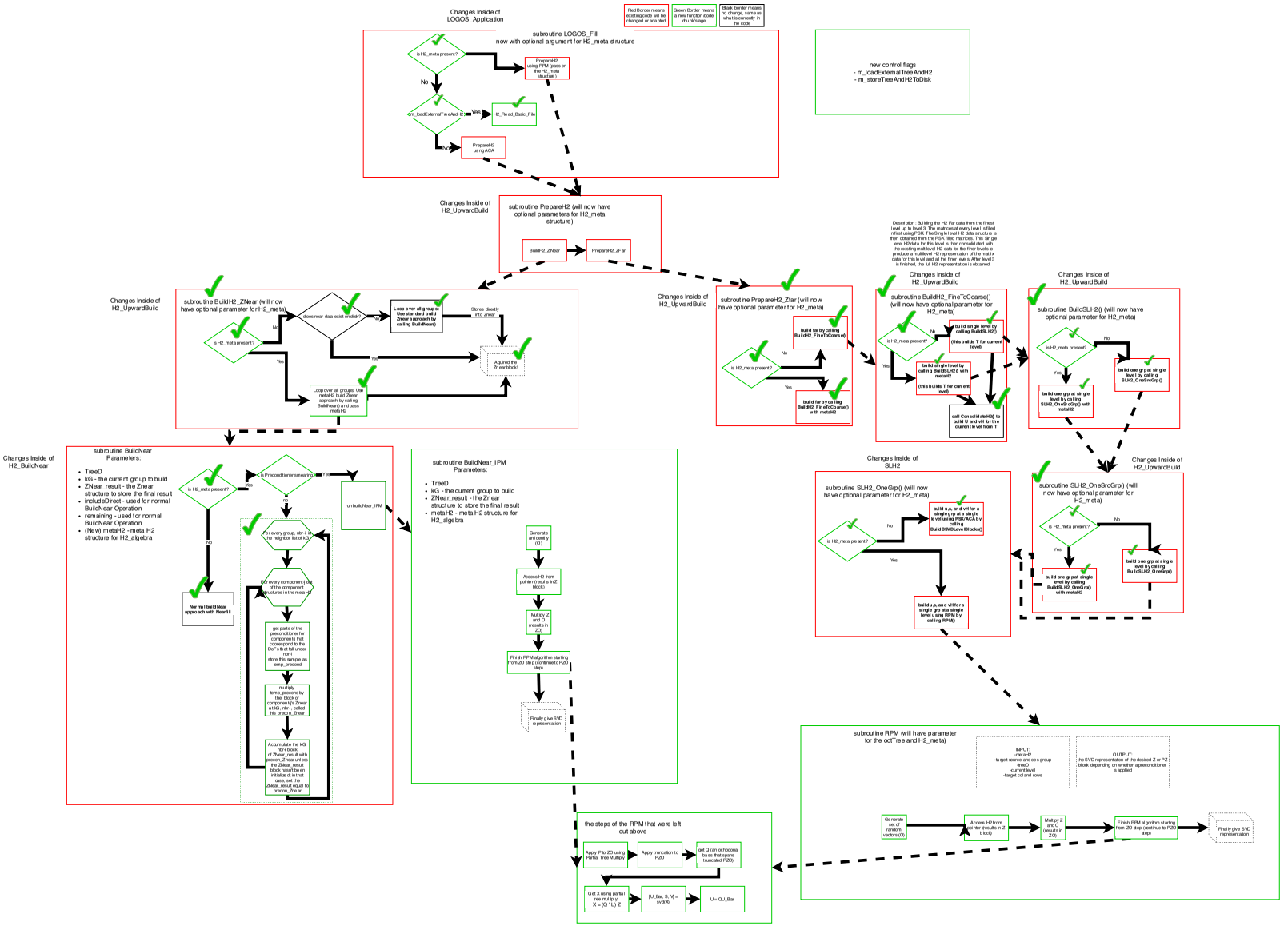
Finally, in separate work, it has also been observed that a diagonal, localization-based factorization is effective at reducing the time necessary to apply the resulting inverse approximations to excitation vectors [22] [31] [18].

Both of the advancements described in this document have demonstrated contributions to the effort of pursuing increasingly massive and computationally expensive problems in computational EM.

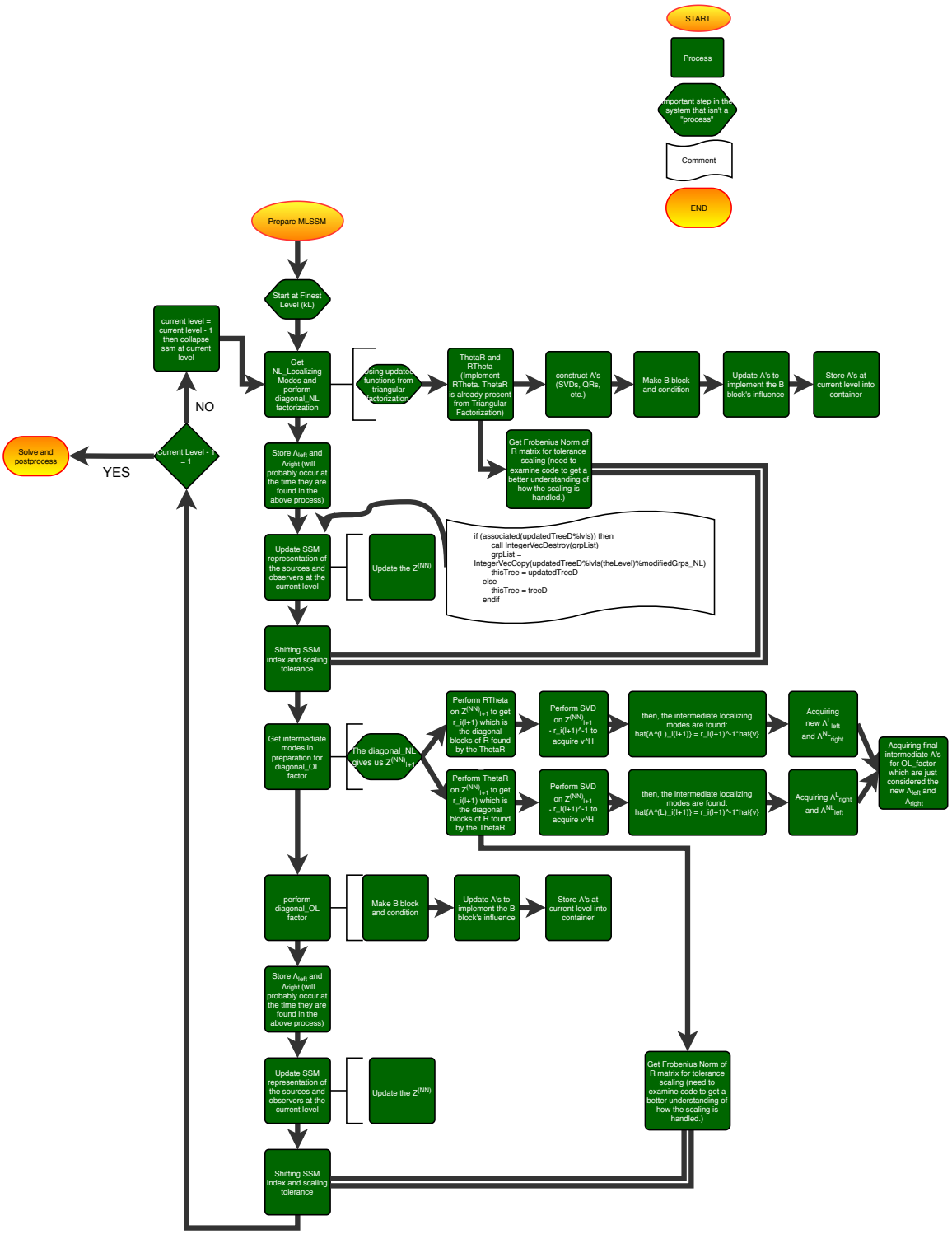
## Appendix

In this appendix, two PDFs are included showing flowcharts that were created during the development of the work discussed in this document to act as road maps for the software development process.

### RSM Development Map


Changes Inside of LOGOS_Application
Red Border means existing code will be changed or adapted
Green Border means a new function inside the HMatrix class
Black border means no change, same as what is currently in the code
subroutine LOGOS_Fill
now with optional argument for H2_meta structure
is H2_meta present?
Prepare H2 using RPM (pass on the H2_meta structure)
No
m_loadExternalTreeAndH2 ?
Yes
H2_Read_Basic_File
No
Prepare H2 using ACA
new control flags
- m_loadExternalTreeAndH2
- m_storeTreeAndH2ToDisk
Changes Inside of H2_UpwardBuild
subroutine PrepareH2 (will now have optional parameters for H2_meta structure)
BuildH2_ZNear
PrepareH2_ZFar
Description: Building the H2 Far data from the finest level up to level 3. The matrices at every level is filled in first using PSK. The Single level H2 data structure is then obtained from the PSK filled matrices. This Single level H2 data for this level is then consolidated with the existing multilevel H2 data for the finer levels to produce a multilevel H2 representation of the matrix data for this level and all the finer levels. After level 3 is finished, the full H2 representation is obtained.
Changes Inside of H2_UpwardBuild
subroutine BuildH2_ZNear (will now have optional parameter for H2_meta)
is H2_meta present?
No
Yes
does near data exist on disk?
Yes
No
Loop over all groups: Use standard build Znear approach by calling BuildNear()
Stores direcly into Znear
Loop over all groups: Use metaH2 build Znear approach by calling BuildNear() and pass meta H2
Aquired the Znear block!
Changes Inside of H2_UpwardBuild
subroutine PrepareH2_Zfar (will now have optional parameter for H2_meta)
is H2_meta present?
No
Yes
build far by calling BuildH2_FineToCoarse()
build far by calling BuildH2_FineToCoarse() with metaH2
Changes Inside of H2_UpwardBuild
subroutine BuildH2_FineToCoarse() (will now have optional parameter for H2_meta)
is H2_meta present?
No
Yes
build single level by calling BuildSLH2() with metaH2
(this builds T for current level)
build single level by calling BuildSLH2() (this builds T for current level)
call ConvertMatToH2 to build U and vhi for the current level from T
Changes Inside of H2_UpwardBuild
subroutine BuildSLH2() (will now have optional parameter for H2_meta)
is H2_meta present?
No
Yes
build one grp at single level by calling SLH2_OneSrcGrp()
build one grp at single level by calling SLH2_OneSrcGrp() with metaH2
Changes Inside of H2_BuildNear
subroutine BuildNear
Parameters:
• TreeD
• kG - the current group to build
• ZNear_result - the Znear structure to store the final result
• includeDirect - used for normal BuildNear Operation
• remaining - used for normal BuildNear Operation
• (New) metaH2 - meta H2 structure for H2_algebra
is H2_meta present?
Yes
No
Normal buildNear approach with NearFill
Is Preconditioner smearing?
Yes
No
run buildNear_IPM
for every group, nbr-i, in the neighbor list of kG
for every component j of the component structures in the meta H2
get parts of the preconditioner for component j that corresponsd to the DoF's that fall under nbr-i
store this sample as temp_precond
multiply temp_precond by the block of component j's Znear at kG, nbr-i, called this precon_Znear
Accumulate the kG, nbr-i block of ZNear_result with precon_Znear unless the ZNear_result block hasn't been initialized, in that case, set the ZNear_result equal to precon_Znear
subroutine BuildNear_IPM
Parameters:
• TreeD
• kG - the current group to build
• ZNear_result - the Znear structure to store the final result
• metaH2 - meta H2 structure for H2_algebra
Generate an identity (O)
Access H2 from pointer (results in Z block)
Multiply Z and O (results in ZO)
Finish RPM algorithm starting from ZO step (continue to PZO step)
Finally give SVD representation
Changes Inside of SLH2
subroutine SLH2_OneGrp() (will now have optional parameter for H2_meta)
is H2_meta present?
No
Yes
build u,s, and vhi for a single grp at a single level using PSK/ACA by calling BuildBSVDLevelBlocks()
build u,s, and vhi for a single grp at a single level using RPM by calling RPM()
Changes Inside of H2_UpwardBuild
subroutine SLH2_OneSrcGrp() (will now have optional parameter for H2_meta)
is H2_meta present?
No
Yes
build one grp at single level by calling BuildSLH2_OneGrp() with metaH2
build one grp at single level by calling BuildSLH2_OneGrp()
subroutine RPM (will have parameter for the octTree and H2_meta)
INPUT:
-metaH2
-target source and obs group
-treeD
-current level
-target col and rows
OUTPUT:
the SVD representation of the desired Z or PZ block depending on whether a preconditioner is applied
Generate set of random vectors (O)
Access H2 from pointer (results in Z block)
Multiply Z and O (results in ZO)
Finish RPM algorithm starting from ZO step (continue to PZO step)
Finally give SVD representation
the steps of the RPM that were left out above
Apply P to ZO using Partial Tree Multiply
Apply truncation to PZO
get Q (an orthogonal basis that spans truncated PZO)
Get X using partial tree multiply X = (Q' L) Z
[U_Bar, S, V] = svd(X)
U = QU_Bar

**Diagonal Factorization Development Map**


START
Process
important step in the system that isn't a "process"
Comment
END
Prepare MLSSM
Start at Finest Level (kL)
current level = current level - 1 then collapse ssm at current level
Get NL_Localizing Modes and perform diagonal_NL factorization
Using updated functions from triangular factorization
ThetaR and RTheta (Implement RTheta. ThetaR is already present from Triangular Factorization)
construct Λ's (SVDs, QRs, etc.)
Make B block and condition
Update Λ's to implement the B block's influence
Store Λ's at current level into container
Get Frobenius Norm of R matrix for tolerance scaling (need to examine code to get a better understanding of how the scaling is handled.)
Store Λleft and Λright (will probably occur at the time they are found in the above process)
Update SSM representation of the sources and observers at the current level
Update the Z(NN)
if (associated(updatedTreeD%lvls)) then
call IntegerVecDestroy(grpList)
grpList = IntegerVecCopy(updatedTreeD%lvls(theLevel)%modifiedGrps_NL)
thisTree = updatedTreeD
else
thisTree = treeD
endif
Shifting SSM index and scaling tolerance
Current Level - 1 = 1
NO
YES
Solve and postprocess
Get intermediate modes in preparation for diagonal_OL factor
The diagonal_NL gives us Z(NN)i+1
Perform RTheta on Z(NN)i+1 to get r_i(i+1) which is the diagonal blocks of R found by the ThetaR
Perform SVD on Z(NN)i+1 * r_i(i+1)^-1 to acquire v^H
then, the intermediate localizing modes are found: hat(Λ^(L)_i(i+1)) = r_i(i+1)^-1*hat{v}
Acquiring new Λ^L_left and Λ^NL_right
Perform ThetaR on Z(NN)i+1 to get r_i(i+1) which is the diagonal blocks of R found by the ThetaR
Perform SVD on Z(NN)i+1 * r_i(i+1)^-1 to acquire v^H
then, the intermediate localizing modes are found: hat(Λ^(L)_i(i+1)) = r_i(i+1)^-1*hat{v}
Acquiring Λ^L_right and Λ^NL_left
Acquiring final intermediate Λ's for OL_factor which are just considered the new Λleft and Λright
perform diagonal_OL factor
Make B block and condition
Update Λ's to implement the B block's influence
Store Λ's at current level into container
Store Λleft and Λright (will probably occur at the time they are found in the above process)
Update SSM representation of the sources and observers at the current level
Update the Z(NN)
Get Frobenius Norm of R matrix for tolerance scaling (need to examine code to get a better understanding of how the scaling is handled.)
Shifting SSM index and scaling tolerance

## Bibliography


[1] P. Drineas and M. W. Mahoney, "Randnla: Randomized numerical linear algebra," *Communications of the ACM*, vol. 59, no. 6, pp. 80–90, 2016.

[2] M. W. Mahoney *et al.*, "Randomized algorithms for matrices and data," *Foundations and Trends® in Machine Learning*, vol. 3, no. 2, pp. 123–224, 2011.

[3] O. of Naval Research. (2017). Broad agency announcement for the office of naval research navy and marine corps fy2018 basic research challenge program, [Online]. Available: `https://webcache.googleusercontent.com/search?q=cache:6C4SPpXd9wcJ:https://www.onr.navy.mil/-/media/Files/Funding-Announcements/BAA/2017/N00014-17-S-BA13.ashx+&cd=3&hl=en&ct=clnk&gl=us#5`.

[4] W. Hackbusch, *Hierarchical Matrices: Algorithms and Analysis*, ser. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2015, ISBN: 9783662473245. [Online]. Available: `https://books.google.com/books?id=LlNECwAAQBAJ`.

[5] W. Hackbusch and S. Börm, "Data-sparse approximation by adaptive h2-matrices," *Computing*, vol. 69, no. 1, pp. 1–35, 2002.

[6] R. Gholami, A. Menshov, and V. I. Okhmatovski, "H-matrix accelerated solution of surface-volume-surface efie for fast electromagnetic analysis on 3-d composite dielectric objects," *IEEE Journal on Multiscale and Multiphysics Computational Techniques*, 2019.

[7] M. Bebendorf, "Approximation of boundary element matrices," *Numerische Mathematik*, vol. 86, no. 4, pp. 565–589, 2000.

[8] 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.

[9] X. Xu and R. J. Adams, "Sparse matrix factorization using overlapped localizing logos modes on a shifted grid," *IEEE Transactions on Antennas and Propagation*, vol. 60, no. 3, pp. 1414–1424, 2011.

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

[11] P.-G. Martinsson and V. Rokhlin, "A fast direct solver for boundary integral equations in two dimensions," *Journal of Computational Physics*, vol. 205, no. 1, pp. 1–23, 2005.

[12] C. A. Balanis, *Antenna theory: analysis and design*. John wiley & sons, 2016.

[13] Y. Xu, R. Adams, and S. Gedney, "A nyström implementation of an augmented electric field integral equation for low frequency electrical analysis," in *2009 IEEE Antennas and Propagation Society International Symposium*, IEEE, 2009, pp. 1–4.

[14] L. F. Canino, J. J. Ottusch, M. A. Stalzer, J. L. Visher, and S. M. Wandzura, "Numerical solution of the helmholtz equation in 2d and 3d using a high-order nyström discretization," *Journal of computational physics*, vol. 146, no. 2, pp. 627–663, 1998.

[15] A. F. Peterson and M. M. Bibby, "An introduction to the locally-corrected nyström method," *synthesis lectures on computational electromagnetics*, vol. 4, no. 1, pp. 1–115, 2009.

[16] S. D. Gedney and J. C. Young, "The locally corrected nyström method for electromagnetics," in *Computational Electromagnetics*, Springer, 2014, pp. 149–198.

[17] X. Xu, "Modular fast direct analysis using non-radiating local-global solution modes," 2008.

[18] R. J. Adams, A. Zhu, and F. X. Canning, "Sparse factorization of the tmz impedance matrix in an overlapped localizing basis," *Progress in Electromagnetics Research*, vol. 61, pp. 291–322, 2006.

[19] H. Cheng, Z. Gimbutas, P.-G. Martinsson, and V. Rokhlin, "On the compression of low rank matrices," *SIAM Journal on Scientific Computing*, vol. 26, no. 4, pp. 1389–1404, 2005.

[20] S. Börm and L. Grasedyck, "Hybrid cross approximation of integral operators," *Numerische Mathematik*, vol. 101, no. 2, pp. 221–249, 2005.

[21] R. J. Adams, Y. Xu, X. Xu, J.-S. Choi, S. D. Gedney, and F. X. Canning, "Modular fast direct electromagnetic analysis using local-global solution modes," *IEEE transactions on Antennas and Propagation*, vol. 56, no. 8, pp. 2427–2441, 2008.

[22] R. Adams and J. Young, "A diagonal factorization for integral equation matrices," in *2016 International Conference on Electromagnetics in Advanced Applications (ICEAA)*, IEEE, 2016, pp. 812–815.

[23] J. Cheng, "Formulation and solution of electromagnetic integral equations using constraint-based helmholtz decompositions," 2012.

[24] R. Adams, F. Canning, and A. Zhu, "Sparse representations of integral equations in a localizing basis," *Microwave and Optical Technology Letters*, vol. 47, no. 3, pp. 236–240, 2005.

[25] A. Tulloch. (2014-09-22). Fast randomized svd, [Online]. Available: `https://research.fb.com/fast-randomized-svd/`.

[26] R. A. Pfeiffer, J. C. Young, R. J. Adams, and S. D. Gedney, "Higher-order simulation of impressed current cathodic protection systems," *Journal of Computational Physics*, 2019.

[27] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," Califonia Institute of Technology, Pasadena, CA, 10.7907/PK8V-V047, 2009.

[28] Z. Jia and Y. Yang, "Modified truncated randomized singular value decomposition (mtrsvd) algorithms for large scale discrete ill-posed problems with general-form regularization," *Inverse Problems*, vol. 34, no. 5, p. 055 013, 2018.

[29] W. C. Gibson, *The method of moments in electromagnetics*. Chapman and Hall/CRC, 2014.

[30] M. Shafieipour, J. Aronsson, I. Jeffrey, and V. I. Okhmatovski, "Exact relationship between the locally corrected nyström scheme and rwg moment method for the mixed-potential integral equation," *IEEE Transactions on Antennas and Propagation*, vol. 63, no. 11, pp. 4932–4943, 2015.

[31] R. Adams, O. Wilkerson, J. Young, I. Chowdhury, and W. Theil, "Localizing sparse direct solvers for circuit simulations," IEEE IMS 2019 Conference Paper.

[32] Y. Xu, X. Xu, and R. J. Adams, "A sparse factorization for fast computation of localizing modes," *IEEE Transactions on Antennas and Propagation*, vol. 58, no. 9, pp. 3044–3049, 2010.

[33] P. Drineas and M. W. Mahoney, "Lectures on randomized numerical linear algebra," *CoRR*, vol. abs/1712.08880, 2017. arXiv: `1712.08880`. [Online]. Available: `http://arxiv.org/abs/1712.08880`.

**Vita**

## Owen Tanner Wilkerson

### Education

Bachelor of Science in Computer Engineering (Summa Cum Laude) from The University of Kentucky, August 2014 - December 2018.

Bachelor of Science in Computer Science (Summa Cum Laude) from The University of Kentucky, August 2014 - December 2018.

### Employment

Algorithm Research Engineer, Gentex Corporation, Starting January 2020.

Graduate Research Assistant and Research Software Engineer, Dr. R. J. Adams at The University of Kentucky, January 2019 - December 2019.

Undergraduate Research Assistant and Research Software Engineer, Dr. R. J. Adams at The University of Kentucky, May 2016 - December 2019.

Systems Software Engineer Internship, 10up, June 2018 - August 2018.

Research and Development Engineer Internship, Massachusetts Institute of Technology (MIT) Lincoln Laboratory, May 2017 - August 2017.

### Honors

NASA Kentucky Graduate Research Fellowship

### Publications

C. Chang, T. Wilkerson, R.J. Adams*, and J.C. Young, "Diagonal Localization-based Direct Solver for Volume Integral Equations and Capacitive Extraction ," 2017 IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting, TU-UB.2A.8, San Diego, CA, July 9 - July 14, 2017. (Abstract)

R. J. Adams, O. T. Wilkerson, J. C. Young, I. Chowdury, and W. Thiel, "Localizing sparse direct solvers for circuit simulations," Proceedings of the 2019 IEEE/MTT-S International Microwave Symposium, pp. 128-131, 2019.

R. J. Adams, O. T. Wilkerson, J. C. Young, and S. D. Gedney, "Binormalized Factorizations for Magnetostatic Integral Equations," 2019 IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting, Atlanta, GA, July 8 - July 12, 2019.