# FAST ALGORITHMS FOR DENSE NUMERICAL LINEAR ALGEBRA AND APPLICATIONS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF INSTITUTE FOR COMPUTATIONAL AND MATHEMATICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Sivaram Ambikasaran

August 2013



This dissertation is online at: http://purl.stanford.edu/tj786mf7514

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

**Eric Darve, Primary Adviser**

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

**Peter Kitanidis**

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

**Lexing Ying**

Approved for the Stanford University Committee on Graduate Studies.

**Patricia J. Gumport, Vice Provost for Graduate Education**

*This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.*

# Preface

Large dense matrices arise in many applications such as gravitation, electrostatics, molecular dynamics simulations, boundary integral equations, imaging etc. Operations involving such large dense matrices is computationally intensive. This dissertation is composed of research and results from papers researched and published under the supervision of Professor Eric F. Darve in the Institute for Computational and Mathematical Engineering, Stanford University. Most of this work is compiled from publications listed below:

- Ambikasaran, S., and Darve, E. F., "An $\mathcal{O}(N \log N)$ fast direct solver for partially hierarchically semi-separable matrices", Journal of Scientific Computing.
- Ambikasaran, S., Saibaba, A. K., Darve, E. F., Kitanidis, P. K., "Fast algorithms for Bayesian inversion", IMA Volume-156, Computational Challenges in the Geosciences.
- Ambikasaran, S., Li, J. Y., Kitanidis, P. K., Darve, E. F., "Large-scale stochastic linear inversion using hierarchical matrices", Computational Geosciences.
- Ambikasaran, S., Aminfar, A., and Darve, E. F., "A sparse matrix approach for constructing fast direct solvers for hierarchical matrices.", in preparation.
- Ambikasaran, S., and Darve, E. F., "Review of fast multipole method & hierarchical matrices.", to be submitted to Mathematics & Mechanics of Complex Systems.
- Ambikasaran, S., Li, J. Y., Darve, E. F., Kitanidis, P. K., "A computationally efficient large-scale Kalman filtering algorithm", to be submitted.

The following packages were released during the course of my doctoral research:

- Black-Box Fast Multipole Method in 2D:
  - Code: `https://github.com/sivaramambikasaran/BBFMM2D`
  - Documentation: `http://sivaramambikasaran.github.io/BBFMM2D/`
- Fast Linear Inversion PACKage:
  - Code: `https://github.com/sivaramambikasaran/FLIPACK`
  - Documentation: `http://sivaramambikasaran.github.io/FLIPACK/`

Other mathematical packages (Fast Kalman filter and Fast direct solvers) from my doctoral research will also be posted at `https://github.com/sivaramambikasaran/` in future.

# Acknowledgements

I would like to thank my advisor Prof. Eric Darve for his constant support, invaluable help and ample freedom to explore this topic & Prof. Peter Kitanidis for giving me an opportunity to apply these algorithms to problems arising out of real situations. I would also like to thank Prof. Lexing Ying for taking time off to be on my reading committee, and Prof George Papanicolaou and Prof. Bala Rajaratnam for being part of my oral defense committee.

I would also like to thank all my friends, well-wishers and acquaintance. My acknowledgements would be incomplete without thanking my parents, Radha Ambikasaran and Ambikasaran Krishnamurti, my fiancée, Krithika, my grandparents, Meenakshi Krishnamoorthy and Krishnamoorthy & Rukmini Krishnamurti and Krishnamurti Ramanathan, for their unwavering support and encouragement.

Dedicated to my grandfather, Krishnamurti Ramanathan

# Contents

**Preface** **v**

**Acknowledgements** **vii**

**viii**

**1 Introduction** **1**

**2 Low-rank matrices** **4**

2.1 Constructing low-rank factorizations . . . . . 5

2.1.1 Algebraic techniques . . . . . 5

2.1.2 Analytic techniques . . . . . 14

2.2 Low-rank algebra . . . . . 21

**3 Hierarchical matrices** **23**

3.1 Hierarchically off-diagonal low-rank matrix . . . . . 25

3.2 HSS matrix . . . . . 27

3.3 $\mathcal{H}$-matrix . . . . . 28

3.4 $\mathcal{H}^2$-matrix . . . . . 28

3.5 FMM-matrix . . . . . 29

**4 Fast linear inversion** **36**

4.1 Introduction . . . . . 36

4.2 Preliminary ideas . . . . . 40

4.2.1 Stochastic linear inversion . . . . . 40

4.2.2 Hierarchical matrices . . . . . 44

4.3 Problem specification . . . . . 47
4.4 Algorithm . . . . . 49
4.5 Numerical benchmark . . . . . 51
4.5.1 Synthetic data set . . . . . 51
4.5.2 Real data set . . . . . 55
4.6 Conclusions . . . . . 60

**5 Fast Kalman filtering** **62**
5.1 Introduction . . . . . 62
5.2 Preliminary ideas . . . . . 63
5.2.1 Linear dynamical system . . . . . 64
5.2.2 Measurement equation . . . . . 64
5.2.3 Algorithm and computational cost . . . . . 65
5.2.4 Computationally efficient Kalman filter . . . . . 65
5.3 Validating the new filter . . . . . 68

**6 Factorization based fast direct solvers** **71**
6.1 Background and motivation . . . . . 71
6.2 Overview of method and relation to previous work . . . . . 73
6.3 Preliminary ideas . . . . . 75
6.3.1 Sherman-Morrison-Woodbury formula . . . . . 76
6.3.2 Fast low-rank factorization . . . . . 76
6.3.3 Hierarchical representations . . . . . 77
6.4 Algorithm . . . . . 80
6.4.1 Factorization phase . . . . . 80
6.4.2 Solving phase . . . . . 84
6.5 Numerical benchmark . . . . . 85
6.5.1 Interpolation using radial basis functions . . . . . 86
6.5.2 Problem specification . . . . . 87
6.5.3 Results and discussion . . . . . 89
6.6 Conclusions . . . . . 99

**7 Extended sparsification based fast direct solvers** **100**
7.1 Introduction . . . . . 100

7.2 Extended sparsification approach . . . . . 102
7.2.1 Fast direct solver for HODLR matrices . . . . . 102
7.2.2 Numerical benchmark . . . . . 113
7.2.3 Fast direct solver for HSS matrices . . . . . 113
7.2.4 Numerical benchmark . . . . . 124
7.3 Conclusions . . . . . 124

**8 Summary and Conclusions** **125**

# List of Tables


3.1 Different hierarchical structures. . . . . 25

3.2 Properties of different hierarchical matrices . . . . . 26

4.1 Comparison of time taken and relative error for the synthetic test case shown in Figure 4.3. The relative error is computed by comparing the reconstructed image (right panel in Figure 4.3) and the true solution (left panel). We chose the approximation order in the FMM such that the FMM error is small compared to the inverse algorithm error. As a result the relative error for the direct and fast algorithms are comparable. A more complete error convergence is shown for the real test case in Figure 4.4 and 4.9. The algorithm was implemented in C++. All the calculations were run on a machine with a single core 2.66 GHz processor and 8 GB RAM. There was no necessity to parallelize the implementation for the present purpose due to the speedup achieved with the proposed algorithm. . . . . 53

4.2 Comparison of the computational time for the real test case of Figure 4.10 . 58

5.1 Comparison of time taken by the exact KF and CEKF . . . . . 69

6.1 Computational complexity; $p$: rank; $r$: number of right-hand sides; $N$: size of matrix . . . . . 85

6.2 Radial basis functions $\phi(r)$ considered for numerical benchmarking. . . . . 89

6.3 Condition number of a $8192 \times 8192$ system for different kernels where the points are distributed randomly on a unit circle. . . . . 90

6.4 Time taken to solve a linear system as a function of the system size holding the rank of the off-diagonal blocks fixed at 30. . . . . 94

7.1 Computational cost for matrix vector product for different hierarchical structures. . . . . 101
7.2 Computational cost for the fast direct solver discussed in this chapter. . . . . 101
7.3 Ordering of unknowns . . . . . 104
7.4 Ordering of equations . . . . . 105
7.5 Ordering of equations . . . . . 116

# List of Figures


2.1 Outer product representation of a low-rank matrix. . . . . 5

3.1 An Euler diagram of the different hierarchical matrices . . . . . 25

3.2 Partitioning of a smooth 1D manifold; Left: Level 0; Middle: Level 1; Right: Level 2. . . . . 26

3.3 HODLR-matrix at different levels for the corresponding tree in Figure 3.2. . 27

3.4 Points distributed uniformly random within a unit square . . . . . 29

3.5 Different levels in the partitioning of a unit square using a binary $\mathcal{H}$-matrix tree. . . . . 30

3.6 $\mathcal{H}$-matrix at levels 3 and 4 corresponding to the tree in Figure 3.5. . . . . 31

3.7 Different levels in the partitioning of a unit square using a quad tree for FMM. 31

3.8 FMM-matrix at levels 2 and 3 corresponding to the tree in Figure 3.7. . . . 32

3.9 Pictorial representation of different sets of clusters at level 2. The interaction of the red cluster $\mathscr{C}$ with the clusters in the interaction list i.e. the green clusters, is computed using analytic techniques. . . . . 33

3.10 Pictorial representation of different sets of clusters at level 3. The interaction of the red cluster $\mathscr{C}$ with the clusters in the interaction list i.e. the green clusters, is computed using analytic techniques. The interaction of the red cluster with the pink clusters have already been computed at the previous level, level 2. . . . . 33

3.11 An unbalanced tree for a nonuniform distribution of charges. . . . . 34

4.1 Hierarchical matrices arising out a 2D problem at different levels in the tree. 45

4.2 Discretization of the domain between the two wells. The line segment $AB$ denotes the well where the $n_s$ sources are placed, while the line segment $CD$ denotes the well where the $n_r$ receivers are placed. The seismic wave is generated by a source and the receiver measures the time taken for it to receive the seismic wave. Each source-receiver pair constitutes a measurement. 48

4.3 Left: known slowness field — synthetic data set; Right: reconstructed slowness field for the synthetic data set using a million grid points. . . . . 52

4.4 Relative error in $QH^T$ versus the number of Chebyshev nodes along one direction for the Gaussian covariance for $m = 40,000$. . . . . 53

4.5 Left: comparison of the time taken by the fast algorithm vs the conventional direct algorithm; Right: comparison of the storage cost. . . . . 54

4.6 Left: time taken by the fast algorithm as a function of the number of unknowns; Right: storage cost by the fast algorithm as a function of the number of unknowns. . . . . 54

4.7 The schematic on the left is the actual crosswell geometry from [30]. . . . . 55

4.8 Relative error for the reconstructed image (compared to the exact answer in the left panel of Fig. 4.10) as a function of $a$, the length of the source array, and $b$, the location of the center of the source array. We used 25 evaluation points (5 along $a$ and $b$). . . . . 58

4.9 Relative error in $QH^T$ versus the number of Chebyshev nodes along one direction for $m = 237 \times 217 = 51,429$. . . . . 58

4.10 Left: true slowness using TOUGH2/ECO2N [100, 101] simulations, 120 hours after injection; Middle: reconstructed slowness using our proposed algorithm for the optimal choice of $a$ and $b$; Right: uncertainty in the estimated solution. All these plots are for a $237 \times 217$ grid. Each unit of slowness corresponds to $10^4$ sec/m. . . . . 59

4.11 Left: time taken by the fast algorithm and the conventional direct algorithm; Right: storage cost. . . . . 59

5.1 Comparison of relative error of the exact and fast Kalman filter . . . . . 70

6.1 Factorization of a three level HODLR/p-HSS matrix . . . . . 85

6.2 Deformation of a one dimensional manifold . . . . . 88

6.3 Decay of singular values of the off-diagonal blocks for different radial basis functions. The singular values are normalized using the largest singular-value. Left: system size $1024 \times 1024$; Right: system size $8192 \times 8192$. . . . . 90
6.4 Keeping the system size fixed at 8192; Left: Relative error; Middle: Time taken to assemble in seconds; Right: Time taken to solve in seconds; versus rank of the off-diagonal blocks . . . . . 93
6.5 Keeping the rank of the off-diagonal block fixed at 30; Left: Relative error; Middle: Time taken to assemble in seconds; Right: Time taken to solve in seconds; versus system size $N$. . . . . 93
6.6 Split up of the time taken by the algorithms at each level for a $1,048,576 \times 1,048,576$ system. Left: $N \log^2 N$ algorithm; Right: $N \log N$ algorithm . . . 94
6.7 Relative error for $1 + (r/a)^2$. Left: versus rank for a system size of 8192 (the green and blue curves overlap); Right: versus system size keeping the off-diagonal rank as 30. . . . . 95
6.8 Relative error for $\sqrt{1 + (r/a)^2}$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30. . . . . 96
6.9 Relative error for $\frac{1}{1+(r/a)^2}$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30. . . . . 96
6.10 Relative error for $\frac{1}{\sqrt{1+(r/a)^2}}$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30. . . . . 97
6.11 Relative error for $\exp(-r/a)$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30. . . . . 98
6.12 Relative error for $\log(1 + r/a)$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30. . . . . 98

7.1 The sparsified matrix at the lowest level . . . . . 108
7.2 The sparsified matrix: After the first set of eliminations. . . . . 109
7.3 The sparsified matrix: After the second set of eliminations. . . . . 110
7.4 The sparsified matrix: After the third set of eliminations. . . . . 111
7.5 The sparsified matrix: After the fourth set of eliminations. . . . . 111
7.6 The sparsified matrix: After the fifth set of eliminations. . . . . 112
7.7 The sparsified matrix: After the sixth set of eliminations. . . . . 112
7.8 The sparsified matrix: After the seventh set of eliminations. . . . . 112

7.9 The sparsified matrix: After the eigth set of eliminations. . . . . 112
7.10 Left: Comparison of relative error versus system size; Right: Comparison of time taken versus system size. . . . . 113
7.11 The sparsified matrix at the lowest level . . . . . 119
7.12 The sparsified matrix: After the first set of eliminations. . . . . 120
7.13 The sparsified matrix: After the second set of eliminations. . . . . 121
7.14 The sparsified matrix: After the third set of eliminations. . . . . 122
7.15 The sparsified matrix: After the fourth set of eliminations. . . . . 122
7.16 The sparsified matrix: After the fifth set of eliminations. . . . . 123
7.17 The sparsified matrix: After the sixth set of eliminations. . . . . 123
7.18 The sparsified matrix: After the seventh set of eliminations. . . . . 123
7.19 The sparsified matrix: After the eigth set of eliminations. . . . . 123
7.20 Left: Comparison of relative error versus system size; Right: Comparison of time taken versus system size. . . . . 124

# Chapter 1

# Introduction

The thesis discusses fast algorithms —algorithms that scale linearly ($\mathcal{O}(N)$) or almost linearly ($\mathcal{O}(N \log^a N)$) with the number of underlying degrees of freedom —for dense numerical linear algebra, especially dense matrix vector products and solvers for dense linear systems arising out of physical applications. The applications considered in the thesis are inverse problems arising in the context of geophysics applications.

Fast computational algorithms, which reduce the computational burden by exploiting the underlying structures like symmetry, self-similarity, etc. play a huge role in solving computational problems, especially given that there is a huge emphasis on big data and large scale problems in recent time.

The most widely known important fast algorithm is the "Fast Fourier Transform" (FFT), which reduces the computational complexity of the Discrete Fourier Transform from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. The value of FFT can never be overestimated given the enormous speed-up gained by the use of FFT for even moderately sized problems. One of the many ways to interpret the FFT is through the language of structured matrices. The matrix arising out of the discrete Fourier transform is a circulant matrix, which has a nice structure associated with it. This structure can be exploited to accelerate the computation of matrix vector products from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. The FFT is exact for problems arising on uniformly spaced grid, but is brittle for problems where the underlying is grid is arbitrary.

On the contrary, the fast algorithms discussed in this thesis are applicable for problem arising from arbitrary grids. These algorithms are approximate and require certain assumptions but are robust and can be made arbitrarily accurate. The *key ingredient* in these algorithms is that, certain interactions —equivalently certain sub-blocks in dense matrices

—can be efficiently represented as a low-rank matrix. Hence, the success of these algorithms rely on efficient low-rank constructions and fast matrix-algebra based on these low-rank sub-blocks.

In this thesis, fast matrix-vector products and fast direct solvers are developed and implemented in the context of inverse problems. Inverse problems are ubiquitous in engineering sciences. For reasonably simple engineering systems, simple physical theories enable us to make predictions, i.e., predict the outcome of certain measurements. However, most of the complex systems arising in engineering sciences are highly non-trivial to even model in the first place. An *inverse problem* is one which consists of using the results of certain measurements to understand the system better and to infer the values of parameters for modeling the system. In this thesis, we consider large-scale inverse modeling, where we have access to a few measurements but are in need for large number of unknowns or parameters that quantify the system. Such inverse problems are typically ill-posed but under certain regularizations these can be made well-posed. We adopt the Bayesian approach, where the regularization for the inverse problems is incorporated using a prior probability distribution function, to solve these inverse problems. Hence, in this approach, we do not obtain a unique solution but rather an ensemble of possible solutions consistent with the prior and measurements, i.e., we obtain

1. A representative depiction of the unknowns.

2. A measure of accuracy in the form of confidence intervals or variance.

3. Many probable solutions, often referred to as conditional realizations.

However, a major hurdle of this approach is that, when the number of unknowns, $m$, to be estimated is large, the method becomes computationally expensive. The bottleneck arises due to the fact that computational storage and matrix operations involving the large dense prior covariance matrix scales as $\mathcal{O}(m^2)$ or $\mathcal{O}(m^3)$, which makes the Bayesian approach computationally time-consuming. However, thanks to the fast algorithms for dense numerical linear algebra, the computational cost can be reduce to almost linear complexity, i.e., $\mathcal{O}(m)$, $\mathcal{O}(m \log m)$ or $\mathcal{O}(m \log^2 m)$.

The remainder of the thesis is organized as follows:

1. Chapter 2 presents a brief introduction to low-rank matrices, discusses fast algebraic & analytical techniques to construct low-rank factorizations and how to make use of these low-rank factorizations to perform fast low-rank matrix algebra.

2. Chapter 3 presents different classes of hierarchical matrices —the class of matrices that are of main interest in this thesis —and discusses how to construct these.

3. Chapter 4 presents an application of a fast $\mathcal{O}(N)$ matrix-vector products for $\mathcal{H}^2$ matrices in the context of large-scale linear inversion.

4. Chapter 5 presents a new fast Kalman filtering algorithm, which relies on fast $\mathcal{O}(N)$ matrix-vector products for $\mathcal{H}^2$ matrices and a novel way to rewrite the Kalman filtering recurrences in a computationally friendly form.

5. Chapter 6 discusses a new factorization based algorithm for solving hierarchically off-diagonal low-rank matrices and hierarchically semi-separable matrices at a computational cost of $\mathcal{O}(N \log^2 N)$ and $\mathcal{O}(N \log N)$ respectively.

6. Chapter 7 discusses a new approach based on extended sparsification for solving different hierarchical matrices. The computational complexity for solving hierarchically off-diagonal low-rank matrices and hierarchically semi-separable matrices based on this algorithm scales as $\mathcal{O}(N \log^2 N)$ and $\mathcal{O}(N)$ respectively.

# Chapter 2

# Low-rank matrices

Low-rank matrices form an important sub-class of dense matrices. Most of the fast algorithms discussed in this thesis relies on low-rank matrices in some form. In this chapter we look in detail at low-rank matrices, construction of these low-rank matrices and matrix algebra using these low-rank matrices.

In a theoretical setting, when we say that a linear operator has rank $p$, it means that the range of the linear operator is a $p$ dimensional space. From a matrix algebra point of view, column rank denotes the number of independent columns of a matrix while row rank denotes the number of independent rows of a matrix.

In an applied setting, the rank of a matrix denotes the *information content* of the matrix. The lower the rank, the lower is the *information content*. For instance, if the rank of a matrix, $A \in \mathbb{R}^{n \times n}$ is 1, the matrix can be written as a product of a column vector times a row vector, i.e., if $a_{11} \neq 0$:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \\ \vdots \\ a_{n1} \end{pmatrix} \times \left(1/a_{11}\right) \times \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \end{pmatrix} \tag{2.1}$$

In general, if we know that a matrix $A \in \mathbb{R}^{m \times n}$ is of rank $p$, then we can write $A$ as $UKV^T$ where $U \in \mathbb{R}^{m \times p}$, $V \in \mathbb{R}^{n \times p}$ and we need to store only $\mathcal{O}(np)$ of its entries. Hence, if we know that a matrix is of low rank, then we can compress and store the matrix and can

perform efficient matrix operations.

When the matrix arises out of a kernel function, then analytic techniques like power series, interpolation, etc., can be used to obtain low-rank factorization of the matrix. The matrix $V^T$, which forms a basis for the rows is termed the *anterpolation operator*. This operator is to be interpreted as aggregating the nodes to a set of super-nodes. The matrix $K$ is termed the *interaction operator*, since it captures the interaction between super-nodes. The matrix $U$ is termed the *interpolation operator*, since it is to be interpreted as disaggregating the information from the super-nodes to the nodes.

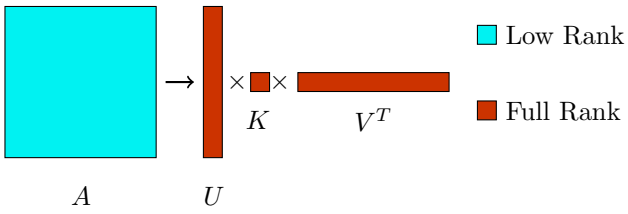



Figure 2.1: Outer product representation of a low-rank matrix.

In the next few sections, we discuss some exact and approximate techniques to obtain a low-rank factorization. These techniques can be broadly divided into algebraic and analytic techniques. We first look at some of the algebraic techniques in section 2.1.1. Low-rank construction using analytic techniques is discussed in section 2.1.2.

## 2.1 Constructing low-rank factorizations

### 2.1.1 Algebraic techniques

In this section, we briefly discuss algebraic techniques, namely, singular value decomposition, pseudo-skeletal approximation, adaptive cross approximation, rank revealing LU factorization, and rank revealing QR factorization, to construct low-rank factorizations.

#### 2.1.1.1 Singular value decomposition

We first look at *singular value decomposition* of a matrix. The singular value decomposition (Proposition 1) of a matrix is one of the most important decompositions of a matrix and is of much practical importance.

**Proposition 1** *Consider a matrix $A \in \mathbb{C}^{m\times n}$. Then there exists a factorization of the form*

$$A = U\Sigma V^* \tag{2.2}$$

*where $U \in \mathbb{C}^{m\times m}$, $V \in \mathbb{C}^{n\times n}$ are unitary matrices and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix with non-negative entries on the diagonal, i.e., $\sigma_k = \Sigma_{kk} \geq 0$. Further, $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0$.*

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \vdots & \vdots & \ddots \end{pmatrix} \tag{2.3}$$

*These $\sigma_k$'s are termed as the singular values of the matrix $A$. Here $V^*$ denotes the conjugate transpose of the matrix $V$. Also, a square matrix $M \in \mathbb{C}^{r\times r}$ is unitary if*

$$MM^* = M^*M = I \tag{2.4}$$

*where $I$ is the identity matrix. The decomposition in Equation (2.2) is termed as "singular value decomposition."*

The singular value decomposition enables us to compute the *optimal* low-rank approximation of a matrix. Theorem 1 reveals how the singular value decomposition gives the "optimal" rank $p$ approximation to a matrix.

**Theorem 1** *Consider $A \in \mathbb{C}^{m\times n}$, and let $\|\cdot\|$ be a unitarily invariant matrix norm (for instance, the 2-norm and the Frobenius/ Hilbert-Schmidt norm). Let $U\Sigma V^*$ be the singular value decomposition of the matrix $A$. Then $\forall p \in \{1, 2, \ldots, \min\{m, n\}\}$, we have*

$$\inf_{M\in\mathbb{R}^{m\times n}} \{\|A - M\| \mid \operatorname{rank}(M) \leq p\} = \|A - A_p\| \tag{2.5}$$

*where,*

$$A_p = U\Sigma_p V^* = \sum_{k=1}^{p} \sigma_k u_k v_k^* \tag{2.6}$$

*and* $\Sigma_k = diag(\sigma_1, \sigma_2, \ldots, \sigma_p, 0, \ldots, 0)$. *Also,* $u_i, v_i, i = 1, \ldots, k$ *are the first* $k$ *columns of the matrices* $U$ *and* $V$ *respectively.*

In other words, the "optimal" approximation of a matrix, of rank at most $p$ is obtained by retaining the first $p$ singular values and vectors of the matrix. In particular, for $A_p$ as defined before, we have

$$\|A - A_p\|_2 = \sigma_{p+1} \qquad \text{and} \qquad \|A - A_p\|_{\text{Fro}}^2 = \sum_{j=p+1}^{\min\{m,n\}} \sigma_j^2 \tag{2.7}$$

In terms of a relative error of approximation, it is useful to talk about $\varepsilon$-rank of a matrix. The $\varepsilon$-rank of a matrix $A$ in the norm $\|\cdot\|$ is defined as

$$p(\varepsilon) := \min \left\{ r \mid \frac{\|A - A_r\|}{\|A\|} \leq \varepsilon \right\} \tag{2.8}$$

where, $A_r = U\Sigma_r V^*$.

One of the major drawbacks of using singular value decomposition in fast algorithms to construct low-rank approximations is that the singular value decomposition is computationally expensive. The computational cost to construct a singular value decomposition of a $m \times n$ matrix is $\mathcal{O}(mn \min(m, n))$, which is not desirable to construct low-rank decomposition.

#### 2.1.1.2 Pseudo-skeletal or $CUR$ approximation

The pseudo-skeletal approximation [48] is another technique to construct approximate low-rank factorizations. Though pseudo-skeletal approximations do not yield the optimal low-rank factorizations, the computational cost to construct low-rank factorizations using the pseudo-skeletal approach is independent of the size of the matrix and depends only on the rank. This makes it attractive to be used for large systems and in the construction of fast algorithms.

A pseudo-skeletal approximation of a matrix $A \in \mathbb{C}^{m \times n}$ is a decomposition of the form $CUR$, where $C \in \mathbb{C}^{m \times p}$ is a sub-matrix of $A$ consisting of certain columns of $A$, $R \in \mathbb{C}^{p \times n}$ is another sub-matrix of $A$ consisting of certain rows of $A$ and $U \in \mathbb{C}^{p \times p}$ is a square matrix. It is to be noted that there are many different $CUR$ decompositions and depending on the matrix $A$, certain $CUR$ decompositions are more optimal than others. Here we look at one such way.

Consider a matrix $A \in \mathbb{C}^{m \times n}$, which has exact rank $p$. Choose $p$ linearly independent rows and columns. Hence, these $r$ rows span the entire $m$ rows and the $p$ columns span the entire $n$ columns. Without loss of generality, we can assume that the first $p$ rows and $p$ columns are linearly independent. (Else pre-multiply and post-multiply by permutation matrices to get the linearly independent rows and columns to the top and to the left respectively).

Let $C$ denote the first $p$ columns and $R$ denote the first $p$ rows and $U$ denote the $p \times p$ sub-matrix at the top-left. We denote: $C = \begin{pmatrix} U \\ C_A \end{pmatrix}$ and $R = \begin{pmatrix} U & R_A \end{pmatrix}$ and $A = \begin{pmatrix} U & R_A \\ C_A & M \end{pmatrix}$. Then:

$$CU^{-1}R = \begin{pmatrix} U \\ C_A \end{pmatrix} U^{-1} \begin{pmatrix} U & R_A \end{pmatrix} = \begin{pmatrix} U \\ C_A \end{pmatrix} \begin{pmatrix} U^{-1}U & U^{-1}R_A \end{pmatrix} \tag{2.9}$$

$$= \begin{pmatrix} U \\ C_A \end{pmatrix} \begin{pmatrix} I & U^{-1}R_A \end{pmatrix} = \begin{pmatrix} U & UU^{-1}R_A \\ C_A & C_AU^{-1}R_A \end{pmatrix} \tag{2.10}$$

$$= \begin{pmatrix} U & R_A \\ C_A & C_AU^{-1}R_A \end{pmatrix} = A \tag{2.11}$$

since $M = C_AU^{-1}R_A$ from the assumption that $A$ is rank $p$.

If the matrix is not of exact rank $p$, but has an $\varepsilon$-rank of $p_\varepsilon$, we can still obtain an $\varepsilon$-rank approximation using theorem 2 from [48].

**Theorem 2** *Assume that the matrix $A^{m \times n}$ has an $\varepsilon$-rank at most $p_\varepsilon$, then there exists a pseudo-skeletal approximation of the form $CUR$, where $U \in \mathbb{R}^{p_\varepsilon \times p_\varepsilon}$ is a matrix that contains appropriate coefficients that are calculated from the sub-matrix of $A$ in the intersection of rows $R$ and columns $C$, such that*

$$\frac{\|A - CUR\|_2}{\|A\|_2} \leq \varepsilon(1 + 2\sqrt{p_\varepsilon}(\sqrt{m} + \sqrt{n})) \tag{2.12}$$

Note that theorem 2 only proves the existence of such a factorization. It doesnot provide us with an algorithm to identify the desired rows and columns and thereby constructing the low-rank approximation. Also, in general if a matrix $A$ is of rank $p$, it is hard to choose the $p$ rows and $p$ columns exactly. One way to circumvent this issue is to choose say $kp$

rows and $kp$ columns. The matrix $U$ now is not invertible. Obtain the pseudo-inverse of $U$ and use singular value decomposition on the pseudo-inverse of $U$ to get the rank $p$ approximation of the pseudo-inverse of $U$. This along with the matrices $C$ and $R$, gives us the $CUR$ decomposition. The cost of $CUR$ decomposition scales as $\mathcal{O}(p^2(m+n))$. However, if the rows and columns are chosen *apriori*, the cost of obtaining the $CUR$ decomposition is $\mathcal{O}(p^3)$.

There are several techniques to choose the "best" set of rows and columns, i.e., the sub-matrices $R$ and $C$. These include greedy algorithms, adaptive techniques, random sampling techniques, interpolatory techniques like using radial basis functions etc. Also, in the above algorithm to construct the matrix, $U$, we ensured that the matrix $CUR$ reproduces exactly certain columns and rows of the matrix $A$. Another popular technique is to compute the matrix $U$ using least squares technique, i.e., given $C$ and $R$, find the matrix $U$ such that $\|A - CUR\|$ is minimum. However, in this case, the computational cost to compute $U$ scales as $\mathcal{O}(p(m^2 + n^2))$.

#### 2.1.1.3 Adaptive cross approximation

Adaptive cross approximation (ACA) [102] is another technique to construct low-rank approximations of dense matrices. ACA can be interpreted as a special case of the pseudo-skeletal algorithm based on an appropriate pivoting strategy. The idea behind the cross approximation is based on the result described in [10], which states that supposing a matrix $A$ is well approximated by a low-rank matrix, then we can approximate $A$ with almost the same approximation quality by a clever choice of $p$ columns $C \in \mathbb{R}^{m\times p}$ and $p$ rows $R$ of the matrix $A$. The algorithm is described in detail in algorithm 1. Though the algorithm is more efficient than the singular value decomposition, it is still computationally expensive, since it chooses the "best" set of rows and columns and hence it costs $\mathcal{O}(pmn)$.

#### 2.1.1.4 Partially pivoted adaptive cross approximation

Several heuristic strategies have been proposed to reduce the computational complexity of the adaptive cross approximation algorithm described in 2.1.1.3. Of these, the most popular one is the *partially pivoted adaptive cross approximation* algorithm 2. The main advantage of the partially pivoted ACA is that it only needs the individually entries $A_{ij}$ of the matrix as opposed to the entire row (or) column. The computational complexity of partially pivoted ACA is $\mathcal{O}(p^2(m+n))$ and is very easy to implement. A practical version of

**Algorithm 1** Adaptive cross approximation technique

Initialize the approximation and the residual matrix.

$$U = 0, K = 0, V = 0 \text{ and } R = A$$

**while** $\|R\|_F \leq \epsilon \|A\|_F$ **do**

1. Obtain the row and column of the pivot element. The pivot element is chosen as the maximum of the absolute value of the entries of the matrix.

$$\{i_{k+1}, j_{k+1}\} = \text{argmax}_{i,j} |R_k(i, j)|$$

2. Get the dominant vectors spanning the row space and column space, i.e.,

$$U(:, k{+}1) = \frac{R_k(:, j_{k+1})}{R_k(i_{k+1}, j_{k+1})}; \; V(k{+}1, :) = \frac{R_k(i_{k+1}, :)}{R_k(i_{k+1}, j_{k+1})} \text{ and } K(k{+}1, k{+}1) = R_k(i_{k+1}, j_{k+1})$$

3. New residual matrix.

$$R = R - U(:, k+1)K(k+1, k+1)V(:, k+1)$$

**end while**

the algorithm, which includes a termination criterion based on a heuristic approximation to the relative approximation in the Frobenius norm, can be found in [15, 103]. The algorithm is described in algorithm 2. The proofs of convergence of this algorithm can be found in [11], and relies on approximation bounds of Lagrange interpolation and a geometric assumptions on the distribution of points, which may not be very practical. For instance, [15] lists some contrived counterexamples that show that this algorithm can produce bad pivots. To fix this issue, several other variants have been proposed such as improved ACA (ACA+) and hybrid cross approximation (HCA).

#### 2.1.1.5 Rank revealing QR factorization

The conventional $QR$ factorization can also be used to construct approximate low-rank approximations. Let $A \in \mathbb{R}^{m \times n}$ be a rank $p$ matrix. The QR factorization with appropriate pivoting, in exact arithmetic, guarantees to produce a low-rank factorization at a

**Algorithm 2** Partially pivoted adaptive cross approximation technique

**while** $k = 0, 1, 2, \ldots$ **do**

1. Generation of the row
$$a = A^T e_{i_{k+1}}$$
2. Row of the residuum and the pivot column
$$(R_k)^T e_{i_{k+1}} = a - \sum_{l=1}^{k} (u_k)_{i_{k+1}} v_k \tag{2.13}$$
$$j_{k+1} = \arg\max_j |(R_k)_{i_{k+1} j}|$$
3. Normalizing constant
$$\gamma_{k+1} = \left((R_k)_{i_{k+1} j_{k+1}}\right)^{-1}$$
4. Generation of the column
$$a = A e_{j_{k+1}}$$
5. Column of the residual and the pivot row
$$R_k e_{j_{k+1}} = a - \sum_{l=1}^{k} (v_k)_{j_{k+1}} u_k \tag{2.14}$$
$$i_{k+2} = \arg\max_i |(R_k)_{i j_{k+1}}|$$
6. New approximation
$$S_{k+1} = S_k + u_{k+1} v_{k+1}^T$$

**end while**

computational cost of $\mathcal{O}(pmn)$, i.e., a reduced QR factorization gives us

$$P_1 A P_2 = \begin{bmatrix} | & | & | & \cdots & | \\ \vec{q}_1 & \vec{q}_2 & \vec{q}_3 & \cdots & \vec{q}_p \\ | & | & | & \cdots & | \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \cdots & r_{1,p-1} & r_{1,p} & r_{1,p+1} & \cdots & r_{1,n} \\ 0 & r_{2,2} & r_{2,3} & \cdots & r_{2,p-1} & r_{2,p} & r_{2,p+1} & \cdots & r_{2,n} \\ 0 & 0 & \ddots & \cdots & r_{3,p-1} & r_{3,p} & r_{3,p+1} & \cdots & r_{3,n} \\ 0 & 0 & 0 & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & r_{p,p} & r_{p,p+1} & \cdots & r_{p,n} \end{bmatrix}$$

where $\vec{q}_j \in \mathbb{R}^{m\times 1}$ are orthonormal vectors. Hence, this gives us a low-rank decomposition of the matrix $A$ as

$$A = P_1 \begin{bmatrix} | & | & | & \cdots & | \\ \vec{q}_1 & \vec{q}_2 & \vec{q}_3 & \cdots & \vec{q}_p \\ | & | & | & \cdots & | \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \cdots & r_{1,p-1} & r_{1,p} & r_{1,p+1} & \cdots & r_{1,n} \\ 0 & r_{2,2} & r_{2,3} & \cdots & r_{2,p-1} & r_{2,p} & r_{2,p+1} & \cdots & r_{2,n} \\ 0 & 0 & \ddots & \cdots & r_{3,p-1} & r_{3,p} & r_{3,p+1} & \cdots & r_{3,n} \\ 0 & 0 & 0 & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & r_{p,p} & r_{p,p+1} & \cdots & r_{p,n} \end{bmatrix} P_2$$

Typically though, only column pivoting is done. For instance, [46] discusses a column pivoting strategy by choosing the maximum diagonal entry in the remainder matrix as one proceeds with the $QR$ factorization. Once we have the permutation matrix, $P \in \mathbb{R}^{n\times n}$, to pivot, the $QR$ factorization of $AP$ is obtained as

$$AP = QR,$$

where $Q \in \mathbb{R}^{m\times n}$ and $R \in \mathbb{R}^{n\times n}$, with $Q^T Q = I_n$ and the upper triangular matrix $R$ is partitioned as

$$R = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix},$$

where $R_{11} \in \mathbb{R}^{p\times p}$ and hopefully $R_{22}$ is small in norm. If $\|R_{22}\|_2 = \mathcal{O}(\epsilon_{\text{machine}})$, we then have that $\sigma_{p+1} = \mathcal{O}(\epsilon_{\text{machine}})$ [66] and thereby we can conclude that the matrix $A$ has a numerical rank of *at most* $p$.

The $QR$ factorization that yields a suitably small $R_{22}$ is referred to as the rank-revealing

$QR$ (RRQR) factorization [19]. While the RRQR is a great way to detect numerical rank of a matrix, the column pivoting strategy discussed in [46] can also fail miserably. Look at example 3.1 in [69] for a counterexample. There are quite a few different pivoting strategies for finding the RRQR factorization of a matrix [13, 19, 40, 46, 70, 86, 91]. However, all of these either fail for the example 3.1 in [69] or do not provide provide a rigorous error bound. However, there are other efficient pivoting strategies to obtain rank-revealing $QR$ factorizations. For instance, the pivoting strategy discussed in [66, 98] based on local maximum volume —the product of the singular values —of a matrix is sufficient for developing an efficient pivoting strategy for obtaining a rank-revealing $QR$ factorization. [20, 54] also discuss efficient pivoting strategy to obtain $RRQR$ factorizations.

#### 2.1.1.6 Rank revealing LU factorization

As with the QR factorization, the conventional LU factorization can also be used to construct fast low-rank approximations. Let $A \in \mathbb{R}^{m\times n}$ be a rank $p$ matrix. The LU factorization with appropriate pivoting, in exact arithmetic, guarantees to produce a low-rank factorization at a computational cost of $\mathcal{O}(pmn)$, i.e., we have

$$P_1 A P_2 = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I_{(m-p)\times(m-p)} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & 0 \end{bmatrix} \tag{2.15}$$

where $P_1$ and $P_2$ are permutation matrices. Hence, this gives us

$$P_1 A P_2 = \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \end{bmatrix}$$

Hence, the low-rank factorization of the matrix $A$ is

$$A = P_1 \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \end{bmatrix} P_2$$

[18] was the first to discuss the existence of a pivoting strategy for RRLU factorizations for certain class of matrices. [97] discusses the existence of a pivoting strategy based on local maximal volume that efficiently extracts full rank and rank deficient portions of the matrix. [97] also claims that the RRLU algorithm is typically faster than the corresponding RRQR algorithms. We refer the readers to [67, 68, 90] for more details regarding RRLU algorithm.

### 2.1.2 Analytic techniques

#### 2.1.2.1 Taylor Series approximation

In this section, we look at an analytic approach to construct low-rank factorization for matrices arising out of analytic kernels. A kernel is termed *asymptotically smooth*, if there exist constants $c_1^{as}, c_2^{as}$ and a real number $g \geq 0$ such that for all multi-indices $\alpha \in \mathbf{N}_0^d$ it holds that

$$\left|\partial_y^\alpha K(x,y)\right| \leq c_1^{as} p!(c_2^{as})^p (|x-y|)^{-g-p}, \quad p = |\alpha| \tag{2.16}$$

Let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ be two pairwise distinct points in $\mathbb{R}^d$ and $D_X, D_Y$ be the convex hulls of $X$ and $Y$ respectively. If we have the additional condition that

$$\text{diam} D_Y \leq \eta \text{dist}(D_X, D_Y), \qquad 0 < \eta < (c_2^{as} d)^{-1} \tag{2.17}$$

then, by using a Taylor expansion at the point $y_0 \in D_\sigma$, where $D_\sigma$ is the convex hull of $X_\sigma$, we get that

$$\kappa(x,y) = \sum_{l=0}^{k-1} \frac{(y-y_0)^l}{l!} \partial_y^l \kappa(x,y_0) + R_k(x,y), \tag{2.18}$$

where $R_k(x,y)$ is the residual in the Taylor series expansion. The residual $R_k(x,y)$ can be written as

$$R_k(x,y) = \frac{(y-y_0)^k}{k!} \partial_y^k \kappa(x,\tilde{y})$$

for some point $\tilde{y} \in D_\sigma$. This then gives us

$$|R_k(x,y)| \leq \frac{d^k}{k!} \left(\frac{|y-y_0|}{|x-\tilde{y}|}\right)^k c_1 k! c_2^k |x-\tilde{y}|^{-g}$$

The rank of the resulting matrix is $\mathcal{O}(k^d)$ [10]. However, there are two problems with this approach. The first is that computing the Taylor series (or for that matter, other analytical expressions such as multipole expansions) in 3D of kernels can be a tedious task. Moreover, every time one needs to use a new kernel, it might be necessary to re-derive the Taylor series for that particular kernel. Therefore, in general we would prefer an approach that is kernel independent.

#### 2.1.2.2 Multipole Expansions

The original version of the FMM is based on certain analytical expansions of the kernel $K(r)$, for example with spherical harmonics, powers of $r$, Bessel functions, etc. As an example, we consider the logarithmic kernel given by $K(x_j, x_k) = -\log(\|x_j - x_k\|)$. Also, we use the language of electrostatics since the original FMM was developed for electrostatic interactions. If a charge of strength $q$ is located at $(x_0, y_0) \in \mathbb{R}^2$, then the potential at $(x, y) \in \mathbb{R}^2$ is given by

$$\phi_{q,z_0}(z) = -q \log(\|z - z_0\|) = \mathrm{Re}\,(-q \log(z - z_0)) \tag{2.19}$$

where $z = x + iy$, $z_0 = x_0 + iy_0$ and $z, z_0 \in \mathbb{C}$. Let $\psi_{q,z_0}(z) = q \log(z - z_0)$, i.e., $\phi_{q,z_0}(z) = -\mathrm{Re}\,(\psi_{q,z_0}(z))$. Now note that for any $z$ such that $|z| > |z_0|$, we have

$$\psi_{q,z_0}(z) = q \log(z - z_0) = q \left( \log(z) - \sum_{k=1}^{\infty} \frac{1}{k} \left( \frac{z_0}{z} \right)^k \right) \tag{2.20}$$

This analytic expansion above is the basis for computing the multipole expansion [51, 104] when we have a cluster of charges.

**Lemma 3** *Consider a set of $n$ charges of strength $q_k$ located at $z_k$, where $k \in \{1, 2, \ldots, n\}$. Let $R = \max_k |z_k|$. For any $z$ such that $|z| > R$, the potential $\phi(z)$ due to the $n$ charges is given by*

$$\phi(z) = q \log(z) - \sum_{k=1}^{\infty} \frac{a_k}{k z^k} \tag{2.21}$$

*where $q = \sum_{j=1}^{n} q_j$ and $a_k = \sum_{j=1}^{n} q_j z_j^k$.*

*More importantly, for any $p \in \mathbb{Z}^+$, if we set*

$$\phi_p(z) = q \log(z) - \sum_{k=1}^{p} \frac{a_k}{k z^k} \tag{2.22}$$

*we have the following error bound, given by Equation* (2.23)*.*

$$|\phi(z) - \phi_p(z)| \leq \frac{\sum_{j=1}^{n} |q_j|}{(p+1)\left(1 - \left|\frac{r}{z}\right|\right)} \left| \frac{r}{z} \right|^{p+1} \tag{2.23}$$

*Note that* $|\frac{r}{z}| < 1$.

In the above, if we have $|z| > 2|r|$, we then get that

$$|\phi(z) - \phi_p(z)| \leq \frac{\sum_{j=1}^{n} |q_j|}{(p+1)2^p} \tag{2.24}$$

Using lemma 3 and arguments of the type found in section 2.1.2.1, we can derive low-rank approximations of matrices.

#### 2.1.2.3 Interpolation

Many problems in computational physics involve evaluation of pairwise interactions of large number of particles. Most of these $N$ boyd problems are of the form

$$f(x_j) = \sum_{k=1}^{m} K(x_j, y_k)\sigma_k \tag{2.25}$$

where $j \in \{1, 2, \ldots, m\}$, $x_j$ are the target points, $y_k$ are the source points and $\sigma_k$ are the sources and $K(x, y)$ is a kernel. A direct computation of 2.25 has a computational complexity of $\mathcal{O}(m^2)$, which is prohibitively expensive for large $m$. However, the computation can be accelerated if the kernel happens to be degenerate, i.e., can be well-represented by a low-rank approximation.

$$K(x, y) \approx \sum_{l=1}^{p} u_l(x)v_l(y) \tag{2.26}$$

where $p$ is a fixed number independent of $m$ and $p \ll m$. Once we have such a low-rank representation, a fast summation technique is available. We have

$$f(x_j) \approx \sum_{k=1}^{m}\sum_{l=1}^{p} u_l(x)v_l(y_k)\sigma_k = \sum_{l=1}^{p} u_l(x)\left(\sum_{k=1}^{m} v_l(y_k)\sigma_k\right) \tag{2.27}$$

See Algorithm 3.

Hence, the algorithm above scales as $\mathcal{O}(pm)$ as opposed to $\mathcal{O}(m^2)$ scaling of the direct computation, since $p \ll m$.

Hence, the goal is to construct a low-rank representation which can be constructed by

**Algorithm 3** Fast summation when the kernel is well approximated by a low-rank representation.

1. Transform the sources using the basis functions $v_l(y)$.

$$W_l = \sum_{k=1}^{m} v_l(y_k)\sigma_k, \text{ where } l \in \{1, 2, \ldots, p\}.$$

The computational cost of this step is $\mathcal{O}(pm)$.

2. Compute $f(x)$ at target location using the basis function $u_l(x)$.

$$f(x_j) = \sum_{l=1}^{p} u_l(x_j)W_l, \text{ where } j \in \{1, 2, \ldots, m\}.$$

The computational cost of this step is $\mathcal{O}(pm)$.

some interpolation scheme as shown below.

Consider interpolating a function $g(x)$ on an interval $[a, b]$. For pedagogical reasons, let us fix the interval to be $[-1, 1]$. An $n$-point interpolant approximating the function $g(x)$ is given by

$$g_{n-1}(x) = \sum_{k=1}^{n} g(x_k)w_k(x) \tag{2.28}$$

where $x_k$'s are the interpolation nodes and $w_k(x)$'s are the interpolating functions $k \in \{1, 2, \ldots, n\}$. The above extends to functions (kernels), $K(x, y)$, that are functions of two variables.

First by treating $K(x, y)$ as a function of $x$, we get that

$$K(x, y) \approx \sum_{j=1}^{p} K(x_j, y)w_j(x) \tag{2.29}$$

Now treating $K(x_j, y)$ as a function of $y$, we get that

$$K(x, y) \approx \sum_{j=1}^{p}\sum_{k=1}^{p} K(x_j, y_k)w_j(x)w_k(y) \tag{2.30}$$

This gives us the desired low-rank representation for the kernel $K(x, y)$, since we have

$$K(x, y) = \sum_{j=1}^{p} u_j(x)v_j(y) \tag{2.31}$$

where $u_j(x) = w_j(x)$ and $v_j(y) = \sum_{k=1}^{p} K(x_j, y_k) w_k(y)$. Note that the above low-rank construction works with any interpolation technique.

**Chebyshev interpolation**:

One of the most popular interpolation technique is to interpolate using Chebyshev polynomials, with the Chebyshev polynomials serving as interpolation basis and the Chebyshev nodes serving as interpolation nodes. Let us briefly review some of the properties of Chebyshev polynomials, before proceeding further.

The first kind Chebyshev polynomial of degree $p$, denoted by $T_p(x)$, is defined for $x \in [-1, 1]$ by the recurrence:

$$T_p(x) = \begin{cases} 1 & \text{if } p = 0 \\ x & \text{if } p = 1 \\ 2xT_{p-1}(x) - T_{p-2}(x) & \text{if } p \in \mathbb{Z}^+ \backslash \{1\} \end{cases} \tag{2.32}$$

$T_p(x)$ has $p$ distinct roots in the interval $[0, 1]$ located at $\bar{x}_k = \cos\left(\frac{2k-1}{2p}\right)$, where $k \in \{1, 2, \ldots, p\}$. These $p$ roots are termed as Chebyshev nodes.

There are many advantages to Chebyshev interpolation. The major advantage of Chebyshev interpolation is that it is a stable interpolation scheme, i.e., it doesn't suffer from Runge's phenomenon and the convergence is uniform.

Hence, the Chebyshev interpolation of a function $g(x)$ is an interpolant of the form

$$g_{p-1}(x) = \sum_{k=0}^{p-1} c_k T_k(x) \tag{2.33}$$

where

$$c_k = \begin{cases} \frac{2}{p} \sum_{j=1}^{p} g(\bar{x}_j) T_k(x_j) & \text{if } k > 0 \\ \frac{1}{p} \sum_{j=1}^{p} g(\bar{x}_j) & \text{if } k = 0 \end{cases} \tag{2.34}$$

where $\bar{x}_j$ are the Chebyshev nodes, i.e., the $p$ roots of $T_p(x)$. Rearranging the terms, we get

that

$$g_{p-1}(x) = \sum_{j=1}^{p} g\left(\bar{x}_j\right) S_p\left(\bar{x}_j, x\right) \tag{2.35}$$

where

$$S_p(x,y) = \frac{1}{p} + \frac{2}{p}\sum_{k=1}^{p-1} T_k(x)T_k(y) \tag{2.36}$$

If $g(x) \in C^r([-1,1])$, then error in the approximation is given by

$$|g(x) - g_p(x)| = \mathcal{O}(p^{-r}) \tag{2.37}$$

Further, if the function $g(x)$ can be extended to a function $g(z)$, which is analytic inside a simple closed curve $\Gamma$ that encloses the points $x$ and all the roots of the Chebyshev polynomial $T_{p+1}(x)$, then the interpolant $g_p(x)$ can be written as

$$g_p(x) = \frac{1}{2\pi i}\oint_{\Gamma} \frac{(T_{p+1}(z) - T_{p+1}(x))g(z)}{T_{p+1}(z)(z-x)}dz \tag{2.38}$$

This gives us an expression for the error:

$$g(x) - g_p(x) = \frac{1}{2\pi i}\oint_{\Gamma} \frac{T_{p+1}(x)g(z)}{T_{p+1}(z)(z-x)}dz \tag{2.39}$$

If $g(z)$ is analytic inside an ellipse $E_r$, given by the locus of points $\frac{1}{2}\left(r\exp(i\theta) + \frac{1}{r}\exp(-i\theta)\right)$, for some $r > 1$ and $\theta \in [0, 2\pi)$ and if $M = \sup_{E_r} |g(z)| < \infty$, then for every $x \in [-1,1]$, we have an exponential convergence since

$$|g(x) - g_p(x)| \leq \frac{\left(r + r^{-1}\right)M}{(r^{p+1} + r^{-(p+1)})(r + r^{-1} - 2)} \tag{2.40}$$

Note that $S_p\left(\bar{x}_j, x\right)$ is the interpolating function and $\{\bar{x}_j\}_{j=1}^{p}$ are the interpolation nodes. As discussed earlier, this enables us to get the low-rank representation of the kernel $K(x,y)$

using Chebyshev polynomials as

$$K(x,y) \approx \sum_{j=1}^{p}\sum_{k=1}^{p} K(\bar{x}_j,\bar{y}_k)S_p(\bar{x}_j,x)S_p(\bar{y}_k,y) \tag{2.41}$$

---

**Algorithm 4** Fast summation using Chebyshev interpolation to construct low-rank representation.

---

1. Compute the weights at the Chebyshev nodes $\bar{y}_k$ by anterpolation.

$$W_k = \sum_{j=1}^{m} S_p(\bar{y}_k,y_j)\sigma_j \text{ where } k \in \{1,2,\ldots,p\}.$$

The computational cost of this step is $\mathcal{O}(pm)$.

2. Compute $f(x)$ at the Chebyshev nodes $\bar{x}_l$.

$$f(\bar{x}_l) = \sum_{k=1}^{p} W_k K(\bar{x}_l,\bar{y}_k) \text{ where } l \in \{1,2,\ldots,p\}.$$

The computational cost of this step is $\mathcal{O}(p^2)$.

3. Compute $f(x)$ at the target location $x_j$ by interpolation.

$$f(x_j) = \sum_{l=1}^{p} f(\bar{x}_l)S_p(\bar{x}_l,x_j) \text{ where } j \in \{1,2,\ldots,m\}.$$

The computational cost of this step is $\mathcal{O}(pm)$.

---

Algorithm 4 describes a fast summation algorithm using Chebyshev interpolation. Since $p \ll m$, the computational cost of the algorithm is $\mathcal{O}(pm)$.

The extension to $d$-dimensions proceeds as follows. For instance, consider the kernel $K(\mathbf{x},\mathbf{y})$, where $\mathbf{x} = (x_1,x_2,\ldots,x_d)$ and $\mathbf{y} = (y_1,y_2,\ldots,y_d)$. A rank $p^d$ low-rank approximation to the kernel $K(\mathbf{x},\mathbf{y})$ is given by

$$K(\mathbf{x},\mathbf{y}) \approx \sum_{j=1}\sum_{k=1} K(\bar{\mathbf{x}}_j,\bar{\mathbf{y}}_k)R_p(\bar{\mathbf{x}}_j,\mathbf{x})R_p(\bar{\mathbf{y}}_k,\mathbf{y}) \tag{2.42}$$

where

$$R_p(\mathbf{x},\mathbf{y}) = S_p(x_1,y_1)S_p(x_2,y_2)\ldots S_p(x_d,y_d) \tag{2.43}$$

and $\bar{\mathbf{x}}_j = (\mathbf{x}_{j_1},\mathbf{x}_{j_2},\ldots,\mathbf{x}_{j_d})$ and $\bar{\mathbf{y}}_k = (\mathbf{y}_{k_1},\mathbf{y}_{k_2},\ldots,\mathbf{y}_{k_d})$ are the Chebyshev nodes in $d$-dimensions.

## 2.2 Low-rank algebra

We now look at operations involving low-rank matrices. Throughout this section, we assume that we have the rank $p$ decomposition of the matrix $A \in \mathbb{R}^{m\times n}$ as $UV^T$, where $U \in \mathbb{R}^{m\times p}$ and $V \in \mathbb{R}^{n\times p}$. Algorithm 5 presents efficient matrix-vector product, when the matrix is in low-rank form. Algorithm 6 discusses compressing a rank $p$ matrix into a rank $r$ matrix, where $r < p$. Algorithm 7 and 8 discusses efficient ways to add and multiply low-rank matrices.

**Algorithm 5** Matrix vector product $Ax$ when the matrix is given in low-rank form $A = UV^T$.

1. Compute $w = V^T x$. The computational cost of this step is $\mathcal{O}(pn)$

2. Compute $y = Uw$. The computational cost of this step is $\mathcal{O}(pm)$

The overall cost is $\mathcal{O}(p(m+n))$ as opposed to a direct matrix-vector product, which would scale as $\mathcal{O}(mn)$. The reduction in computational cost is significant when $p \ll \min(m, n)$.

**Algorithm 6** Compressing a rank $p$ matrix, $A = UV^T$, into a rank $r$ matrix, where $r < p$.

1. Compute the QR decomposition of $U$ and $V$, i.e., $U = Q_U R_U$ and $V = Q_V R_V$. Computational cost is $\mathcal{O}(p^2(m+n))$.

2. Compute the singular value decomposition of $R_U R_V^T$, i.e., $R_U R_V^T = \hat{U}\hat{\Sigma}\hat{V}$. Computational cost is $\mathcal{O}(p^3)$.

3. Set $\tilde{U} = Q_U\hat{U}(:, 1:r)$, $\tilde{V} = Q_V\hat{V}(:, 1:r)$ and $\tilde{\Sigma} = \hat{\Sigma}(1:r, 1:r)$ (MATLAB style notation). Computational cost is $\mathcal{O}(rp(m+n))$.

Now $\tilde{A} = \tilde{U}\tilde{\Sigma}\tilde{V}$ is the desired low-rank approximation of $UV^T$.

**Algorithm 7** Adding two low-rank matrices and obtaining a rank $p'$ approximation.

Given $A = U_A V_A^T$, $B = U_B V_B^T$, where $U_A \in \mathbb{R}^{m\times p}$, $V_A \in \mathbb{R}^{n\times p}$, $U_B \in \mathbb{R}^{m\times r}$ and $V_B \in \mathbb{R}^{n\times r}$.

$A + B = U_A V_A^T + U_B V_B^T = [U_A; U_B]\,[V_A; V_B]^T$ in MATLAB notation.
Now use algorithm 6 to compress into the desired rank $p'$ matrix.

**Algorithm 8** Multiplying two low-rank matrices.

Given $A = U_A V_A^T$, $B = U_B V_B^T$, where $U_A \in \mathbb{R}^{m \times p}$, $V_A \in \mathbb{R}^{l \times p}$, $U_B \in \mathbb{R}^{l \times r}$ and $V_B \in \mathbb{R}^{n \times r}$.

Compute $\Sigma_{AB} = V_A^T U_B$. The computational cost of this step is $\mathcal{O}(prl)$.

Now $AB = U_A \Sigma_{AB} V_B^T$ is the desired product in low-rank form.

# Chapter 3

# Hierarchical matrices

Hierarchical matrices are data-sparse approximations of a class of dense matrices. The data-sparse representation relies on the fact that certain dense matrices, can be recursively sub-divided based on a tree structure and certain sub-blocks of these matrices arising at different levels in the tree can be well-represented using a low-rank matrix. For these matrices dense linear algebra like matrix-vector products, matrix factorizations, solving linear equations, etc., can be performed in almost linear complexity. There are different possible hierarchical matrices depending on the tree structure and algorithms to obtain low-rank different matrix. Before discussing the different hierarchical matrices, we briefly discuss how to interpret a matrix using a tree structure. We will only be dealing with balanced hierarchical matrix trees in this chapter.

Any entry of a matrix in the current chapter, should be interpreted as an *interaction* between two points in $\mathbb{R}^d$, where $d$ is typically $1, 2$, or $3$, as discussed below. Consider the index set $N = \{1, 2, 3, \ldots, n\}$, corresponding to $n$ points, $\{\vec{r}_1, \vec{r}_2, \vec{r}_3, \ldots, \vec{r}_n\}$, in the set $X \subset \mathbb{R}^d$, where typically $d \in \{1, 2, 3\}$. A cluster $\mathscr{C}_I \subset X$ is a collection of points close to each in a topological sense. Typically, the topology is induced by the usual Euclidean norm. Let $I$ be the index corresponding to the cluster $\mathscr{C}_I$. Let $A \in \mathbb{R}^{n \times n}$, where $A(i, j)$ denote the interaction between points $\vec{r}_i$ and $\vec{r}_j$, i.e., $A(i, j) : (i, j) \in N \times N \mapsto \mathbb{R}$. Let $\#\mathscr{C}_I$ denote cardinality of $\mathscr{C}_I$.

The matrix $A(I, J) \in \mathbb{R}^{(\#\mathscr{C}_I) \times (\#\mathscr{C}_J)}$ is a sub matrix of $A$ and contains the interaction between points in the cluster $\mathscr{C}_I \subset X$ with the points in the cluster $\mathscr{C}_J \subset X$. Note that $A(X, X) = A$. Let $D_I, D_J$ denote the diameter of the clusters $\mathscr{C}_I$ and $\mathscr{C}_J$ respectively and $D_{IJ}$ denote the distance between the clusters $\mathscr{C}_I$ and $\mathscr{C}_J$. Now let us look a hierarchical

division of points in $X$ based on a tree structure, typically a binary tree, quad tree or oct tree. Now lets define the following terms, which we shall use over the course of the remainder of the chapter. We will only consider balanced trees in the current context.

1. Each node in the tree is a cluster of points, i.e., to each node we associate a cluster $\mathscr{C}_I$.
2. A cluster $\mathscr{C}_I$ is a ***child*** of cluster $\mathscr{C}_J$ if $I \subset J$. The cluster $\mathscr{C}_J$ is termed the ***parent*** of cluster $\mathscr{C}_I$.
3. A cluster $\mathscr{C}_I$ is a ***sibling*** of cluster $\mathscr{C}_J$, if $\mathscr{C}_I$ and $\mathscr{C}_J$ share the same parent. The set of all siblings of a cluster $\mathscr{C}_I$ is denoted by $\mathcal{S}(\mathscr{C}_I)$.
4. A node in the tree is a ***leaf*** if the cardinality of the associated cluster is less than $n_{\min}$.
5. A cluster $\mathscr{C}_J$ is said to be in the ***interaction list*** of $\mathscr{C}_I$ if the matrix capturing the interaction between cluster $\mathscr{C}_I$ and $\mathscr{C}_J$ is a low-rank matrix. The set of all clusters in the interaction list of $\mathscr{C}_I$ is denoted by $\mathcal{I}(\mathscr{C}_I)$.
6. Two disjoint clusters are termed ***well-separated***, if there is a distance of separation either geometrically (or) based on a graph between the two clusters. The set of all well-separated clusters of a cluster $\mathscr{C}_I$ is denoted by $\mathcal{W}(\mathscr{C}_I)$.
7. Two disjoint clusters are termed ***neighbors***, if they are not well-separated. The set of all ***neighbors*** of a cluster $\mathscr{C}_I$ is denoted by $\mathcal{N}(\mathscr{C}_I)$.
8. Two disjoint clusters are termed ***cousins***, if the parents of the two clusters are neighbors. The set of all ***cousins*** of a cluster $\mathscr{C}_I$ is denoted by $\mathcal{C}(\mathscr{C}_I)$.

Different hierarchical structures arise based on the following criteria:

1. **Tree structure**, i.e., binary, quad or oct tree.
2. **Admissibility**, i.e., whether the neighbors are in the interaction list.
3. **Nested low-rank structure**, i.e., if the basis for the interpolation and anterpolation operator of the parent can be constructed from the basis for the interpolation and anterpolation operator of its children's.
4. **Construction of low-rank**, i.e., using analytic or algebraic techniques.

Figure 3.1 presents an Euler diagram of the different hierarchical structures discussed in this chapter.

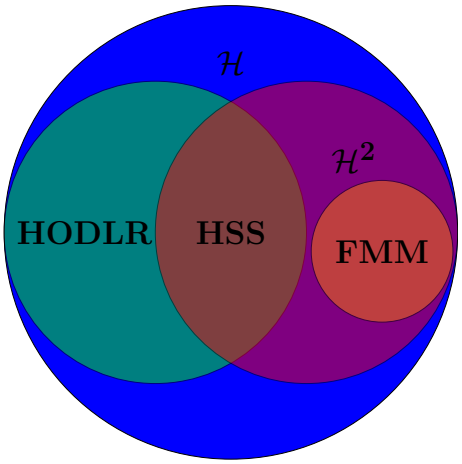



Figure 3.1: An Euler diagram of the different hierarchical matrices

Table 3.1: Different hierarchical structures.

| Admissibility \ Nested basis? | Not-nested | Nested |
|---|---|---|
| Weak | HODLR | HSS |
| Strong | $\mathcal{H}$ | $\mathcal{H}^2$ |

## 3.1 Hierarchically off-diagonal low-rank matrix

The defining characteristic of hierarchically off-diagonal low-rank matrices (denoted by HODLR) is that the interaction between a cluster and any non-intersecting cluster can be efficiently represented by a low-rank interaction, i.e., if $\mathscr{C}_J \cap \mathscr{C}_I = \emptyset$, then $\mathscr{C}_J \in \mathcal{I}(\mathscr{C}_I)$, where $\mathcal{I}(\mathscr{C}_I)$ is the interaction list of cluster $\mathscr{C}_I$. Matrices arising out of interaction between points lying on a smooth one-dimensional manifold as shown in Figure 3.2 can be efficiently

| | Tree structure | Interaction list | Nested low-rank | Low-rank construction |
|---|---|---|---|---|
| HODLR | Any | Siblings | Maybe | Any |
| HSS | Any | Siblings | Yes | Any |
| $\mathcal{H}$ | Any | Cousins | Maybe | Any |
| $\mathcal{H}^2$ | Any | Cousins | Yes | Any |
| FMM | $2^d$-tree | Cousins | Yes | Analytic |

Table 3.2: Properties of different hierarchical matrices

represented as HODLR matrices, after an appropriate ordering of points. A 2-level HODLR

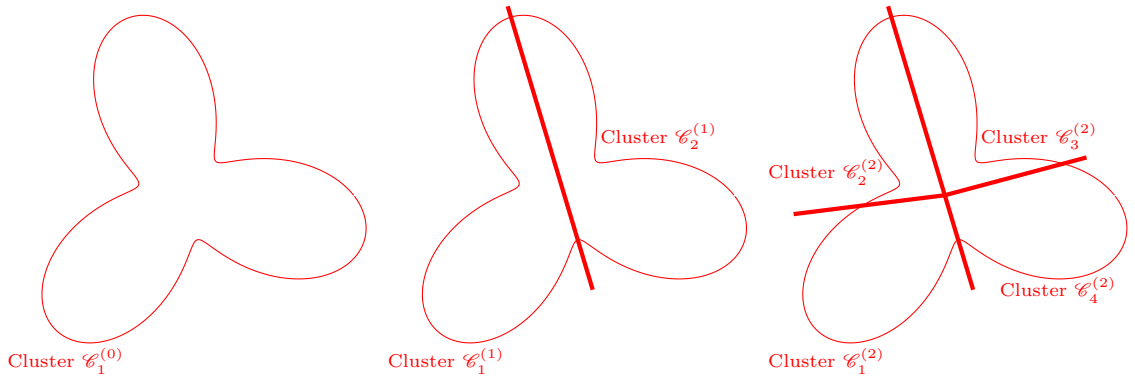



Figure 3.2: Partitioning of a smooth 1D manifold; Left: Level 0; Middle: Level 1; Right: Level 2.

matrix is one that can be written in the form shown in equation (3.2).

$$K = \begin{bmatrix} K_1^{(1)} & U_1^{(1)} K_{1,2}^{(1)} V_2^{(1)^T} \\ U_2^{(1)} K_{2,1}^{(1)} V_1^{(1)^T} & K_2^{(1)} \end{bmatrix} \tag{3.1}$$

$$= \begin{bmatrix} \begin{bmatrix} K_1^{(2)} & U_1^{(2)} K_{1,2}^{(2)} V_2^{(2)^T} \\ U_2^{(2)} K_{2,1}^{(2)} V_1^{(2)^T} & K_2^{(2)} \end{bmatrix} & U_1^{(1)} K_{1,2}^{(1)} V_2^{(1)^T} \\ U_2^{(1)} K_{2,1}^{(1)} V_1^{(1)^T} & \begin{bmatrix} K_3^{(2)} & U_3^{(2)} K_{3,4}^{(2)} V_4^{(2)^T} \\ U_4^{(2)} K_{4,3}^{(2)} V_3^{(2)^T} & K_4^{(2)} \end{bmatrix} \end{bmatrix} \tag{3.2}$$

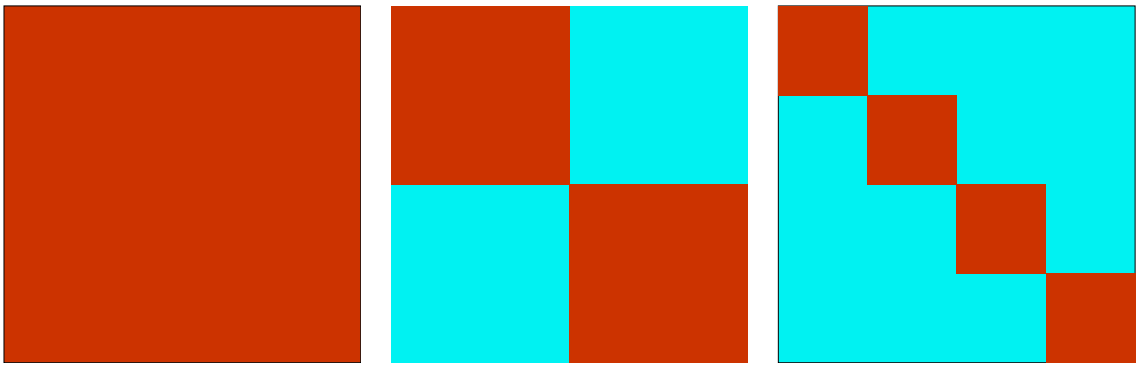
where $K_i^{(k)} \in \mathbb{R}^{N/2^k \times N/2^k}$, $K_{i,j}^{(k)} \in \mathbb{R}^{r \times r}$, $U_i^{(k)}, V_i^{(k)} \in \mathbb{R}^{N/2^k \times r}$ and $r \ll N$ is the rank of interaction between siblings.

In general, a $\kappa$-level HODLR matrix is one in which, the $i^{th}$ diagonal block at level $k$,

where $1 \leq i \leq 2^k$ and $0 \leq k < \kappa$, denoted as $K_i^{(k)}$, can be written as

$$K_i^{(k)} = \begin{bmatrix} K_{2i-1}^{(k+1)} & U_{2i-1}^{(k+1)} K_{2i-1,2i}^{(k+1)} V_{2i}^{(k+1)^T} \\ U_{2i}^{(k+1)} K_{2i,2i-1}^{(k+1)} V_{2i-1}^{(k+1)^T} & K_{2i}^{(k+1)} \end{bmatrix}. \tag{3.3}$$

The operators $K_{2i-1,2i}^{(k+1)}$ and $K_{2i,2i-1}^{(k+1)}$ are the *interaction operators*, $U_{2i-1}^{(k+1)}, U_{2i}^{(k+1)}$ are the *interpolation operators* and $V_{2i-1}^{(k+1)}, V_{2i}^{(k+1)}$ are the *anterpolation operators*. The maximum number of levels, in case of a perfectly balanced HODLR tree, is given by $\kappa = \lfloor \log_2(N/2r) \rfloor$. A pictorial description of the HODLR-matrix, corresponding to the binary tree in Figure 3.2,



(a) Level 0 (b) Level 1 (c) Level 2

Figure 3.3: HODLR-matrix at different levels for the corresponding tree in Figure 3.2.

is shown in Figure 3.3.

## 3.2 HSS matrix

The HSS matrix is a sub-class of the HODLR matrix with the additional property that the low-rank basis for the interaction of a node with its siblings can be constructed from the low-rank basis of the interaction of its children.

The recursive hierarchical structure of the HSS representation is seen when we consider a $4 \times 4$ block partitioning of a HSS matrix, $K$. The two-level HSS representation is shown

in equation (3.4).

$$K = \begin{bmatrix} K_1^{(1)} & U_1^{(1)} K_{1,2}^{(1)} V_2^{(1)^T} \\ U_2^{(1)} K_{2,1}^{(1)} V_1^{(1)^T} & K_2^{(1)} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} K_1^{(2)} & U_1^{(2)} K_{1,2}^{(2)} V_2^{(2)^T} \\ U_2^{(2)} K_{2,1}^{(2)} V_1^{(2)^T} & K_2^{(2)} \end{bmatrix} & \begin{bmatrix} U_1^{(2)} S_1^{(2)} \\ U_2^{(2)} S_2^{(2)} \end{bmatrix} K_{1,2}^{(1)} \begin{bmatrix} V_3^{(2)} R_3^{(2)} \end{bmatrix}^T \\ \begin{bmatrix} U_3^{(2)} S_3^{(2)} \\ U_4^{(2)} S_4^{(2)} \end{bmatrix} K_{2,1}^{(1)} \begin{bmatrix} V_1^{(2)} R_1^{(2)} \end{bmatrix}^T & \begin{bmatrix} K_3^{(2)} & U_3^{(2)} K_{3,4}^{(2)} V_4^{(2)^T} \\ U_4^{(2)} K_{4,3}^{(2)} V_3^{(2)^T} & K_4^{(2)} \end{bmatrix} \end{bmatrix} \tag{3.4}$$

Note that the interpolation operator, $U_i^{(k)}$ at level $k$, can be constructed from the interpolation operators of its children at level $k+1$.

$$U_i^{(k)} = \begin{bmatrix} U_{2i-1}^{(k+1)} S_{2i-1}^{(k+1)} \\ U_{2i}^{(k+1)} S_{2i}^{(k+1)} \end{bmatrix} \tag{3.5}$$

The operators $S_{2i-1}^{(k+1)}$ and $S_{2i}^{(k+1)}$ are termed the *downward pass operators*. Similarly, the anterpolation operator, $V_i^{(k)}$ at level $k$, can be constructed from the anterpolation operators of its children at level $k+1$.

$$V_i^{(k)} = \begin{bmatrix} V_{2i-1}^{(k+1)} R_{2i-1}^{(k+1)} \\ V_{2i}^{(k+1)} R_{2i}^{(k+1)} \end{bmatrix} \tag{3.6}$$

The operators $R_{2i-1}^{(k+1)}$ and $R_{2i}^{(k+1)}$ are termed the *upward pass operators*.

Most of the HODLR matrices can also be efficiently represented as a HSS matrix.

## 3.3 $\mathcal{H}$-matrix

The defining feature of the $\mathcal{H}$-matrix is that the interaction list of a cluster, $\mathscr{C}_I$, consists of other clusters that are well-separated. Most matrices arising out of interaction between points lying in $\mathbb{R}^d$, can be efficiently represented as $\mathcal{H}$-matrices. Note that the HODLR matrices form a strict subset of the $\mathcal{H}$ matrices. A pictorial description of the $\mathcal{H}$-matrix, corresponding to the binary tree in Figure 3.5, is shown in Figure 3.6.

## 3.4 $\mathcal{H}^2$-matrix

The $\mathcal{H}^2$-matrix is a sub-class of the $\mathcal{H}$-matrix with the additional property that the low-rank basis for the interaction of a node with its siblings can be constructed from the low-rank

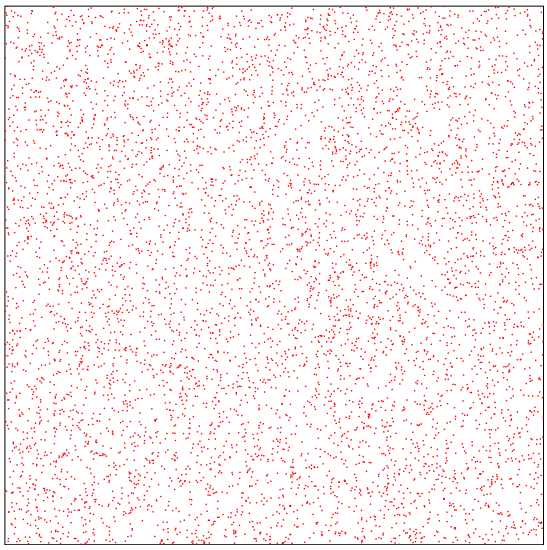


Figure 3.4: Points distributed uniformly random within a unit square

basis of the interaction of its children. Most of the $\mathcal{H}$-matrices arising in practice can also be efficiently represented as a $\mathcal{H}^2$-matrix.

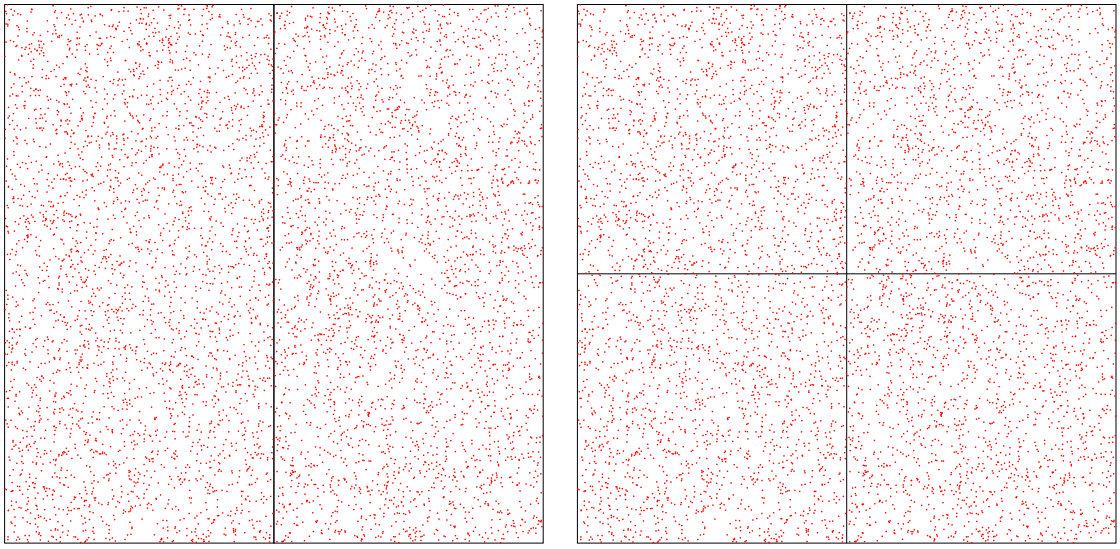
## 3.5 FMM-matrix

The FMM-matrix is a sub-class of the $\mathcal{H}^2$-matrix with the following additional properties:

1. A $2^d$ tree is used if the points lie on a $d$-dimensional manifold.
2. The interaction between siblings is strictly not low-rank.
3. Low-rank approximation is obtained using an analytic technique.

The storage and algorithm for constructing a FMM-tree is the same as $\mathcal{H}^2$-tree with the above additional properties enforced. Figure 3.7 shows the FMM-tree for a unit square and Figure 3.8 represents the corresponding matrices at the different levels. Note that FMM requires a quad tree to be used for a 2D manifold.

To start off, the entire square domain is our cluster. This is level 0 in the algorithm. At the next level i.e. at level 1, the entire square domain is divided into four equal clusters each being a smaller square domain. In general, the clusters at level $l+1$ are obtained by dividing each cluster at level $l$ into four equal squares. This is pictorially described in Figures 3.9



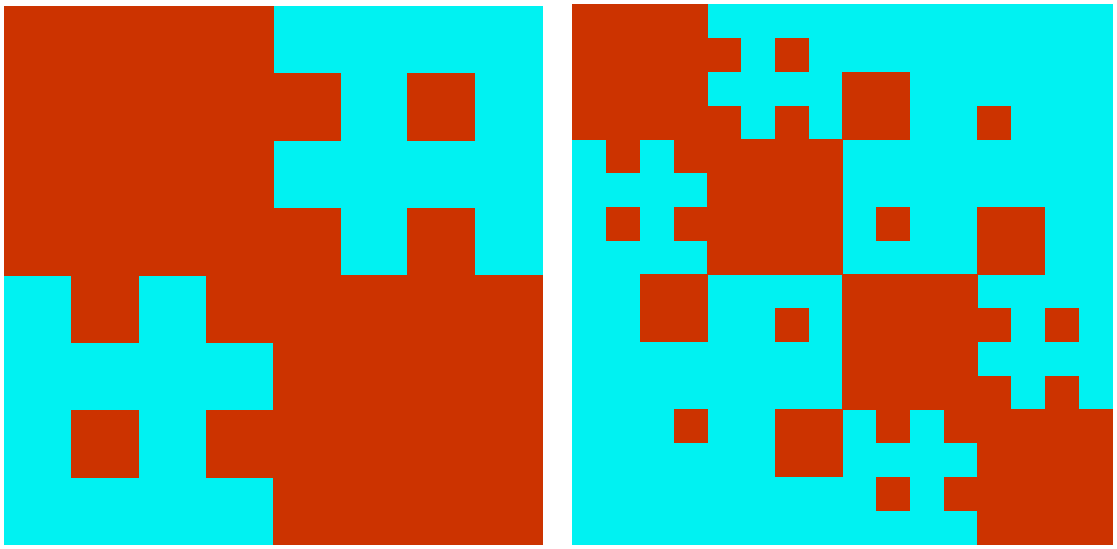
(a) Clusters at level 1

(b) Clusters at level 2

(c) Clusters at level 3

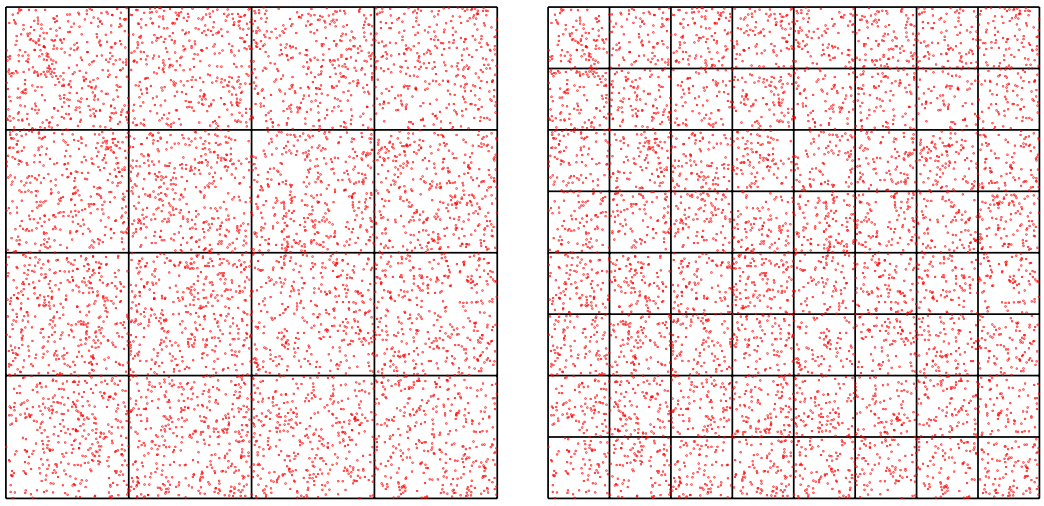
(d) Clusters at level 4

Figure 3.5: Different levels in the partitioning of a unit square using a binary $\mathcal{H}$-matrix tree.



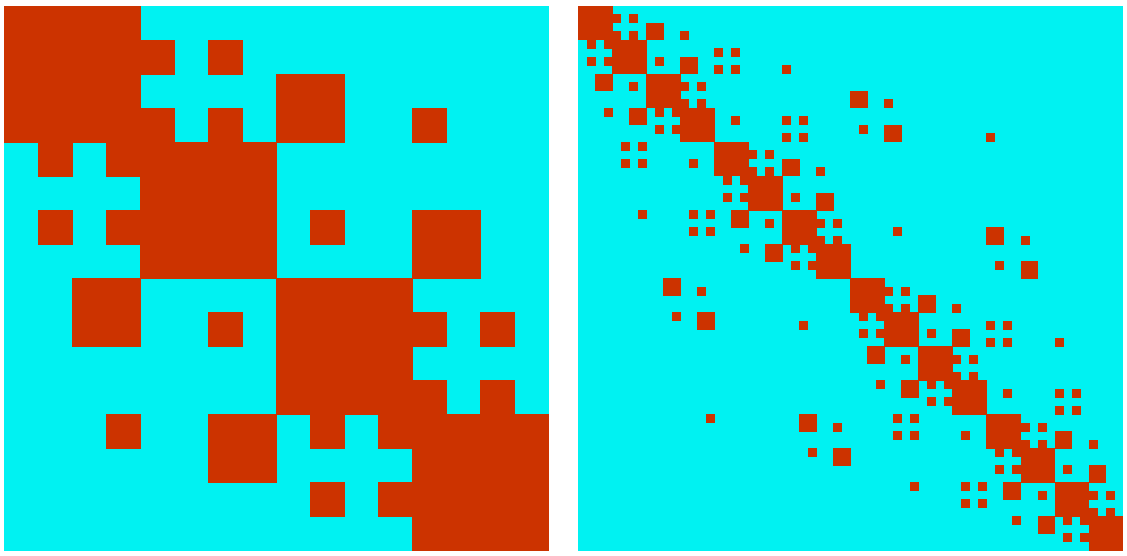
(a) Level 3 (b) Level 4

Figure 3.6: $\mathcal{H}$-matrix at levels 3 and 4 corresponding to the tree in Figure 3.5.



(a) Clusters at level 2 (b) Clusters at level 3

Figure 3.7: Different levels in the partitioning of a unit square using a quad tree for FMM.



(a) Level 2 (b) Level 3

Figure 3.8: FMM-matrix at levels 2 and 3 corresponding to the tree in Figure 3.7.

and 3.10. Note that for a cluster $\mathscr{C}$ at a given level, we can partition the remaining clusters at the same level into three mutually disjoint sets:

- $\mathcal{N}_{\mathscr{C}}$: the set of clusters that are neighbors to $\mathscr{C}$
- $\mathcal{I}_{\mathscr{C}}$: the set of clusters that are in the interaction list of $\mathscr{C}$
- $\mathcal{W}_{\mathscr{C}}$: the remaining clusters.

The three sets for a cluster at level 3 are indicated in Figure 3.10. Often in many practical applications, the distribution of charges is highly non-uniform. This would in-turn imply that the partition of the space must be slightly different to take into account the non-uniform distribution of the charges. An example of such a partition is shown in Figure 3.11. The partition of a cluster is terminated when the number of charges within a cluster reaches a threshold.

The algorithm 3.5 describes the general algorithm to construct any of the above hierarchical matrix structures. The total storage cost depends on whether the nested low-rank

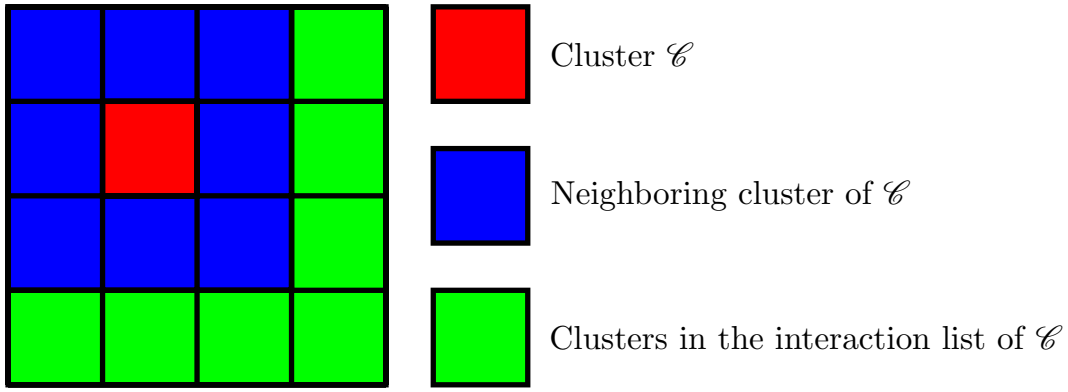



Figure 3.9: Pictorial representation of different sets of clusters at level 2. The interaction of the red cluster $\mathscr{C}$ with the clusters in the interaction list i.e. the green clusters, is computed using analytic techniques.

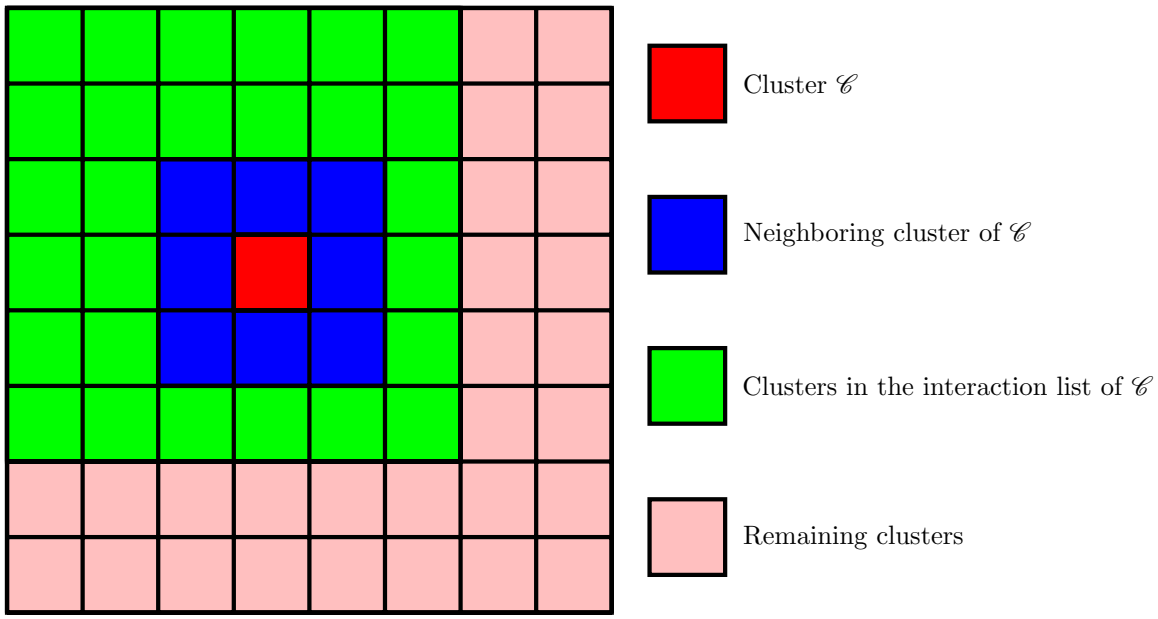



Figure 3.10: Pictorial representation of different sets of clusters at level 3. The interaction of the red cluster $\mathscr{C}$ with the clusters in the interaction list i.e. the green clusters, is computed using analytic techniques. The interaction of the red cluster with the pink clusters have already been computed at the previous level, level 2.

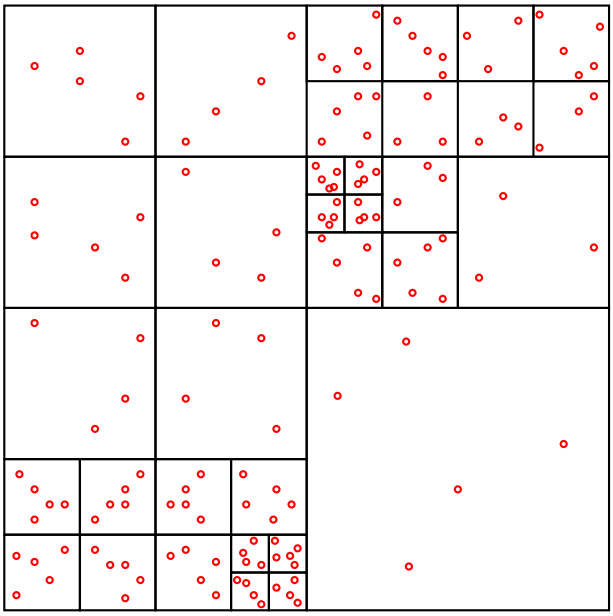


Figure 3.11: An unbalanced tree for a nonuniform distribution of charges.

---

**Algorithm 9** Construction of a hierarchical matrix tree.

---

1. Let the root of the tree represent all the points inside the domain as shown in Figure 3.2 or Figure 3.4.
2. Recursively, sub-divide the domain as shown in Figure 3.2 or Figure 3.5 or Figure 3.7.
3. For a cluster $i$ at level $k+1$,

**if** (The matrix has a nested low-rank structure) **then**

Store the upward & downward pass matrices, i.e., $S_i^{(k+1)}$ and $R_i^{(k+1)}$ and the interaction matrices $K_{i,j}^{(k+1)}$. The cost of storing these matrices is $\mathcal{O}\left(\left(r_i^{(k)}\right)^2\right)$.

**else**

Store the interpolation $(U_i^{(k+1)})$, anterpolation $(V_i^{(k+1)})$ and interaction $(K_{i,j}^{(k+1)})$ matrices, which enables the interaction of the cluster with clusters in its interaction list. The cost of storing these matrices is $\mathcal{O}\left(r_i^{(k)} N/2^k + \left(r_i^{(k)}\right)^2\right)$.

**end if**

4. Proceed till the number of points in a cluster is less than $n_{\min}$.
5. At the leaf level, store the matrix corresponding to the interaction of a cluster with itself (& possibly neighbors if the interaction with the neighbors is not low-rank) along with the interpolation $(U_i^{(\kappa)})$, anterpolation $(V_i^{(\kappa)})$ and interaction $(K_{i,j}^{(\kappa)})$ matrices. The cost to store these matrices at a single leaf is $\mathcal{O}(n_{\min}^2)$.

---

structure is present or not.

$$\text{Storage cost } = \begin{cases} \mathcal{O}(pN) & \text{if nested low-rank structure is present} \\ \mathcal{O}(pN \log_2(N)) & \text{if no nested low-rank structure} \end{cases}$$

where $p = \left(n_{\min} + \max_{i,k} \left\{ r_i^{(k)} \right\}\right)$.

# Chapter 4

# Fast linear inversion

## 4.1 Introduction

Large-scale stochastic inverse modeling has been drawing substantial attention in recent times, for example in the context of subsurface modeling. Inverse modeling is an essential ingredient in subsurface modeling due to the fact that direct measurements of hydrogeologic parameters such as conductivity and specific storage and variables such as pressure and solute concentration are expensive and sometime impossible to obtain. Engineering applications, such as remediation, real-time monitoring of $CO_2$ sequestration sites, deep-well injection projects and hydrocarbon reservoir exploitation, demand ever more accurate prediction of the unknowns such as hydraulic conductivity, hydraulic head, solute concentrations, and others.

Such inverse problems are ill-posed because they involve few measurements and a large number of unknowns and require solving underdetermined linear systems. A popular technique that has found numerous applications in the context of subsurface imaging for solving such underdetermined inverse problems is the Bayesian geostatistic approach [29, 73, 74, 76–78, 93, 99]. The motivation behind this approach is to combine the data obtained through measurements with a stochastic model of the structure of the function to be estimated. The structure of the underlying function is characterized through an a priori given probability density function. The prior probability density function is parameterized through variograms and generalized covariance functions. The approach also accounts for the fact that the measurements include errors and thus must not be reproduced exactly. One of the advantages of this approach is that it not only gives the "best" estimate, but

also allows us to quantify the uncertainty through confidence intervals or through functions that are samples from the posterior distribution (conditional realizations).

However, a major hurdle of this method is that, when the number of unknowns $m$ to be estimated is large, the method becomes computationally expensive. The bottleneck arises due to the fact that storage and matrix operations involving large dense covariance matrices scale as $\mathcal{O}(m^2)$, making application of the approach next to impossible for large-scale problems.

There have been various approaches to circumvent this difficulty. For instance, [44, 83, 94] make use of the fast Fourier transform to address this problem on a regular grid. The fast Fourier transform takes advantage of the fact that for a regular grid the covariance matrix has a Toeplitz or block-Toeplitz structure and this structure can be exploited to construct fast matrix vector products in $\mathcal{O}(m \log m)$. However, one of the drawbacks of the method is that its extension to other grids is non-trivial and most often in applications, such as when using the finite-element approach, we rely on non-uniform unstructured grids.

To deal with unstructured grids, [80] proposes an algorithm that relies on functional parameterization of the spatial random field by the Karhunen-Loève (KL) expansion. The spatial random field is parameterized by weighted base functions that are derived from the covariance function. The KL expansion relies on the assumption that the covariance function is smooth and that the correlation length is large compared to the size of the domain. In terms of matrix algebra, the KL expansion can be interpreted as approximating the large dense covariance matrix as a low-rank matrix. One of the drawbacks of this method is that the assumption that the covariance matrix can be approximated by a low-rank matrix is not always valid.

Recently, [82] circumvented the enormous computational cost by reformulating the problem under certain assumptions and by employing a sparse formulation for the generalized inverse of the covariance matrix, so that there is no longer a need for using the large dense covariance matrix.

In contrast, in our work, we propose an algorithm that takes advantage of the structure of large dense prior covariance matrices. Previously, computationally exploitable structure of these covariance and generalized covariance matrices, like isotropy and separability, have been studied in [121, 122]. However, the structure exploited by our proposed algorithm is a hierarchical low-rank structure, which is different from the aforementioned studies.

As stated earlier, the main stumbling block in the application of the geostatistical theory for inversion for large scale problems is that the cost of dense matrix vector products

increases quadratically with the number of unknowns $m$. In the past, the cost of performing dense matrix-vector products, especially in the context of boundary integral equations, have been reduced using fast summation techniques like the fast multipole method (FMM) [8, 27, 51, 92], the Barnes-Hut algorithm [5], panel clustering [58], FFT [28], wavelet based methods [17, 84, 85], etc. The literature on the FMM is vast and we refer the readers to [7, 8, 25, 27, 32, 33, 39, 52, 92, 120] for more details.

Over the last decade, the fast multipole method has been studied in terms of matrices known as hierarchical matrices. There are many different possible hierarchical structures, the most common of them being hierarchical semiseparable matrices [119], denoted as HSS, and hierarchical matrices [15, 49, 57, 59, 60], denoted as $\mathcal{H}$ and $\mathcal{H}^2$ matrices (where $\mathcal{H}^2$ requires somewhat stricter assumptions than $\mathcal{H}$, i.e., $\mathcal{H}^2 \subset \mathcal{H}$). The FMM has been recognized as a method that takes advantage of properties of $\mathcal{H}^2$ matrices to accelerate matrix-vector products. The generalized covariance matrices, that we consider, possess this underlying $\mathcal{H}^2$ structure. This is due to the fact the far field covariance can be well-represented by a smooth function. As a result, we can employ the fast multipole method proposed in [39] to compute matrix-vector products.

One of the main advantages of the fast multipole method discussed in [39] is that the algorithm is applicable to a wide range of covariance and generalized covariance functions. By using the $\mathcal{H}^2$ matrix structure, the computational cost of these dense matrix vector products are reduced from $\mathcal{O}(m^2)$ to $\mathcal{O}(m)$. Another desirable feature of our implementation is that our algorithm takes into account the sparsity of the measurement operator as discussed in section 4.3. This fact also plays a significant role in reducing the time taken to estimate the unknowns.

Our fast, new, stable algorithm for large-scale linear inversion problem relies on $\mathcal{H}^2$-matrix algebra to accelerate matrix-vector products. The algorithm is applicable when the number of unknowns $m$ is large — around $100,000$ — and the number of measurements is of the order of 100–500. The algorithm is applied to a linear inversion problem arising out of a real crosswell tomography application as discussed in [30]. The computational speedup gained by our algorithm allows us to (i) quantify the uncertainty in the solution (ii) optimize the capture geometry. Quantifying uncertainty is crucial in computational modeling and simulation, especially in the context of inverse problems, to enable accurate assessment of the solution and to take decisions. The computational speedup gained also allows us to analyze different capture geometries and determine the optimal capture geometry. In most

of the linear inversion problems, it is imperative to have a good capture geometry to get the maximum possible information for a given number of measurement devices. However, the optimization procedure involves assessing a sequence of different computational geometries. Due to the computational expense of conventional algorithms for stochastic linear inversion, analyzing a single capture geometry in itself is a computationally intensive task. Using our algorithm, optimizing the capture geometry becomes tractable. We illustrate this by optimizing the capture geometry for a real crosswell tomography application [30].

To sum up, the major contributions of our work are the following. The algorithm proposed by us results in a significant reduction in computational cost compared to the conventional algorithms, thus enabling the Bayesian geostatistic approach for large-scale problems. The computational speedup gained enables us to make new estimates such as quantifying the uncertainty in the solution and optimizing the capture geometry, which were previously inaccessible due to its enormous computational cost. We highlight the performance of our algorithm in section 4.5 by comparing our implementation with a conventional algorithm for a real crosswell tomography problem discussed in [30]. The fast algorithm enables us to provide an optimal capture geometry for this crosswell tomography problem [30]. To give a quantitative sense of the capability and to highlight the potential of the proposed algorithm, the proposed algorithm can solve a crosswell tomography problem (Section 4.3) with $250,000$ unknowns on an ordinary desktop in less than 20 minutes. The conventional algorithm on the other hand takes nearly 2 hours to solve a problem with just $40,000$ unknowns. The algorithm was implemented in C++ and is made available at `http://sivaramambikasaran.github.io/FLIPACK2D/`.

The remainder of this chapter is organized as follows. Section 4.2 introduces the preliminary ideas, i.e., stochastic linear inversion in subsection 4.2.1 and the hierarchical matrices in subsection 4.2.2 to solve large-scale stochastic linear inversion problems. Section 4.3 discusses the crosswell tomography problem. Section 4.4 presents a discussion of the general algorithm and its computational cost. This is then followed by Section 4.5, which explains how the algorithm discussed in Section 4.4 is applied to solve the crosswell tomography problem. It also presents the numerical results obtained by the algorithm both on synthetic and real data sets.

## 4.2 Preliminary ideas

In this section, we discuss the two key ingredients for large-scale stochastic linear inversion.

### 4.2.1 Stochastic linear inversion

The stochastic Bayesian approach is briefly introduced. This is a general method [77] to solve underdetermined linear systems arising out of linear inverse problems.

#### 4.2.1.1 Prior

Let $s(x)$ be the function to be estimated at location $x$. Its "structure" is represented through the prior probability density function. The basic model of the function to be estimated is taken as

$$s(x) = \sum_{k=1}^{p} f_k(x)\beta_k + \epsilon(x). \tag{4.1}$$

The first term is the prior mean, where $f_k(x)$ are known functions, typically step functions, polynomials, and $\beta_k$ are unknown coefficients where $k \in \{1, 2, \ldots, p\}$. The second term $\epsilon(x)$ is a random function with zero mean and characterized through a covariance function. In a sense, the first part is the deterministic part and the second part is the stochastic part. This model is especially popular in geostatistics and in other data analysis approaches. The *linear model* is represented by Equation (4.1). For instance, the zonation/regression approach is obtained by setting

$$f_k = \begin{cases} 1, & \text{if in zone } k \\ 0, & \text{otherwise} \end{cases},$$

where $k \in \{1, 2, \ldots, p\}$, $p$ being the number of zones. The covariance of $\epsilon$ is set to zero. In the regression approach, the variability is described through deterministic functions. In stochastic approaches, one describes at least some of the variability through the stochastic part (while still retaining flexibility in the use of the deterministic part).
After discretization (e.g., through application of finite-difference and finite element models), $s(x)$ is represented by a vector $s \in \mathbb{R}^m$. The mean of $s$ is given by

$$\mathbb{E}[s] = \mathbf{X}\beta \qquad \text{(notation: } \mathbb{E} \text{ is the expectation)},$$

where $\mathbf{X} \in \mathbb{R}^{m \times p}$ is the drift matrix and $\beta \in \mathbb{R}^{p \times 1}$ are the $p$ unknown drift coefficients. The covariance matrix of $s$ is given by

$$\mathbb{E}\left[(s-\mu)(s-\mu)^T\right] = Q.$$

The entries of the covariance matrix are given by $Q_{ij} = K(x_i, x_j)$, where $K(\cdot,\cdot)$ is a generalized covariance function, which must be conditionally positive definite. For a more detailed discussion on permissible covariance kernels, we refer the reader(s) to the following references: [14, 71, 72, 75, 110].

#### 4.2.1.2 Measurement equation

The observation/measurement is related to the unknown by the linear relation

$$y = Hs + v, \tag{4.2}$$

where $H \in \mathbb{R}^{n \times m}$ is the *observation/measurement matrix*, $v \in \mathbb{R}^{n \times 1}$ is a Gaussian random vector of observation/measurement error independent from $s$, with zero mean and covariance matrix $R$, i.e., $v \sim \mathcal{N}(0, R)$. Typically, the matrix $R$ is a diagonal matrix since each measurement is independent of other measurements.

The form mentioned in Equation (4.2) is encountered frequently in practice, because many important inverse problems are linear "deconvolution problems." Furthermore, many nonlinear problems are solved through a succession of linearized problems.

The prior statistics of $y$ are obtained as shown below. The mean is given by

$$\mu_y = \mathbb{E}\left[Hs + v\right] = H\mathbb{E}\left[s\right] + \mathbb{E}\left[v\right] = HX\beta = \Phi\beta,$$

and the covariance is given by

$$\begin{aligned}\Psi &= \mathbb{E}\left[\left(H\left(s - X\beta\right) + v\right)\left(H\left(s - X\beta\right) + v\right)^T\right] \\ &= HQH^T + R.\end{aligned}$$

The $y$ to $s$ cross-covariance

$$C_{ys} = \mathbb{E}\left[\left(H\left(s - X\beta\right) + v\right)\left(s - X\beta\right)^T\right] = HQ.$$

#### 4.2.1.3 Bayesian Analysis

The prior probability distribution function is modeled as a Gaussian, i.e.,

$$p(s|\beta) \propto \exp\left(-\frac{1}{2}\left(s - X\beta\right)^T Q^{-1}\left(s - X\beta\right)\right).$$

To express that $\beta$ is unknown a priori, its prior probability density function is modeled uniformly over all space, i.e.,

$$p(\beta) \propto 1,$$

and thus

$$p(s, \beta) \propto \exp\left(-\frac{1}{2}\left(s - X\beta\right)^T Q^{-1}\left(s - X\beta\right)\right).$$

The likelihood function is then given by

$$p(y|s) \propto \exp\left(-\frac{1}{2}\left(y - Hs\right)^T R^{-1}\left(y - Hs\right)\right).$$

Thus, the posterior probability density function is

$$\begin{aligned} p(s, \beta) \propto \exp\left(-\frac{1}{2}\left(s - X\beta\right)^T Q^{-1}\left(s - X\beta\right)\right) \times \\ \exp\left(-\frac{1}{2}\left(y - Hs\right)^T R^{-1}\left(y - Hs\right)\right), \end{aligned}$$

which is again a Gaussian. The negative log of the posterior probability density function is

$$\begin{aligned} \mathcal{L} &= -\ln\left(p(s, \beta)\right) \\ &= \frac{1}{2}\left(s - X\beta\right)^T Q^{-1}\left(s - X\beta\right) \\ &+ \frac{1}{2}\left(y - Hs\right)^T R^{-1}\left(y - Hs\right) + \text{ constant.} \end{aligned}$$

The posterior mean values, indicated by $\hat{s}$ and $\hat{\beta}$ minimize $\mathcal{L}$. Setting the partial derivative of $\mathcal{L}$ with respect to $s$ and $\beta$ to zero, we get the following equations:

$$\begin{aligned} (\hat{s} - X\hat{\beta})^T Q^{-1} - (y - H\hat{s})^T R^{-1} H = 0 \\ -\left(\hat{s} - X\hat{\beta}\right)^T Q^{-1} X = 0 \end{aligned} \tag{4.3}$$

To bring the solution to a computationally convenient form, we introduce a vector $\xi \in \mathbb{R}^{n\times 1}$ defined through

$$y - HX\hat{\beta} = \Psi\xi. \tag{4.4}$$

We then have

$$\boxed{\hat{s} = X\hat{\beta} + QH^T\xi.} \tag{4.5}$$

Substituting the above into Equation (4.3), we get

$$\xi^T HX = 0. \tag{4.6}$$

Combining the equations (4.4) and (4.6), we get that

$$\boxed{\begin{bmatrix} \Psi & \Phi \\ \Phi^T & 0 \end{bmatrix} \begin{bmatrix} \xi \\ \hat{\beta} \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix}.} \tag{4.7}$$

Thus, the solution is obtained by solving a system of $n + p$ linear equations. The key equations are Equations (4.5) and (4.7).

To quantify the uncertainty in the solution, the posterior covariance matrix is given by

$$\begin{aligned} V = Q - QH^T P_{yy} HQ - XP_{bb}X^T - XP_{yb}^T HQ \\ - QH^T P_{yb} X^T, \end{aligned} \tag{4.8}$$

where $P_{yy}, P_{yb}, P_{bb}$ are obtained as

$$\begin{bmatrix} P_{yy} & P_{yb} \\ P_{yb}^T & P_{bb} \end{bmatrix} = \begin{bmatrix} \Psi & \Phi \\ \Phi^T & 0 \end{bmatrix}^{-1}. \tag{4.9}$$

The diagonal entries of the matrix $V$ enable us to quantify the uncertainty, since each diagonal entry represents the variance of each of the unknowns.

#### 4.2.1.4 Computational cost

Our goal is to obtain the best estimate $\hat{s}$ and the diagonal entries of the matrix $V$ efficiently. To do so, we first need to solve the linear system in Equation (4.7) and then obtain $\hat{s}$ using Equation (4.5). To quantify the uncertainty in the solution, we need to obtain the diagonal entries of $V$ using Equation (4.8) and Equation (4.9).

In all these equations, **we need the matrix matrix product $QH^T$, which is the bottleneck in terms of computational cost.** A conventional direct algorithm, where the number of measurements, $n$, is much smaller than the number of unknowns, $m$, would proceed as shown in Algorithm 10.

Since we have $p \ll n \ll m$, the total computational cost is $\mathcal{O}(nm^2 + n^2m)$.

The cost to solve Equation (4.7) is independent of $m$. Hence, once the linear system is constructed, it can be solved by any conventional direct method like Gaussian elimination, which costs $\mathcal{O}((n+p)^3)$. Since we are interested in the case where the number of measurements is relatively small, i.e., $n \approx 200$, we are not interested in optimizing the solving phase, since the cost is independent of $m$. Also, the cost to compute a single diagonal entry of $V$ from Equation (4.8) is independent of $m$. Hence, the bottleneck is the cost $\mathcal{O}(nm^2)$, which arises from the computation of the matrix-matrix product $Q_H = QH^T$. The storage cost is dominated by the dense matrix $Q$, which costs $\mathcal{O}(m^2)$ to store. **Hence, our focus will be on efficiently storing $Q$ and efficiently constructing $Q_H = QH^T$, since this is the bottleneck in the algorithm.** Some typical values of $p$, $n$ and $m$ encountered in practice are $p \sim$ 1–3, $n \sim$ 100–200 and $m \sim 10^5$–$10^6$.

### 4.2.2 Hierarchical matrices

In this section, we briefly describe the analytic and computational foundations of hierarchical matrices, which we will be using in this chapter. Hierarchical matrices are data-sparse approximations of dense matrices arising in applications like boundary integral equations, interpolation, etc. Hierarchical matrices, introduced by Hackbusch, et al. [15, 49, 57, 59, 60], are generalizations of the class of matrices for which the fast multipole method is applicable. An example of a hierarchical matrix arising out of a two-dimensional application is shown in Figure 4.1. Of the class of hierarchical matrices, $\mathcal{H}^2$-matrices are precisely matrices for which the fast multipole method (FMM) is applicable. The FMM is a numerical technique

to calculate matrix-vector products (or) equivalently sums of the form

$$f(x_i) = \sum_{j=1}^{N} K(x_i, y_j)\sigma_j,$$

where $i \in \{1, 2, 3, \ldots, M\}$, in $\mathcal{O}(M+N)$ operations as opposed to $\mathcal{O}(MN)$ with a controllable error $\epsilon$. The FMM was originally introduced by Greengard and Rokhlin [51] based on Legendre polynomial expansions and spherical harmonics for the kernel $K(\vec{x}, \vec{y}) = \dfrac{1}{|x-y|}$. Historically the FMM has been used for the treatment of integral operators (boundary

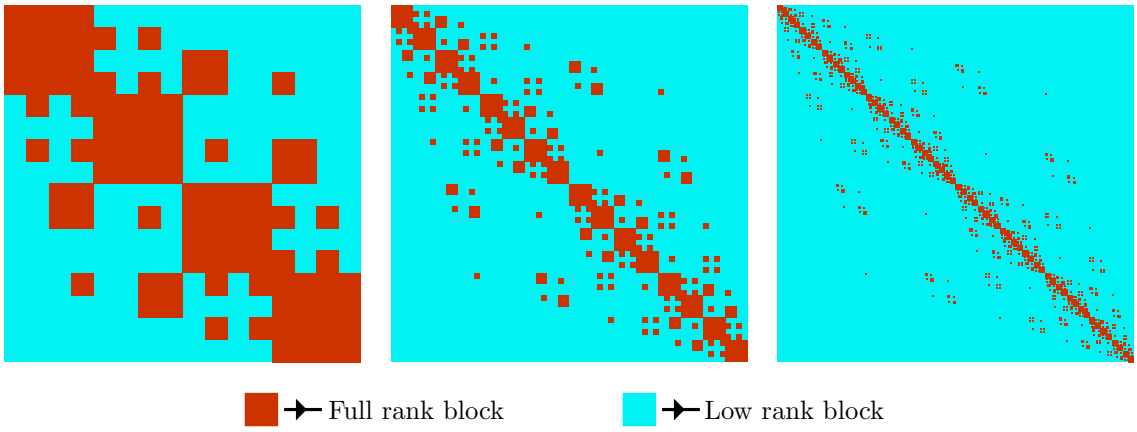



Figure 4.1: Hierarchical matrices arising out a 2D problem at different levels in the tree.

element method) and the solution of the resulting linear systems, which arise when solving partial differential equations, such as the Laplace, Helmholtz or Maxwell equations. It has been applied in a variety of fields including molecular dynamics, astrophysics, elastic materials, in graphics, etc. In all these cases, the FMM is used, as part of an iterative solution of linear systems (GMRES, conjugate gradient), to accelerate matrix-vector products. This is often the part of the calculation that is computationally the most expensive.

In the current context, the FMM will be used to construct the linear system, i.e., more precisely in constructing $QH^T$ as discussed in the previous section. If the measurement matrix $H$ is sparse, then the FMM can also be easily adapted to exploit the sparsity of the measurement matrix. Once the linear system has been constructed, since the size of the linear system is relatively small, the linear system can be solved directly using conventional direct solvers like the Gaussian elimination. One of the main advantages of FMMs, in

general, is that they can be easily applied to non-uniform distributions of points. The FMM also comes with sharp *a-priori* error bounds.

As mentioned above, the original FMM is based on certain analytical expansions of the kernel $K(r)$, for example with spherical harmonics, powers of $r$, Bessel functions, etc. However it has since been realized that the FMM is based on a more general idea (see for example [39]), which is that certain matrices can be well-approximated using a low-rank matrix. More precisely, the key property behind the FMM is that the interactions $K(x_i, y_j)$ between far away clusters of points (far field interaction) can be efficiently represented using low-rank approximations. In general, matrices from boundary element method or the covariance matrix, $Q$, in our problem are full-rank. However, certain off-diagonal blocks (associated with well-separated clusters of points) can be efficiently represented using a low-rank matrix. This implies that the covariance matrix $Q$ can be well-represented as a hierarchical matrix. An example of a hierarchical matrix is shown in Figure 4.1.

Generally speaking, given a matrix $A \in \mathbb{R}^{M\times N}$ (in our case a sub-matrix of $Q$), the *optimal* rank $p$ approximation can be obtained from its singular value decomposition (SVD) [47]. However, SVD is computationally expensive with a cost of $\mathcal{O}(MN\min(M,N))$. In recent years, there has been a tremendous progress [26, 43, 54, 90] in constructing fast approximate low-rank factorizations for matrices. Techniques like adaptive cross approximation [102] (ACA), pseudo-skeletal approximations [48], interpolatory decomposition [39], randomized algorithms [43, 81, 117] provide great ways for constructing efficient approximate low-rank representations.

For smooth kernels, Chebyshev polynomials have proven to be an attractive way to construct low-rank approximations and we refer the readers to [39, 89] for more details. In our work, we follow the fast multipole method discussed in [39]. One of the main reasons for choosing [39] is that the method is very general and is applicable for a wide range of kernels/covariance functions. Another advantage of [39] is that the precomputation cost is very small since the method is based on interpolating from Chebyshev nodes, for which efficient numerical algorithms are available. We will be making use of this fast multipole method coupled with the sparsity of the matrix $H$, to compute the product $QH^T$.

We briefly explain why the FMM is applicable to this class of problems. Given $m$ points in space with position vector, $\{\vec{r}_k\}_{k=1}^m$, the covariance matrix $Q$ is typically given as $Q(i,j) = K(\|\vec{r}_i - \vec{r}_j\|)$. For instance, some of the popular choices of $K(r)$ are $\exp(-r)$, $\frac{1}{r}$, $\exp(-r^2)$, $\frac{1}{1+r^2}$, etc. All the above mentioned covariance kernel functions are amenable

to the use of the FMM, since the interaction between well-separated clusters of points can be well captured by modeling these interactions as low-rank interactions. This low-rank interaction enables us to reduce both the storage and computational cost. Theorems are available [39] to determine the rank $p$ required to achieve an accuracy $\varepsilon$. Roughly speaking, the kernel $K$ needs to be a smooth function, that is with a controllable growth in the complex plane away from the real line [39].

## 4.3 Problem specification

In this section, we describe the specific problem, which we solve by large-scale linear inversion. The problem deals with large scale crosswell tomography to monitor the $CO_2$ plume in $CO_2$ sequestration sites. The problem configuration is shown in Figure 4.2. The motivation for this capture geometry stems from [30]. We will compare the results we obtain using our fast large scale linear inversion algorithm with the results in [30], which were obtained by running the forward model using TOUGH2, a multiphase-flow reservoir model [101] and patchy rock physical model [115].

A crosswell tomography is setup with $n_s$ sources along the vertical line $AB$ and $n_r$ receivers along the vertical line $CD$. The sources emit a seismic wave and the receivers measure the time taken by the seismic wave from a source to hit the receiver. This is done for each source-receiver pair. Our goal is to image the slowness of the medium inside the rectangular domain. Slowness is defined as the reciprocal of the speed of the seismic wave in the medium. In the context of $CO_2$ sequestration for example, the seismic wave travels considerably slower through $CO_2$ saturated rock [30, 31, 38] as opposed to the rest of the medium. Hence by measuring the slowness in the medium, we can estimate the $CO_2$ concentration at any point in the domain (with some uncertainty) and thereby the location of the $CO_2$ plume.

The above configuration is typical for most of the continuous crosswell seismic monitoring sites. We go about modeling the problem as follows. As a first order approximation, the seismic wave is modeled as traveling along a straight line from the sources to receivers with no reflections/refractions. The time taken by the seismic wave to travel from the source to the target is measured. Each source-receiver pair gives us a measurement and hence there are a total of $n = n_s \times n_r$ measurements. To obtain the slowness in the domain $ABDC$, the domain is discretized into $m$ grid points (i.e., an $m_x \times m_y$ grid such that $m = m_x m_y$) and

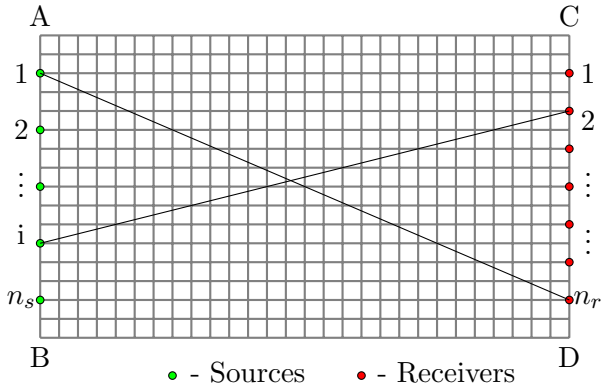



Figure 4.2: Discretization of the domain between the two wells. The line segment $AB$ denotes the well where the $n_s$ sources are placed, while the line segment $CD$ denotes the well where the $n_r$ receivers are placed. The seismic wave is generated by a source and the receiver measures the time taken for it to receive the seismic wave. Each source-receiver pair constitutes a measurement.

within each cell the slowness is assumed to be constant. Let $y_{ij}$ denote the time taken by the seismic wave to travel from source $i$ to receiver $j$. Let $s_k$ be the slowness of the $k^{th}$ cell. For every source-receiver $(i, j)$ pair, we then have that

$$\sum_k l_{ij}^k s_k + v_{ij} = y_{ij}, \tag{4.10}$$

where $l_{ij}^k$ denotes the length traveled by the ray from source $i$ to the receiver $j$ through the $k^{th}$ cell and $v_{ij}$ denotes the measurement error $i \in \{1, 2, \ldots, n_s\}$, $j \in \{1, 2, \ldots, n_r\}$, $k \in \{1, 2, \ldots, m\}$. Equation (4.10) can be written in a matrix-vector form:

$$Hs + v = y,$$

where $H \in \mathbb{R}^{n \times m}$, $s \in \mathbb{R}^{m \times 1}$, $v \in \mathbb{R}^{n \times 1}$ and $y \in \mathbb{R}^{n \times 1}$. Typically, the measurement error is modeled as a Gaussian white-noise. For most problems of practical interest, the number of measurements $n$ is much smaller than the number of unknowns $m$, i.e., $n \ll m$. This under-determined linear system constitutes our inverse problem.

We will now analyze the matrix $H$ to figure out the structure of the linear system. Each row of the matrix $H$ corresponds to a source-receiver pair. Consider the source $i$ and receiver $j$ where $i \in \{1, 2, \ldots, n_s\}$ and $j \in \{1, 2, \ldots, n_r\}$. This corresponds to the $((i-1)n_r + j)^{th}$ row

of the matrix $H$. Since we have modeled the wave traveling from a source to a receiver as a straight line without reflections/refractions, the non-zero entries along each row correspond to the cells hit by the ray from a source to the receiver. Since the wave from the source to receiver travels along a straight line, only the cells that lie on this straight line contribute to the non-zero entries. Hence, every row of $H$ has only $\mathcal{O}(\sqrt{m})$ non-zeros. Hence, the matrix $H$ is sparse since it has only $\mathcal{O}(n\sqrt{m})$ entries as opposed to $\mathcal{O}(nm)$ entries. We would hence like to take advantage of this sparsity to accelerate our computations. This underdetermined system is solved using Bayesian approach discussed in subsection 4.2.1. Since we would also like to characterize the fine-scale features, $m$ can be much larger than $n$. Section 4.4 discusses the fast algorithm to solve the above problem using Bayesian approach.

## 4.4 Algorithm

In this section, we discuss in detail the fast algorithm for the large scale linear inverse problem using Bayesian geostatistical approach and how this algorithm is applied to the crosswell tomography problem. We would like to point out that the algorithm is far more general and can be used for other linear inverse problems and not just the crosswell tomography case.

As discussed in section 4.2.1.4, the main bottleneck in the entire computation is constructing the matrix product $QH^T$, where $Q \in \mathbb{R}^{m\times m}$ is the covariance matrix and $H \in \mathbb{R}^{n\times m}$ is the measurement matrix. Let

$$H = \begin{bmatrix} h_1 & h_2 & \cdots & h_n \end{bmatrix}^T,$$

where $h_i \in \mathbb{R}^{m\times 1}$ are column vectors, each corresponding to a measurement. As mentioned in 4.2.2, the fast multipole method can be used to accelerate matrix-vector product computations, because the sub-matrices, of a dense covariance matrix $Q$, corresponding to the interaction between well-separated clusters can be efficiently represented using low-rank matrices.

Now note that

$$QH^T = \begin{bmatrix} Qh_1 & Qh_2 & \cdots & Qh_n \end{bmatrix}.$$

Hence, a straight-forward way to accelerate the computation of the matrix-matrix product $QH^T$ is to accelerate the computation of each matrix vector product $Qh_i$, where $i \in \{1, 2, \ldots, n\}$. The black box FMM [39] reduces the cost of computing $Qh_i$ for each $i$ from

$\mathcal{O}(m^2)$ to $\mathcal{O}(m)$. (For more details on the algorithm, see page 8716–17 of [39].) Hence, the total cost to compute the matrix-matrix product $QH^T$ is just $\mathcal{O}(nm)$ as opposed to $\mathcal{O}(nm^2)$. As discussed in 4.2.1.4, this step is the bottleneck in the computation, since the number of measurements is much smaller than the number of unknowns, i.e., $n \ll m$. This is described in Algorithm 10. With the FMM, the first step is no longer the bottleneck, while the second

**Algorithm 10** Computational cost for different steps in the large scale linear inversion problem.

| **Operation** | **Computational cost using different methods** | | |
|---|---|---|---|
| | Conventional | FMM | FMM with sparsity |
| $Q_H = QH^T$ using the FMM | $\mathcal{O}(nm^2)$ | $\mathcal{O}(nm)$ | $\mathcal{O}(nm)$ |
| $\tilde{\Psi} = HQ_H$ | $\mathcal{O}(n^2m)$ | $\mathcal{O}(n^2m)$ | $\mathcal{O}(n^2\sqrt{m})$ |
| $\Psi = \tilde{\Psi} + R$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ |
| $\Phi = HX$ | $\mathcal{O}(nmp)$ | $\mathcal{O}(nmp)$ | $\mathcal{O}(n\sqrt{m}p)$ |
| Solve (4.7) to get $\xi$ and $\beta$ | $\mathcal{O}((n+p)^3)$ | $\mathcal{O}((n+p)^3)$ | $\mathcal{O}((n+p)^3)$ |
| $\hat{s} = X\beta + Q_H\xi$ | $\mathcal{O}(mp+mn)$ | $\mathcal{O}(mp+mn)$ | $\mathcal{O}(mp+mn)$ |
| Diagonal entries of $V$ from (4.11) | $\mathcal{O}(n^2m)$ | $\mathcal{O}(n^2m)$ | $\mathcal{O}(n^2m)$ |

$$V(i,i) = Q(i,i) - \underbrace{Q_H(i,:)P_{yy}(Q_H(i,:))^T}_{\mathcal{O}(n^2)} - \underbrace{X(i,:)P_{bb}(X(i,:))^T}_{\mathcal{O}(p^2)} - \underbrace{2X(i,:)P_{yb}^T(Q_H(i,:))^T}_{\mathcal{O}(pn)} \tag{4.11}$$

step becomes the limiting step. The overall computational cost scales as $\mathcal{O}(n^2m)$. This computational cost cannot be further reduced if the measurement matrix $H$ is assumed to be dense.

However, if the matrix $H$ happens to be sparse, the fast multipole method can be easily adapted to exploit this sparsity and thereby further reduce the computational complexity. For instance, in the case of the crosswell tomography application, every row of $H$, i.e., $\{h_i^T\}_{i=1}^n$ has only $\mathcal{O}(\sqrt{m})$ non-zeros as opposed to $\mathcal{O}(m)$ entries. This is due to the fact that the non-zero entries along a row of the measurement matrix $H$ are due to a seismic wave from a source to a receiver (see explanation in the previous section).

We can take advantage of this sparsity twice. First, in the FMM, although the asymptotic cost is $\mathcal{O}(nm)$ irrespective of whether $H$ is sparse or not, the constant in front of $nm$ is reduced with a sparse $H$. This is due to the fact that many source leaves in the FMM tree are empty (that is do not intersect the ray). Second, the computational cost for the second

step, which was the bottleneck in algorithm 10, can be reduced from $\mathcal{O}(n^2m)$ to $\mathcal{O}(n^2\sqrt{m})$ using a sparse matrix-vector product technique such as the compressed sparse row (CSR) format [35, 105] (e.g., since there are only $\mathcal{O}(\sqrt{m})$ non-zero entries in each row of $H$). See Algorithm 10.

Hence, the large-scale linear inversion problem has an overall computational complexity of $\mathcal{O}(nm + n^2\sqrt{m})$.

To quantify the uncertainty in the solution, we need to obtain the diagonal entries of $V$ using Equations (4.8) and (4.9). As before, once $Q_H = QH^T$ has been obtained, computing the uncertainty of a single cell costs us only $\mathcal{O}(n^2)$ from Equation (4.11). Hence, the total cost of computing the uncertainty for all the cells grows linearly with the number of cells and is given by $\mathcal{O}(n^2m)$.

## 4.5 Numerical benchmark

We now present numerical benchmarks for two cases. The first one is a synthetic data set for the crosswell tomography problem while the second one is a real data set for the crosswell tomography problem. The problem set up for both the cases is similar to the one shown in Figure 4.2, i.e., the sources are distributed along the vertical line $AB$ and the receivers are distributed along the vertical line $CD$. The goal is to image the slowness in both the cases.

### 4.5.1 Synthetic data set

We first present results for a synthetic data set. In the case of a synthetic data set, the domain we consider is a $70m \times 40m$ rectangular domain, i.e., the horizontal distance between the sources and receivers is $70m$. To benchmark our algorithm, we feed in a known slowness field as shown in Figure 4.3. We compute our measurements as follows. We distribute 12 uniformly spaced sources along $AB$, i.e., $n_s = 12$ and 24 uniformly spaced receivers along $CD$, i.e., $n_r = 24$. This gives us a total of 288 measurements. We compute the time taken for the seismic wave to travel from each source to a receiver. We add a random Gaussian noise with mean 0 and variance $10^{-4}$ to each of the measurement to model the measurement error. The covariance function $K(r)$ is taken as the Gaussian covariance, i.e., $K(r) = \exp(-(r/\theta)^2)$ and the matrix $X$ is taken as a vector with all 1's. The correlation length $\theta$ is set to 10. As a next step, the FMM requires the number of Chebyshev nodes along each direction as an input. In this case, for the Gaussian, the number of Chebyshev nodes in each direction is

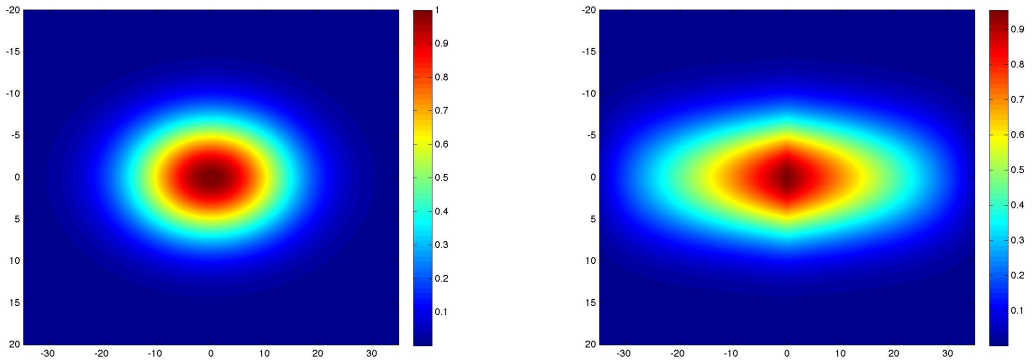


Figure 4.3: Left: known slowness field — synthetic data set; Right: reconstructed slowness field for the synthetic data set using a million grid points.

taken as 5.

We now discretize the domain into $\sqrt{m} \times \sqrt{m}$ grid points and reconstruct the image. The number of grid points $m$ is varied from $2,500$ to $1,000,000$ and the time taken by the fast algorithm to image the slowness is compared against the conventional direct algorithm. The comparison of the time taken is tabulated in Table 4.1. For $m = 1,000,000$, the reconstructed image using the fast algorithm is shown on the right side in Figure 4.3. Note that there is a discrepancy between the true and reconstructed slowness for the synthetic data set. This discrepancy shows that in case of limited source-receiver pairs, it is highly important to optimize the capture geometry. This optimization of the capture geometry has been done for the real data set. Once the capture geometry has been optimized, grid refinement is crucial to study the true evolution of the plume. Also, the discrepancy between the true and reconstructed image will further be reduced if we have more measurements, in which case again refining the grid is important. Table 4.1 also compares the estimation error obtained by the stochastic linear inversion using the direct algorithm and our fast algorithm. The relative estimation error is defined as $\|s_{\text{reconstructed}} - s_{\text{true}}\|/\|s_{\text{true}}\|$. As the table indicates, there is very little difference in the estimation error between the conventional direct algorithm and our fast algorithm. This is in fact a strong argument in favor of fast algorithms. In the presence of large modeling errors, we can afford to approximate the covariance matrix, which is what our algorithm precisely does.

Figure 4.5 compares the time taken and storage cost of a conventional direct algorithm

Table 4.1: Comparison of time taken and relative error for the synthetic test case shown in Figure 4.3. The relative error is computed by comparing the reconstructed image (right panel in Figure 4.3) and the true solution (left panel). We chose the approximation order in the FMM such that the FMM error is small compared to the inverse algorithm error. As a result the relative error for the direct and fast algorithms are comparable. A more complete error convergence is shown for the real test case in Figure 4.4 and 4.9. The algorithm was implemented in C++. All the calculations were run on a machine with a single core 2.66 GHz processor and 8 GB RAM. There was no necessity to parallelize the implementation for the present purpose due to the speedup achieved with the proposed algorithm.

| Grid size | Time taken in secs | | Relative error | |
|---|---|---|---|---|
| $m$ | Direct algorithm | Fast algorithm | Direct algorithm | Fast algorithm |
| $2,500$ | $2.1 \cdot 10^{+1}$ | $2.3 \cdot 10^{+1}$ | $4.45 \cdot 10^{-1}$ | $4.39 \cdot 10^{-1}$ |
| $10,000$ | $3.9 \cdot 10^{+2}$ | $6.7 \cdot 10^{+1}$ | $4.39 \cdot 10^{-1}$ | $4.41 \cdot 10^{-1}$ |
| $22,500$ | $2.0 \cdot 10^{+3}$ | $1.4 \cdot 10^{+2}$ | $3.82 \cdot 10^{-1}$ | $3.89 \cdot 10^{-1}$ |
| $40,000$ | $6.4 \cdot 10^{+3}$ | $2.6 \cdot 10^{+2}$ | $3.47 \cdot 10^{-1}$ | $3.39 \cdot 10^{-1}$ |
| $250,000$ | – | $1.3 \cdot 10^{+3}$ | – | $2.85 \cdot 10^{-1}$ |
| $1,000,000$ | – | $5.1 \cdot 10^{+3}$ | – | $2.69 \cdot 10^{-1}$ |

with the fast algorithm proposed by us. Note that the plots are on a log-log scale and clearly indicate that the time taken and storage cost for the conventional algorithm scales as $\mathcal{O}(m^2)$ whereas the time taken and storage cost for the novel fast algorithm scales as $\mathcal{O}(m)$. We

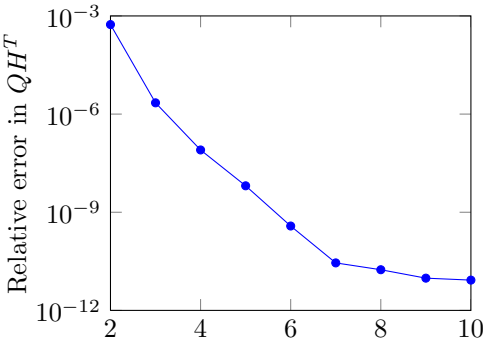



Figure 4.4: Relative error in $QH^T$ versus the number of Chebyshev nodes along one direction for the Gaussian covariance for $m = 40,000$.

also perform an analysis of the time taken and the storage cost by our fast algorithm when

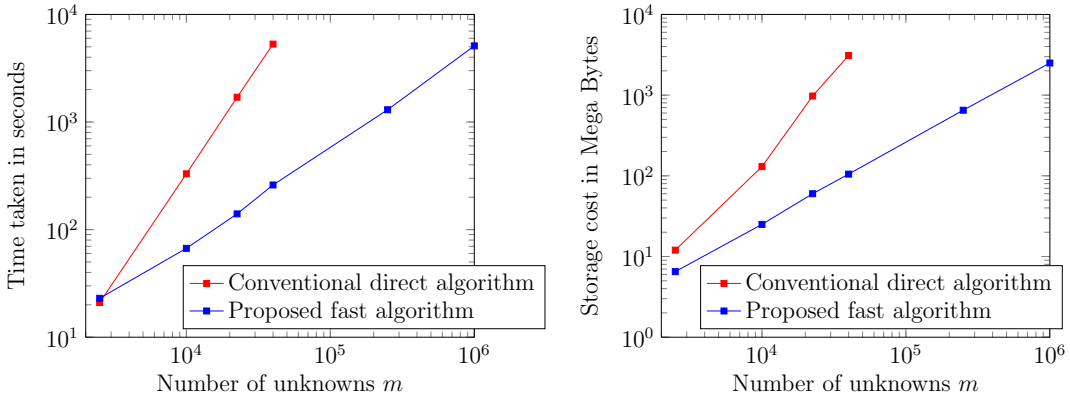



Figure 4.5: Left: comparison of the time taken by the fast algorithm vs the conventional direct algorithm; Right: comparison of the storage cost.

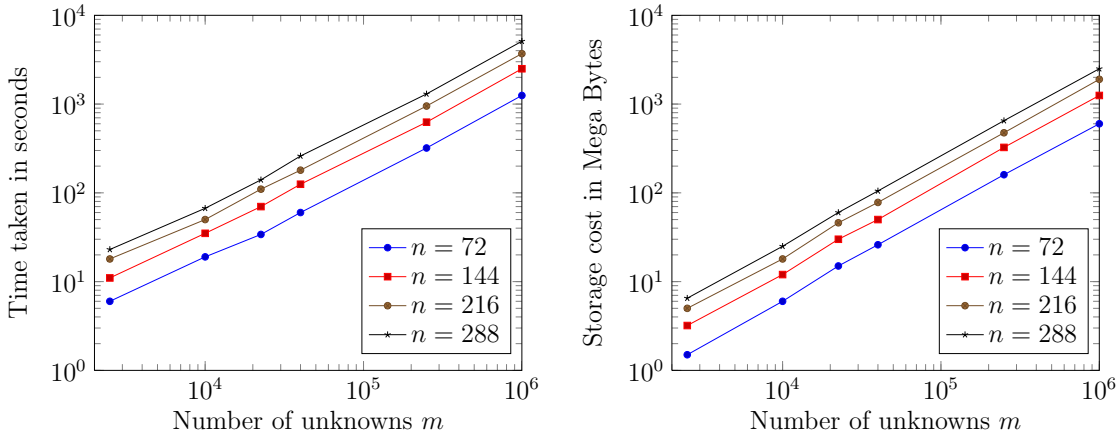



Figure 4.6: Left: time taken by the fast algorithm as a function of the number of unknowns; Right: storage cost by the fast algorithm as a function of the number of unknowns.

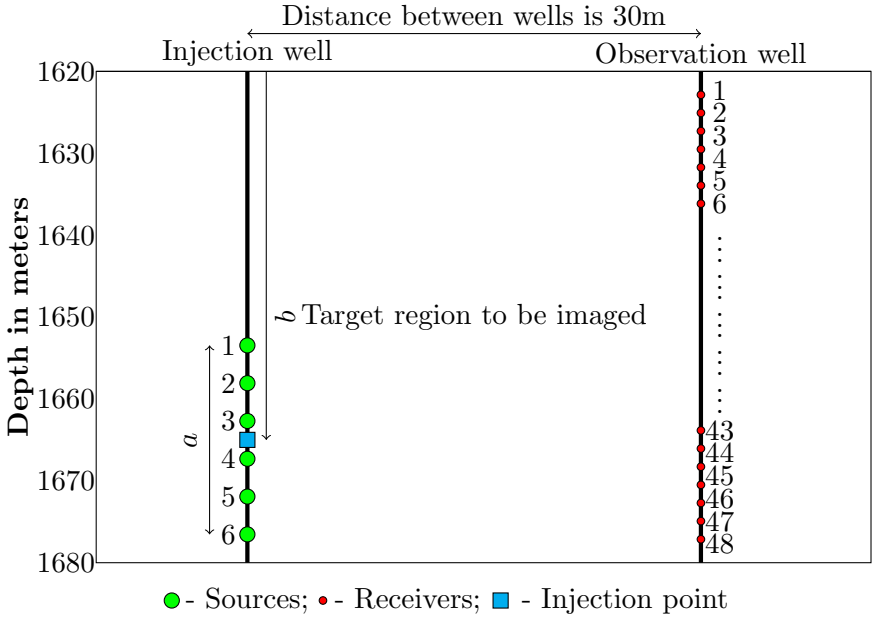



Figure 4.7: The schematic on the left is the actual crosswell geometry from [30].

varying the number of measurements, $n$. The results are plotted in Figure 4.6. This further validates the fact that the time taken and the storage cost scales only linearly with the number of unknowns $m$, irrespective of the number of measurements $n$. Note that all the plots are on a log-log scale.

### 4.5.2 Real data set

We now present detailed numerical benchmarking for a real data set. The linear inverse problem arises from monitoring a $CO_2$ plume in the subsurface for safe carbon storage. The site is the Frio II test site in southeast Texas near the Gulf of Mexico. The experiment is described in [30].

We briefly describe the capture geometry. There are two wells, an injection well and a monitoring well-separated by 30 meters at a depth of around $1,600$ m. To be precise, we consider a depth ranging from $1,620$ m to $1,680$ m. The $CO_2$ injection occurs at $1,665$ m underground. The experiment measures the time taken by the seismic wave from a series of sources to the receivers. Using the seismic traveltime measured as constraints, a series of forward flow models governing the migration are produced by a state-of-the-art model,

the TOUGH2/ECO2N system [100, 101]. This gives us the $CO_2$ plume induced slowness at different time instances. We obtained this data set, i.e., the location of the $CO_2$ plume after 120 hours, from one of our collaborators, Jonathan Ajo-Franklin, from Lawrence Berkeley National Laboratories. The data obtained is shown in Figure 4.10. For more details on the test site and the capture geometry, we refer the readers to [30]. A pictorial representation of the test site is shown in Figure 4.7.

From the data, i.e., the $CO_2$ plume after 120 hours of injection, the measurements to perform the linear inversion are obtained as follows. Given the $CO_2$ plume configuration after 120 hours, for a fixed capture geometry, we measure the time taken from a source to a receiver by modeling that the ray from the source to the receiver travels in a straight line. The signal to noise ratio of each of the measurements is typically around 65 dB. This dictates the variance of the measurement error in our measurements, which is set at $10^{-5}$. These are the diagonal entries of the matrix $R$. The covariance function we chose here is $K(r) = -r/\theta$, where the correlation length, $\theta$, is 90 m. The matrix, $X$, is taken as a vector with all 1's. As a next step, the fast multipole method requires the number of Chebyshev nodes along each direction as an input, which we set equal to 5. This means that the rank of interaction between well-separated clusters is 25. We then discretize the domain into $m$ grid points to reconstruct the image. Once we reconstruct the image, we are able to obtain the uncertainty in the solution.

Note that in the above procedure, optimization of the capture geometry [1], i.e., the placement of the sources and receivers, is of prime importance in crosswell tomography applications, especially when the number of measurements is not large, since it has a direct bearing on the resolution of the image obtained. We place 6 sources and 48 receivers. The source transducers are typically more expensive around \$50,000. Hence, there is a large number of receivers, in our case 48, compared to the sources. Because of this, we place the receivers along the monitoring well such that they are uniformly spaced. However, since we only have a few sources (6), it is important for us to choose where exactly we need to place these 6 sources. Hence, we need to optimize the source locations. The unknown parameters we want to optimize are the array length of the sources, $a$, and the location of the source array center, $b$. Refer to Figure 4.7 for more details. Given this set of 6 sources and 48 receivers, for a fixed $a$ and $b$, we have a total of $6 \times 48 = 288$ measurements. We now use these 288 measurements to perform the linear inversion for each pair $(a, b)$.

To optimize for the parameters '$a$' and '$b$', we proceed as follows. We consider a $237 \times 217$

grid. For a fixed '$a$' and '$b$', we estimate the relative error using

$$F(a,b) = \|s_{\text{computed}} - s_{\text{true}}\| \,/\, \|s_{\text{true}}\|.$$

The above can be computed since we already have access to the true data. This is the objective function we are minimizing, i.e.,

$$(a_{\text{opt}}, b_{\text{opt}}) = \text{argmin}_{a,b} F(a,b).$$

Note that in the current context, since we have access to the true slowness, we can compute the exact relative error. In applications where we do not have access to the true slowness, we could resort to the method of gradient desecent to find the local minimum of the relative error. Note that in the context of gradient descent as well, different capture geometries need to evaluated, which would be computationally impractical if we were to rely on the conventional algorithm.

The key point of this calculation is not to demonstrate an algorithm to optimize the placement of sensors but rather show how fast algorithms will be useful. Although computing one solution for a mid-size problem, e.g., a grid of size $200 \times 200$, is feasible without a fast algorithm, it would not be practical to calculate many solutions in the context of an optimization loop. Without the fast algorithm, **we would require close to 2 hours to obtain the solution and evaluate the objective function for a single capture geometry. With our fast algorithm in place, we are able to obtain the solution and evaluate the objective function for 25 different capture geometries in the same 2 hours.** The comparison will be even starker if we want a finer resolution. To find the optimal value for '$a$' and '$b$', we evaluate $F(a,b)$ for a range of $a$ and $b$. Each evaluation requires computing $QH^T$, which is efficiently done using the FMM. The plot of $F(a,b)$ versus '$a$' and '$b$' is shown in Figure 4.8.

We also mention that if we were to tackle three dimensional problems, the size of the 3D mesh would probably make a single calculation using the direct method infeasible. We obtained the optimal array length as $a_{\text{opt}} = 18$ m and the center of the source array to be at $b_{\text{opt}} = 1665$ m, i.e., this choice of '$a$' and '$b$' minimizes the relative error $F(a,b)$. We fixed the center of the source array at $b = 1665$ m $= b_{\text{opt}}$ and the array length at $a = 20$ m (this is very close to the optimum and there is no significant difference in the estimation relative error), and performed the following numerical benchmarks.

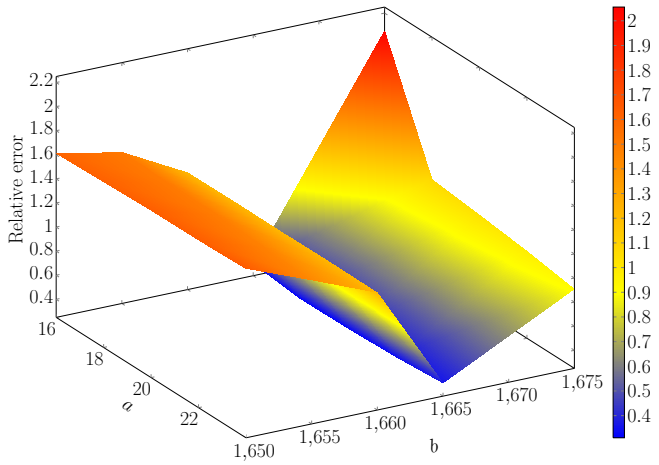



Figure 4.8: Relative error for the reconstructed image (compared to the exact answer in the left panel of Fig. 4.10) as a function of $a$, the length of the source array, and $b$, the location of the center of the source array. We used 25 evaluation points (5 along $a$ and $b$).

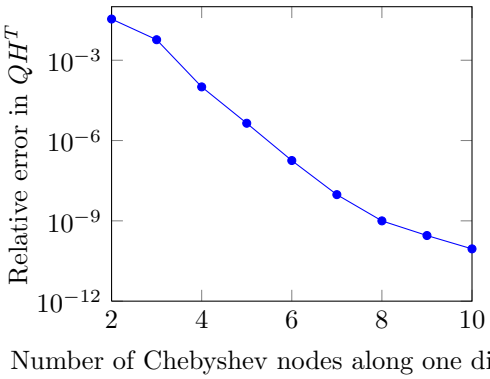



Figure 4.9: Relative error in $QH^T$ versus the number of Chebyshev nodes along one direction for $m = 237 \times 217 = 51,429$.

Table 4.2: Comparison of the computational time for the real test case of Figure 4.10

| Grid size | Time taken in seconds | |
|---|---|---|
| $m$ | Conventional direct algorithm | Fast algorithm |
| $55 \times 59$ | $2.7 \cdot 10^{+1}$ | $3.1 \cdot 10^{+1}$ |
| $117 \times 109$ | $4.0 \cdot 10^{+2}$ | $9.2 \cdot 10^{+1}$ |
| $237 \times 217$ | $6.4 \cdot 10^{+3}$ | $2.8 \cdot 10^{+2}$ |
| $500 \times 500$ | – | $1.1 \cdot 10^{+3}$ |

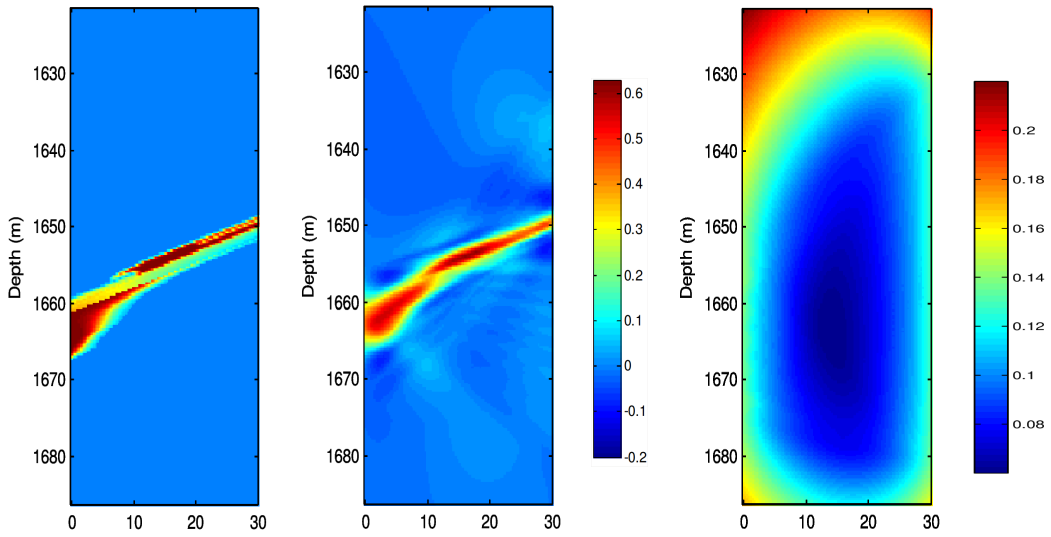



Figure 4.10: Left: true slowness using TOUGH2/ECO2N [100, 101] simulations, 120 hours after injection; Middle: reconstructed slowness using our proposed algorithm for the optimal choice of $a$ and $b$; Right: uncertainty in the estimated solution. All these plots are for a $237 \times 217$ grid. Each unit of slowness corresponds to $10^4$ sec/m.

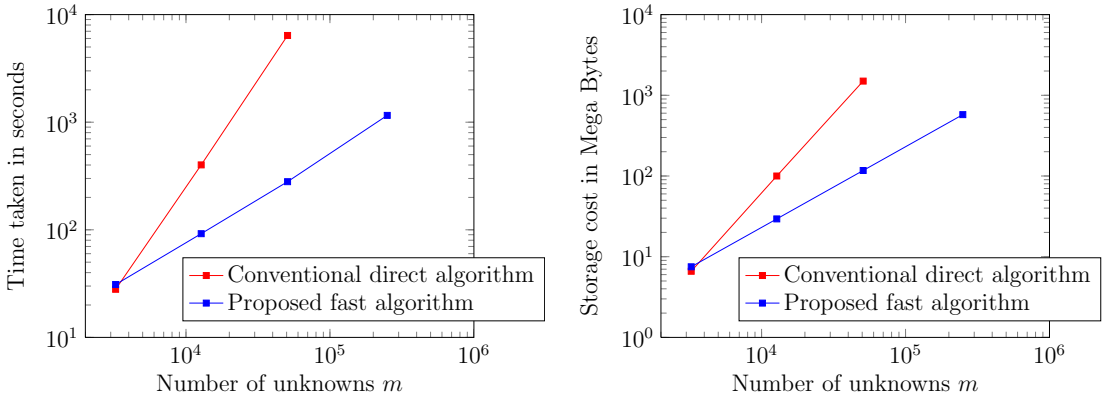



Figure 4.11: Left: time taken by the fast algorithm and the conventional direct algorithm; Right: storage cost.

First, we performed validation on the matrix matrix product obtained using the FMM. The Figure 4.9 shows the decay in the relative error in the matrix-matrix product $QH^T$ using the FMM and the direct matrix-matrix product vs the number of Chebyshev nodes along one direction, i.e.,

$$\epsilon = \|QH^T_{\text{computed}} - QH^T_{\text{exact}}\| \,/\, \|QH^T_{\text{exact}}\|,$$

for a $237 \times 217$ grid. The covariance kernel is given by $-r/\theta$. In the rest of our calculation we chose 5 Chebyshev nodes in each direction, which corresponds to an error around $10^{-6}$. This is significantly below the reconstruction error due to the limited number of measurements.

For $a = 20$ m and $b = 1665$ m, we plot the image of the estimated slowness. This is shown in Figure 4.10. There seems to be a good agreement with the true data shown in Figure 4.10. The uncertainty in the image reconstructed is shown on the right panel of Figure 4.10.

We now compare the time taken by the fast algorithm to image the slowness and the time taken by the conventional direct algorithm for different grid sizes keeping the array length and the center of the sources fixed at $a = 20$ m and $b = 1665$ m. The comparison of the time taken is tabulated in Table 4.2 and the corresponding figure comparing the time taken and storage cost is also shown.

## 4.6 Conclusions

We have presented a novel fast algorithm for large scale linear inversion. In particular, we solve a problem with few measurements ($n$) and a large number of unknowns ($m$), i.e., $n \ll m$. The algorithm is illustrated by applying it to a synthetic data set (tomography problem), where we estimate one million unknowns. The fast algorithm is also applied for a real data set, from a crosswell tomography of a $CO_2$ sequestration site, to estimate 250,000 unknowns. The reconstruction using the fast linear inversion agrees well with the actual data. The computational speedup achieved by our algorithm allows us to, for example, (i) quantify the uncertainty in the solution; (ii) optimize the capture geometry. Optimizing the capture geometry would not have been possible without the fast algorithm, which highlights the potential impact of this type of technique. Further note that the entire algorithm is mostly a matrix-free approach, once the covariance kernel is given in an analytic form. This is of critical importance, especially in large-scale real-time monitoring applications, where

the measurement matrix, $H$, might also be given in matrix free form. The matrix-matrix product $QH^T$, which is all we need for the inversion, can then be obtained using a matrix-free approach. This algorithm can also be combined with different filtering algorithms, thereby enabling large-scale real-time data-assimilation and inversion possible, which is the subject of our future work.

The fast algorithm for linear inversion hinges on the stochastic Bayesian approach and the hierarchical matrix approach. The algorithm can also exploit the underlying sparsity of the measurement operator. This in turn also results in a significant reduction in runtime. The new algorithm wins on two important counts. The speedup gained by using this algorithm is very significant, as demonstrated in the numerical benchmarks, and the storage is minimized tremendously. The speedup is especially significant when the number of unknowns $m$ is very large.

# Chapter 5

# Fast Kalman filtering

Kalman filtering is frequently used in many fields for sequential data-assimilation problem. Kalman filter estimates the current state of a time evolving process based on the measurements at each time instant and the observed history of the process. The Kalman filtering has two significant steps: (i) Prediction step; (ii) Update step. When the covariance matrix is dense, both these steps are computationally expensive with a computational cost of $\mathcal{O}(nm^2 + n^2m)$, where $m$ is the number of underlying unknowns and $n$ is the number of measurements. Typically, we have $n \ll m$. The computational cost becomes prohibitively expensive when $m$ is large, which is often the case in real sequential data-assimilation problems, especially in the context of geosciences. In our work, we propose an $\mathcal{O}(n^2m)$ Kalman filter. The effectiveness of the proposed Kalman filtering algorithm is demonstrated by solving a realistic crosswell tomography problem and a synthetic problem by formulating them as a stochastic linear inverse problem. In both the above cases, the sparsity of the measurement operator can be exploited to further reduce the computational time taken though the overall complexity of the proposed Kalman filtering algorithm remains the same as $\mathcal{O}(n^2m)$. We perform numerical benchmarking of our algorithm by comparing it with the conventional exact Kalman filter and the ensemble Kalman filter.

## 5.1 Introduction

In many fields of engineering and applied science, it is imperative to have good filtering and estimation techniques. A good filtering technique is needed in all applications, to remove unwanted noise from measurements. Once an efficient filtering is done, it is important to

make use of the filtered data to estimate the unknowns and make *good* predictions. When the underlying system can be modeled efficiently as a linear dynamical system, the *Kalman filter* gives us the *minimum mean-square error estimator*. The Kalman filter makes use of the underlying system dynamics and also the measurements obtained at every time instant to form an estimate of the current state of the system. The popularity of the Kalman filter is due to the fact on one hand it is relatively simple to understand and implement while on the other hand, it is an extremely powerful tool. In particular, we highlight that the proposed algorithm is much better than the ensemble Kalman filtering algorithm.

In general, any efficient data assimilation technique incorporates both the knowledge of the system dynamics and also the observations to obtain the best estimate of the current state of the system. However, in certain cases, for instance, when the underlying system is too complex to model, a random walk forecast model is adopted.

Kalman filter is the most widely used filtering techniques to obtain an estimate of the current state of a system from the observed history and measurements. Filtering is desirable in many situations in engineering and embedded systems. For example, any signal/measurement is corrupted with noise. A good filtering algorithm can remove the noise from the measurements while retaining the "useful" information. The Kalman filter is a tool that can estimate the variables of a wide range of processes. In mathematical terms, the Kalman filter provides the "optimal" estimate of the state of a linear system. The Kalman filter not only works well in practice, but it is theoretically attractive because it can be shown that of all possible filters, it is the one that minimizes the variance of the estimation error. Kalman filters are often implemented in embedded control systems because in order to control a process an accurate estimate of the process variables is needed.

## 5.2 Preliminary ideas

In this section, we briefly introduce the conventional Kalman filtering algorithm for a linear dynamical system. The overarching idea of Kalman filtering is to estimate the state of a time evolving system based on its previous states and noisy measurements. For a linear dynamical system, the Kalman filter provides *the linear unbiased, minimum mean-square estimator*.

### 5.2.1 Linear dynamical system

We will fix some notations first. Let $m$ be the number of unknowns. Let $s_t \in \mathbb{R}^{m\times 1}$ denote the state of the system after $t$ time-steps. A linear dynamical system with noise is of the form

$$s_{t+1} = Fs_t + w_t \tag{5.1}$$

where $F \in \mathbb{R}^{m\times m}$ is the state transition model and $w_t \in \mathbb{R}^{m\times 1}$ is the noise which is typically modeled as a multivariate Gaussian random vector with covariance $Q \in \mathbb{R}^{m\times m}$. In general, the state transition model, $F$, and the covariance $Q$ could be a functions of time as well. However, in most applications these are independent of time. The state transition model, $F$, captures the underlying physical model of the expected evolution of the system, while the covariance matrix $Q = \mathbb{E}\left(w_t w_t^T\right)$ represents the state noise of the system.

### 5.2.2 Measurement equation

The measurement equation is modeled as a linear equation of the form

$$y_t = Hs_t + v_t \tag{5.2}$$

where $y_t \in \mathbb{R}^{n\times 1}$ is the observation at time-step $t$, $H \in \mathbb{R}^{n\times m}$ is the measurement matrix and $v_t \in \mathbb{R}^{n\times 1}$ is the measurement error. Note that the number of measurements is $n$. The measurement $v_t \in \mathbb{R}^{n\times 1}$ is often modeled as a zero mean Gaussian random vector with covariance $R$. Again, in general the matrix $R$ could be a function of time as well. However, in most problems of practical interest, it is independent of time. The form mentioned in (5.2) is encountered frequently in practice, because many important inverse problems are linear "deconvolution problems." Further, more often, many nonlinear problems are solved through a succession of linearized problems.

The Kalman filter estimates the evolution of the state of the system, $s_t$, from the evolution Equation (5.3) and the measurement Equation (5.2). The Kalman filter has two significant steps: (i) Predict and (ii) Update. The prediction step, as the name suggests, predicts the state of the system at the next time step using *only* information from the current time step. The estimate obtained from the prediction step is termed as the *a priori* state estimate. The update step combines this prediction with the measurements obtained at the next time step, to refine the estimate of the state of the system at the next time step. The estimate

obtained from the update step is termed as the *a posteriori* state estimate. The *a posteriori* state estimate can be thought of as a weighted average of the *a priori* state estimate and the estimate of the system obtained from the measurements.

### 5.2.3 Algorithm and computational cost

The conventional Kalman filtering algorithm gives us an estimate of the unknown state of the system $s_t$ at each time step $t$. The conventional algorithm, along with its computational cost, for the linear dynamical system with random walk forecast model and time independent measurement operator is presented in algorithm 11. In all the cases, we will be dealing with, we have the number of measurements, $n$, to be much smaller than the number of unknowns, $m$, i.e. $n \ll m$. Let $\hat{s}_{t_1|t_2}$ and $\hat{P}_{t_1|t_2}$ denote the estimate of the state, $s_{t_1}$, and the posterior covariance, $P_{t_1}$, of the system at time-step $t_1$, given the measurements up-to and including the first $t_2$ time-steps.

**Algorithm 11** Conventional Kalman filtering algorithm

**Predict:**

| Operation | | Cost |
|---|---|---|
| ***a priori* state estimate** | $\hat{s}_{t+1\|t} = F_{t+1}\hat{s}_{t\|t}$ | - |
| ***a priori* covariance estimate** | $\hat{P}_{t+1\|t} = F_{t+1}\hat{P}_{t\|t}F_{t+1}^T + Q_{t+1}$ | $\mathcal{O}(m^2)$ |

**Measurement update:**

| Operation | | Cost |
|---|---|---|
| **Kalman gain** | $K_{t+1} = \hat{P}_{t+1\|t}H_{t+1}^T\left(H_{t+1}\hat{P}_{t+1\|t}H_{t+1}^T + R\right)^{-1}$ | $\mathcal{O}(nm^2 + n^2m)$ |
| ***a posteriori* state** | $\hat{s}_{t+1\|t+1} = \hat{s}_{t+1\|t} + K_{t+1}\left(y_{t+1} - H_{t+1}\hat{s}_{t+1\|t}\right)$ | $\mathcal{O}(nm)$ |
| ***a posteriori* covariance** | $\hat{P}_{t+1\|t+1} = \hat{P}_{t+1\|t} - K_{t+1}H_{t+1}\hat{P}_{t+1\|t}$ | $\mathcal{O}(nm^2)$ |

### 5.2.4 Computationally efficient Kalman filter

In this subsection, we discuss our new computationally efficient Kalman filter algorithm. There are couple of key observations, which we take advantage of to reduce the computational cost of Kalman filtering for the random walk forecast model. The first observation is that the *initial covariance matrix can be well-represented as a $\mathcal{H}^2$-matrix*. This enables us to compute the matrix matrix product $QH^T$ in $\mathcal{O}(nm)$ as opposed to $\mathcal{O}(nm^2)$ using the fast

multipole method [39], thereby reducing the computational cost of the first step in the Kalman filtering algorithm. The details on how to accelerate computing $QH^T$ using the $\mathcal{H}^2$-matrix approach is discussed in the previous chapter 4. It is also worth noting that the algorithm discussed in the previous chapter 4 also exploits the sparsity of the measurement operator $H$ in reducing the running time. The second key observation that enables us to reduce the computational cost of the subsequent steps in the Kalman filtering algorithm is the fact that *it is enough to store and compute the cross-covariance matrices, rather than storing and computing with the entire covariance matrices.* We will explain this key fact in the next few pages. The third key observation is that the accuracy of this cross-covariance matrices is only as good as the accuracy of the forward model.

First we rewrite the general Kalman filter 11 as shown below. For pedagogical reasons, we let $Q_k = Q$ and $H_k = H$ (though the proposed fast algorithm will proceed on similar lines even if $Q$ and $H$ vary at each time step). Essentially, we write 12 by merging the

**Algorithm 12** Kalman filtering algorithm with the predict and update step combined

| | |
|---|---|
| **Cross-covariance** | $T_{t+1} = F_{t+1}P_{t\|t}F_{t+1}^T H^T + QH^T$ |
| **Innovation covariance** | $S_{t+1} = HT_{t+1} + R$ |
| **Kalman gain** | $K_{t+1} = T_{t+1}S_{t+1}^{-1}$ |
| **Update state** | $s_{t+1} = F_{t+1}s_t + K_{t+1}(z_{t+1} - HF_{t+1}s_t)$ |
| **Update covariance** | $P_{t+1\|t+1} = F_{t+1}P_{t\|t}F_{t+1}^T + Q - K_{t+1}HF_{t+1}P_{t\|t}F_{t+1}^T - K_{t+1}HQ$ |

predict and measurement updates steps. From the above recurrences, we see that the key is to keep track of just $P_{t|t}F_{t+1}^T H^T$ and not the entire covariance matrix $P_t$. Keeping these in mind, we propose a fast Kalman filtering algorithm first for the random-walk forecast model and for a more general forecast model.

#### 5.2.4.1 CEKF for Random walk forecast model

In general, the state transition model is an approximation to the true evolution of the process. In the absence of a good model or if the underlying physical system is too complex to model or when the rate at which the data is acquired is at a much faster rate than the rate of evolution of the system, a random walk forecast model is used. This essentially means the only source of information we have to estimate the process are its measurements. A random walk forecast model is of the form

$$s_{t+1} = s_t + w_t \tag{5.3}$$

Let us rewrite 12 using this forward model. Note that in 13 the most expensive steps are

**Algorithm 13** Kalman filtering algorithm for the random walk forecast model

| | |
|---|---|
| **Cross-covariance** | $T_{t+1} = P_{t\|t}H^T + QH^T + \mathcal{O}(\Delta t)$ |
| **Innovation covariance** | $S_{t+1} = HT_{t+1} + R$ |
| **Kalman gain** | $K_{t+1} = T_{t+1}S_{t+1}^{-1}$ |
| **Update state** | $s_{t+1} = s_t + K_{t+1}(z_{t+1} - Hs_t)$ |
| **Update covariance** | $P_{t+1\|t+1} = P_{t\|t} + Q - K_{t+1}HP_{t\|t} - K_{t+1}HQ + \mathcal{O}(\Delta t)$ |

computing the cross-covariance and updating the covariance, since both scale as $\mathcal{O}(nm^2)$. However, from 13 we realize that we only need to keep track of $(PH)_t = P_{t|t}H^T$, we can modify 13 as shown in 14, i.e., it is enough to propagate $(PH)_t$ only. Note that in the

**Algorithm 14** CEKF for the random walk forecast model

| | |
|---|---|
| **Cross-covariance** | $T_{t+1} = (PH)_t + QH^T + \mathcal{O}(\Delta t)$ |
| **Innovation covariance** | $S_{t+1} = HT_{t+1} + R$ |
| **Kalman gain** | $K_{t+1} = T_{t+1}S_{t+1}^{-1}$ |
| **Update state** | $s_{t+1} = s_t + K_{t+1}(z_{t+1} - Hs_t)$ |
| **Update cross-covariance** | $(PH)_{t+1} = T_{t+1} - K_{t+1}HT_{t+1}$ |

CEKF, we use FMM to compute $QH^T$ at a computational expense of $\mathcal{O}(n^2m)$ and update the cross-covariance as opposed to the covariance. The computational cost for each of the step in 14 scales as $\mathcal{O}(n^2m)$ (This can be further reduced if $H$ is sparse).

#### 5.2.4.2 CEKF for a general forecast model

Note that in general the forward operator of the forecast model is of the form

$$F_t = I + \Delta_t + \mathcal{O}((\Delta_t)^2)$$

where $\|\Delta_t\| \ll 1$. This is due to the fact that as the time step goes to zero, i.e., as $\delta t \to 0$, we need $F_t \to I$. As opposed to a random walk forecast model, the more general evolution based on this forecast model is of the form

$$s_{t+1} = s_t + \Delta_t s_t + w_t + \mathcal{O}((\Delta_t)^2) \tag{5.4}$$

Let us rewrite 12 using this forward model. Note that in 15 the most expensive steps are computing the cross-covariance and updating the covariance, since both scale as $\mathcal{O}(nm^2)$.

**Algorithm 15** Kalman filtering algorithm for a general forecast model

| | |
|---|---|
| **Cross-covariance** | $T_{t+1} = F_t P_{t\|t} H^T + P_{t\|t} F_t^T H^T - P_{t\|t} H^T + QH^T + \mathcal{O}((\Delta_t)^2)$ |
| **Innovation covariance** | $S_{t+1} = HT_{t+1} + R$ |
| **Kalman gain** | $K_{t+1} = T_{t+1} S_{t+1}^{-1}$ |
| **Update state** | $s_{t+1} = s_t + K_{t+1}(z_{t+1} - Hs_t)$ |
| **Update covariance** | $P_{t+1\|t+1} = (I - K_{t+1}H)(F_t P_{t\|t} + P_{t\|t} F_t^T - P_{t\|t} + Q) + \mathcal{O}((\Delta_t)^2)$ |

However, from 15 we realize that we only need to keep track of $P_t^H = P_{t|t}H^T$ and $P_t^F = P_{t|t}F_t^T H^T$, we can modify 15 as shown in 16, i.e., it is enough to propagate $P_t^H$ and $P_t^F$. More importantly, note that 15, it is enough to propagate $P_t^F$, which has accuracy $\mathcal{O}(\Delta_t^2)$. Note that in the CEKF, we use FMM to compute $QH^T$ at a computational expense of

**Algorithm 16** CEKF for a general forecast model

| | |
|---|---|
| **Cross-covariance** | $T_{t+1} = F_t P_t^H + P_t^F + QH^T - P_t^H + \mathcal{O}((\Delta_t)^2)$ |
| **Innovation covariance** | $S_{t+1} = HT_{t+1} + R$ |
| **Kalman gain** | $K_{t+1} = T_{t+1} S_{t+1}^{-1}$ |
| **Update state** | $s_{t+1} = s_t + K_{t+1}(z_{t+1} - Hs_t)$ |
| **Update covariance** | $P_{t+1}^H = T_{t+1} - K_{t+1} H T_{t+1} + \mathcal{O}((\Delta_t)^2)$ |
| | $P_{t+1}^F = (I - K_{t+1}H)(F_t P_t^F + P_t^F + QF_t^T H^T - P_t^H) + \mathcal{O}((\Delta_t)^2)$ |

$\mathcal{O}(n^2 m)$ and update the cross-covariance as opposed to the covariance. The computational cost for each of the step in 16 scales as $\mathcal{O}(n^2 m)$ (Again note that, this can be further reduced if $H$ is sparse). The algorithm can be extended to more general forward models where $F_t$ can be obtain to an accuracy of $\mathcal{O}((\Delta_t)^{s+1})$, by writing appropriate recurrences for $PH^T, PF_t^T H^T, P(F_t^T)^2 H^T, \ldots, P(F_t^T)^s H^T$, which are $\mathcal{O}((\Delta_t)^{s+1})$ accurate.

## 5.3 Validating the new filter

The evolution of the random process after non-dimensionalization is given as

$$\frac{\partial \phi}{\partial t} = \nabla^2 \phi + \frac{dW}{dt} \tag{5.5}$$

Rewriting Equation 5.5, we get

$$d\phi = dt \nabla^2 \phi + \theta \sqrt{dt} w_t$$

Table 5.1: Comparison of time taken by the exact KF and CEKF

| | Kalman filter | New filter |
|---|---|---|
| Pre-computation | - | 0.273 sec |
| Computation | nearly 50 hours | nearly 11 hours. |

where $w_t \sim \mathcal{N}(0, Q)$, $\theta \in \Theta(1)$ is a scaling constant, $\mathbb{E}(w_t^2(i)) = 1$ i.e. $w_t = Lv_t$. The non-dimensionalized equation is solved over a unit square with Dirichlet boundary conditions and initial $\phi$ being 0 throughout the domain. The non-dimensionalized relaxation time is 1.

We solve this on a $51 \times 51$ grid. Hence, $dx = dy = 0.02$. Hence, for stability $dt \leq 4 \cdot 10^{-4}$. We take $dt = 10^{-5}$ and run the simulation for $2 \times 10^5$ time steps i.e. for 2 seconds of non-dimensionalized time, essentially till twice the relaxation time. $\theta$ is taken as $\sqrt{10}$. The forward model is run and the head height is collected at all locations.

Now the head height is measured at 20 points in the domain. The head height is taken as the result from the forward model plus a measurement error i.e. $z_{i,j}^{(k)} = \phi_{i,j}^{(k)} + \mathcal{N}(0, 1)$. Hence, $R = I_{20\times 20}$.

Discretizing 5.5 using explicit Euler in time and central difference in space, we get that

$$\phi_{i,j}^{(k+1)} = \phi_{i,j}^{(k)} + \frac{dt}{(dx)^2}\left(\phi_{i+1,j}^{(k)} - 2\phi_{i,j}^{(k)} + \phi_{i-1,j}^{(k)}\right) + \frac{dt}{(dy)^2}\left(\phi_{i,j+1}^{(k)} - 2\phi_{i,j}^{(k)} + \phi_{i,j-1}^{(k)}\right) + \sqrt{10dt}\, w_{i,j}^{(k)} \tag{5.6}$$

We then reconstruct the image using 5.6 as the forward model. We compare the exact Kalman filter and our new filter to construct the image. Figure 5.1 compares the relative error, i.e., $\dfrac{\|\phi_{\text{filter}} - \phi_{\text{actual}}\|}{\|\phi_{\text{actual}}\|}$, obtained using the two filters. The pre-computation for the new filter involves computing $QH^T$ and $QF^TH^T$ since both $F$ and $H$ do not change with time. The total time taken is shown in Table 5.1.

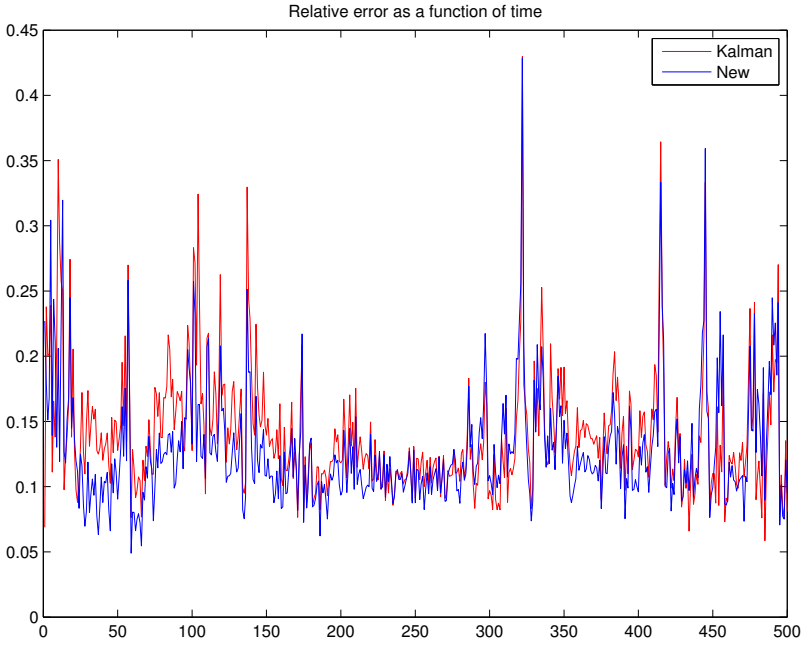



Figure 5.1: Comparison of relative error of the exact and fast Kalman filter

# Chapter 6

# Factorization based fast direct solvers

## 6.1 Background and motivation

Large dense linear systems arising in engineering applications are often solved by iterative techniques. Most iterative solvers based on Krylov subspace methods such as Arnoldi iteration [4], Conjugate Gradient [64], GMRES [106], MINRES [96], Biconjugate Gradient Stabilized Method [111], QMR [42], TFQMR [41], and others, rely on matrix-vector products. The number of iterations required to achieve a target accuracy is very problem dependent. In many instances, the large condition number of the matrix or distribution of the eigenvalues of the matrix in the complex plane (e.g., widely spread eigenvalues) result in a large number of iterations. Consequently, preconditioners need to be devised to improve the numerical properties of the matrix and accelerate convergence of the iterative solver. Once such a method is in place, the cost of performing the matrix-vector products can be reduced using fast summation techniques like the fast multipole method (FMM) [8, 27, 92], the Barnes-Hut algorithm [5], panel clustering [58], FFT, wavelet based methods, and others. Of these different fast summation techniques, the FMM has often been used in the context of linear systems arising out of boundary integral equations. This is because many kernel functions resulting from such integral equations are amenable to the FMM. The literature on the FMM is vast and we refer the readers to some seminal publications and our previous work [7, 25, 32, 33, 39, 51, 52, 120]. These fast iterative solvers accelerate the solution procedure and are able to solve (with some prescribed error tolerance $\varepsilon$) a system in linear

or almost linear time.

In this chapter however we will focus on direct solvers, and before proceeding further, we would like to compare some of the features of direct solvers over their iterative counterparts:

- One of the most important advantages of a direct solver is that it scales well with multiple right hand sides. Any direct solver involves two phases: the factorization phase and the solving phase. The factorization phase is independent of the right hand side and is computationally more expensive than the solving phase. Hence, the key ingredient in *fast direct solvers* is to accelerate the factorization phase. Once an efficient factorization is obtained, all right hand sides can be solved with relatively low computational cost. Iterative solvers, on the other hand, do not modify the matrix and rely solely on matrix vector products and other basic algebra operations. Hence, in most cases for an iterative solver, the entire procedure needs to be restarted for each right-hand side (although this process can be optimized).

- For iterative solvers to be efficient, choosing a good preconditioner [9, 24, 56, 113] is imperative, but in some cases finding a good preconditioner is a difficult task. Many linear systems do not have known good preconditioners, leading to expensive linear solves.

To overcome the disadvantages of iterative solvers and to take advantage of the desirable features of direct solvers, there has been an increasing focus on *fast direct solvers* over the last decade. Such solvers are concerned with dense matrices that have sub-blocks that can be well-approximated by low-rank matrices. Of such class of matrices, the matrices termed hierarchical matrices, denoted as $\mathcal{H}$-matrices and which were introduced by Hackbusch et al. [15, 49, 57, 59, 60] in the late 1990's, are of particular interest. The basic idea is to sub-divide a given matrix into a hierarchy of rectangular blocks and approximate them by low-rank matrices. Due to this special structure, these dense systems are often described as *data sparse*. This is because these matrices can be reconstructed approximately by considering only a subset of their entries.

Matrices that are well approximated by this procedure occur especially in the context of boundary integral equations, interpolation, inverse problems, and others. This class of matrices is very broad and includes for example FMM matrices. A particular class of $\mathcal{H}$-matrices is the class of hierarchically semi-separable matrices (HSS). These matrices were introduced by Chandrasekaran et al. [22, 23, 119] as a generalization of semi-separable

matrices. A semi-separable matrix [112] of separability rank $p$ is a matrix that can be written as a sum of the lower triangular part of a rank-$p$ matrix and the upper triangular part of another rank-$p$ matrix. We refer the readers to [22, 23, 119] for more details regarding the HSS representation.

In the last few years, Greengard [53], Rokhlin [79], Martinsson [87, 88], Lexing Ying [108, 109] et al. have proposed various fast direct solvers making use of the underlying hierarchical low-rank structure. The common theme behind all these algorithms is to construct a low-rank approximation for certain dense sub-blocks and perform a fast update to the solution in a recursive manner.

In this context, the present work discusses an $\mathcal{O}(N \log^2 N)$ solver for *hierarchical off-diagonal low-rank systems* (HODLR) and $\mathcal{O}(N \log N)$ solver for the class of *partial hierarchically semi-separable systems* (p-HSS, a subset of HODLR matrices). The p-HSS structure is discussed in detail in section 6.3.3.2. Linear systems arising from the discretization of boundary integral equations of potential theory [79, 88] in two dimensions, interpolation by radial basis function along a curve [8, 56], and others, can be efficiently modeled by p-HSS matrices. Many papers have proposed fast algorithms for matrices that are similar to p-HSS matrices (some having slightly more restrictive assumptions as we will see later), in particular Gillman et al. [45], Martinsson [87], Martinsson and Rokhlin [88], and Kong et al. [79].

One of the advantages of this new method is its relative simplicity, resulting in a low computational cost. We show that the computational cost of this method scales as $\mathcal{O}(p^2 N \log N)$ for partial hierarchically semi-separable matrices, where the rank of interaction between the clusters is $p$. For problems arising out of one-dimensional manifolds, $p$ can be shown to be independent of $N$. In other situations, for example when considering 2D manifolds — which are cases not covered in [45, 79, 87, 88] — the method in its current form still applies but no longer scales as $\mathcal{O}(N \log N)$. For example, the algorithm scales like $O(N^2)$ for matrices arising in 3D boundary element methods. This results from the fact that the rank of interaction between clusters, as discussed in this method, will scale as $O(N^{1/2})$.

## 6.2 Overview of method and relation to previous work

The current work discusses a fast direct solver for a partial hierarchically semi-separable matrix. The partial hierarchically semi-separable representation is discussed in detail in

section 6.3.3.2. As with most of the other fast direct solvers, the current solver relies on a fast low-rank factorization of the off-diagonal blocks of the matrix to get it into p-HSS form. Once we have the p-HSS representation of the matrix, the solver relies on the Sherman-Morrison-Woodbury update to solve the linear system [47, 63, 116]. The total cost to construct the factorization and to solve the linear system is $\mathcal{O}(N \log N)$. If the matrix is exactly p-HSS, the factorization is exact. In most cases however, the p-HSS matrix is only an approximation (with an error controllable by choosing appropriate ranks in the approximation), in which case the solution produced by the fast direct solver is only approximate. To illustrate the performance and accuracy of this algorithm, we solve a linear interpolation problem along a one dimensional manifold using radial basis functions in section 6.5.

We now discuss our algorithm based on factorization in reference to existing ones. We will restrict ourselves to existing methods that also take advantage of the low-rank off-diagonal blocks. The work by Chandrasekaran et al. [23] constructs a $\mathcal{O}(N)$ solver for HSS systems. It constructs a $ULV^H$ decomposition ($U$ and $V$ are unitary matrices, and $L$ is lower triangular, $^H$ is the transpose conjugate operator) of a hierarchically semi-separable matrix. Their approach differs from ours in many ways. The starting point of their algorithm is to recognize that when we have a low-rank approximation of the form $UBV^H$ ($U$ and $V$ are thin matrices with $p$ columns) it is possible to apply a unitary transformation so that only the last $p$ rows of $U$ are non-zero. This result is then used to reduce the size of $A$ recursively ("bottom-up" approach) until we are left with a small enough linear system that can be solved by a conventional method. They describe an $\mathcal{O}(N^2)$ algorithm to construct the HSS representation and an $\mathcal{O}(N)$ algorithm when the matrix is associated with a smooth kernel. For the latter, Chebyshev polynomials are used to construct low-rank approximations, which is similar to our approach and also Fong et al. [39]. The solver described in the present chapter constructs a different one-sided factorization of the matrix. Further, our algorithm is applicable for a p-HSS matrix, which is a superset of HSS matrices, as will be explained in detail in section 6.3.3.

The work by Rokhlin and Martinsson [88] constructs an $\mathcal{O}(N)$ fast direct solver for boundary integral equations in two-dimensions making use of off-diagonal low-rank blocks. The algorithm constructs the inverse using a compressed block factorization that takes advantage of the low-rank off-diagonal blocks to factor the matrix. A two-sided hierarchical factorization of the inverse is constructed. The approach follows some of the ideas in [23], and is based on applying transformations to low-rank matrices such that only $p$ non-zero rows

(resp. columns) remain while other rows (resp. columns) are set to 0. This allows compressing the matrix and progressively reducing its size. The solver proposed in the present work, on the other hand, is conceptually simple to understand and easier to implement. Further, this paper presents a factorization of the matrix (followed by a solve) whereas [88] builds a compressed factorization of the inverse matrix.

A recently published work, while we were working on this, by Kong et al. [79] proposes an $\mathcal{O}(N \log^2 N)$ solver for boundary integral equations in two-dimensions taking advantage of off-diagonal low-rank blocks. Though their algorithm is similar to parts of our algorithm, e.g., their algorithm also uses Sherman-Morrison-Woodbury updates, we proceed further and reduce the computational complexity of the algorithm to $\mathcal{O}(N \log N)$ with the additional assumption of p-HSS structure. Our approach highlights the bottleneck in the $\mathcal{O}(N \log^2 N)$ algorithm, enabling us to reduce the computational complexity to $\mathcal{O}(N \log N)$. In fact, our $\mathcal{O}(N \log N)$ algorithm can be viewed as an extension of the algorithm proposed by Kong et al. [79] by making the additional assumption of p-HSS structure and thereby reducing the cost from $\mathcal{O}(N \log^2 N)$ to $\mathcal{O}(N \log N)$. In our benchmarks, the $\mathcal{O}(N \log N)$ resulted in a speed-up of nearly 4 compared to the $\mathcal{O}(N \log^2 N)$, even for moderately large $N$. The speed-up in general depends on $N$ and will improve even more for larger $N$.

The strategy is presented in such a way that it clearly indicates how the structure of the matrix dictates the cost, i.e., the algorithm explicitly reveals how the assumption of the p-HSS structure cuts down the cost from $\mathcal{O}(N \log^2 N)$ to $\mathcal{O}(N \log N)$. In a similar spirit, this algorithm can be extended to an HSS structure to yield an $\mathcal{O}(N)$ algorithm, which will be the subject of future work.

The structure of tis chapter is as follows. The next section discusses some of the key ideas including the different hierarchical representations, low-rank approximations and Sherman-Morrison-Woodbury formula that are the primary ingredients for the algorithm. Section 6.4 motivates and provides an overview of the algorithm, and discusses the computational complexity of the algorithm. Section 6.5 discusses the applicability, performance and accuracy of this solver by providing numerical benchmarks for an interpolation problem on an one-dimensional manifold using radial basis functions. The final section concludes the chapter by highlighting the capabilities of this solver.

## 6.3 Preliminary ideas

In this section, we discuss the key ideas that will be of help in understanding the algorithm.

### 6.3.1 Sherman-Morrison-Woodbury formula

The key ingredient in our algorithm is the Sherman-Morrison-Woodbury formula [63, 116], which provides a convenient way to update the solution of a linear system perturbed by a low-rank update.

Consider solving a system of the form

$$\left(I + UV^T\right) x = b \tag{6.1}$$

where $I \in \mathbb{R}^{N\times N}$ is the identity matrix, $U, V \in \mathbb{R}^{N\times p}$, $x, b \in \mathbb{R}^{N\times r}$, and $p \leq r \ll N$.
The Sherman-Morrison-Woodbury formula gives us

$$x = b - U\left(I + V^T U\right)^{-1} V^T b \tag{6.2}$$

The computational cost for solving (6.1) using (6.2) is $\mathcal{O}(prN)$. We refer the readers to [63, 116] for the derivation of the above result and the computational cost, though the direct proof takes only a few steps:

$$\begin{aligned}\left(I + UV^T\right) x &= \left(I + UV^T\right) b - \left(I + UV^T\right) U\left(I + V^T U\right)^{-1} V^T b \quad \text{[Eq. (6.2)]} \\ &= \left(I + UV^T\right) b - U\left(I + V^T U\right)\left(I + V^T U\right)^{-1} V^T b \quad \text{[Refactor the second term]} \\ &= \left(I + UV^T\right) b - UV^T b = b\end{aligned}$$

### 6.3.2 Fast low-rank factorization

Given a matrix $A \in \mathbb{R}^{M\times N}$, the *optimal* rank $p$ approximation can be obtained from its singular value decomposition [47]. However, singular value decomposition is computationally expensive with a cost of $\mathcal{O}(MN\min(M, N))$. In recent years, there is an increasing focus [26, 43, 54, 90] on constructing fast approximate low-rank factorizations for matrices. Techniques like adaptive cross approximation [102] (ACA), pseudo-skeletal approximations, [48] interpolatory decomposition, [39] randomized algorithms [43, 81, 117], rank-revealing LU [90, 97] and QR [54] algorithms provide great ways for constructing

efficient approximate low-rank representations. The proposed algorithm for solving the linear system is independent of the algorithm to construct the low-rank factorizations and therefore can be combined with any low-rank factorization technique. All these different techniques have already been discussed in detail in chapter 2.

In the numerical illustrations discussed in section 6.5, we consider linear systems arising from interpolation schemes based on radial basis functions such as quadrics ($r^2 + a^2$), inverse multi-quadric ($1/\sqrt{r^2 + a^2}$), Gaussian ($\exp(-r^2/a^2)$), exponential ($\exp(-r/a)$), and others. These radial basis functions are smooth and non-singular. For smooth kernels, interpolation using Chebyshev polynomials is an attractive method to construct low-rank factorizations [39, 89]. Although any interpolation scheme can be used to construct a low-rank factorization, in the present work, the Chebyshev polynomials will serve as the interpolation basis along with their roots as the interpolation nodes.

### 6.3.3 Hierarchical representations

We briefly look at the relevant hierarchical representations, which we will be dealing with in this chapter. For more details, kindly refer to chapter 3.

#### 6.3.3.1 Hierarchical off-diagonal low-rank matrix

A matrix, $K \in \mathbb{R}^{N\times N}$, is termed a 2-level hierarchical off-diagonal low-rank matrix, denoted as HODLR, if it can be written in the form shown in equation (6.3).

$$K = \begin{bmatrix} K_1^{(1)} & U_1^{(1)}V_{1,2}^{(1)^T} \\ U_2^{(1)}V_{2,1}^{(1)^T} & K_2^{(1)} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} K_1^{(2)} & U_1^{(2)}V_{1,2}^{(2)^T} \\ U_2^{(2)}V_{2,1}^{(2)^T} & K_2^{(2)} \end{bmatrix} & U_1^{(1)}V_{1,2}^{(1)^T} \\ U_2^{(1)}V_{2,1}^{(1)^T} & \begin{bmatrix} K_3^{(2)} & U_3^{(2)}V_{3,4}^{(2)^T} \\ U_4^{(2)}V_{4,3}^{(2)^T} & K_4^{(2)} \end{bmatrix} \end{bmatrix} \tag{6.3}$$

where $K_i^{(2)} \in \mathbb{R}^{N/4\times N/4}$, $U_{2i-1}^{(k)}, U_{2i}^{(k)}, V_{2i-1,2i}^{(k)}, V_{2i,2i-1}^{(k)} \in \mathbb{R}^{N/2^k\times p}$ and $p \ll N$.

In general, a $\kappa$-level HODLR matrix is the one in which, the $i^{th}$ diagonal block at level $k$, where $1 \leq i \leq 2^k$ and $0 \leq k < \kappa$, denoted as $K_i^{(k)}$, can be written as

$$K_i^{(k)} = \begin{bmatrix} K_{2i-1}^{(k+1)} & U_{2i-1}^{(k+1)}V_{2i-1,2i}^{(k+1)^T} \\ U_{2i}^{(k+1)}V_{2i,2i-1}^{(k+1)^T} & K_{2i}^{(k+1)} \end{bmatrix} \tag{6.4}$$

where $K_i^{(k)} \in \mathbb{R}^{N/2^k \times N/2^k}$, $U_{2i-1}^{(k)}, U_{2i}^{(k)}, V_{2i-1,2i}^{(k)}, V_{2i,2i-1}^{(k)} \in \mathbb{R}^{N/2^k \times p}$ and $p \ll N$. The maximum number of levels, $\kappa$, is $\lfloor \log_2(N/2p) \rfloor$. The construction of a $\kappa$-level HODLR matrix, using interpolation to obtain low-rank of the off-diagonal blocks, is described below.

1. Let the root level (level 0) contain the location of all the points in the domain.
2. For all the clusters at level $\kappa$, compute the interaction of each cluster with itself, i.e., $K_i^{(\kappa)}$ for all $i \in \{1, 2, \ldots, 2^\kappa\}$.
3. At all levels, $k \in \{1, 2, \ldots, \kappa\}$, for all the clusters, $i \in \{1, 2, \ldots, 2^k\}$ and $j \in \{1, 2, \ldots, 2^{k-1}\}$, compute the interaction with its sibling, using a low-rank representation, i.e.,
   - Compute the interpolation matrices, $U_i^{(k)}$, using $p$ Chebyshev nodes for the cluster $i$ at level $k$
   - Compute the interaction of the Chebyshev nodes of the cluster with its sibling cluster, i.e., $V_{2j-1,2j}^{(k)}$ and $V_{2j,2j-1}^{(k)}$. Note: $V_{a,b}^{(k)}$ captures the interaction of the Chebyshev nodes of the cluster $a$ with its sibling cluster $b$ at level $k$.

This gives the desired HODLR representation. The total cost to construct and store a HODLR matrix is $\mathcal{O}(pN(1+\kappa))$.

#### 6.3.3.2 Partial hierarchically semi-separable matrix

The partial hierarchically semi-separable matrices, denoted as p-HSS, are a subset of the HODLR matrices and a superset of hierarchically semi-separable (HSS) matrices. Some of the notations we use are similar to those used by Chandrasekaran et al. [22, 23] The proposed $\mathcal{O}(N \log N)$ algorithm is applicable for this class of matrices. The recursive hierarchical structure of the p-HSS representation is seen when we consider the $4 \times 4$ block partitioning of a p-HSS matrix, $K$. The two-level p-HSS representation is as shown in equation (6.5).

$$K = \begin{bmatrix} K_1^{(1)} & U_1^{(1)} V_{1,2}^{(1)^T} \\ U_2^{(1)} V_{2,1}^{(1)^T} & K_2^{(1)} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} K_1^{(2)} & U_1^{(2)} V_{1,2}^{(2)^T} \\ U_2^{(2)} V_{2,1}^{(2)^T} & K_2^{(2)} \end{bmatrix} & \begin{bmatrix} U_1^{(2)} S_1^{(2)} \\ U_2^{(2)} S_2^{(2)} \end{bmatrix} V_{1,2}^{(1)^T} \\ \begin{bmatrix} U_3^{(2)} S_3^{(2)} \\ U_4^{(2)} S_4^{(2)} \end{bmatrix} V_{2,1}^{(1)^T} & \begin{bmatrix} K_3^{(2)} & U_3^{(2)} V_{3,4}^{(2)^T} \\ U_4^{(2)} V_{4,3}^{(2)^T} & K_4^{(2)} \end{bmatrix} \end{bmatrix} \tag{6.5}$$

where $U_{2i-1}^{(k)}, U_{2i}^{(k)}, V_{2i-1,2i}^{(k)}, V_{2i,2i-1}^{(k)} \in \mathbb{R}^{N/2^k \times p}$, $S_i^{(2)} \in \mathbb{R}^{p \times p}$ and $p \ll N$.

The key feature is that $U_1^{(1)}$ is defined in terms of $U_1^{(2)}$ and $U_2^{(2)}$; similarly, $U_2^{(1)}$ is defined in terms of $U_3^{(2)}$ and $U_4^{(2)}$, i.e., we have

$$U_1^{(1)} = \begin{bmatrix} U_1^{(2)} S_1^{(2)} \\ U_2^{(2)} S_2^{(2)} \end{bmatrix}, \; U_2^{(1)} = \begin{bmatrix} U_3^{(2)} S_3^{(2)} \\ U_4^{(2)} S_4^{(2)} \end{bmatrix}. \tag{6.6}$$

An equivalent statement is that $U_1^{(1)}$ lies in the span of

$$\begin{bmatrix} U_1^{(2)} & 0 \\ 0 & U_2^{(2)} \end{bmatrix}$$

In a $\kappa$-level p-HSS representation, if we denote the $i^{th}$ diagonal block at level $k$ as $K_i^{(k)}$ (6.4), then $U_i^{(k)}$ is constructed from $U_{2i-1}^{(k+1)}$ and $U_{2i}^{(k+1)}$, i.e., we have

$$U_i^{(k)} = \begin{bmatrix} U_{2i-1}^{(k+1)} S_{2i-1}^{(k+1)} \\ U_{2i}^{(k+1)} S_{2i}^{(k+1)} \end{bmatrix} \tag{6.7}$$

The maximum number of levels, $\kappa$, is $\lfloor \log_2(N/2p) \rfloor$. The construction of a $\kappa$-level p-HSS matrix, using interpolation is discussed below.

1. Let the root level (level 0) contain the location of all the points in the domain.

2. For all the clusters at level $\kappa$, compute the interaction of each cluster with itself, i.e., $K_i^{(\kappa)}$ for all $i \in \{1, 2, \ldots, 2^\kappa\}$.

3. For all the clusters at level $\kappa$, compute the interpolation matrices using $p$ Chebyshev nodes, i.e., $U_i^{(\kappa)}$ for all $i \in \{1, 2, \ldots, 2^\kappa\}$.

4. For all clusters at level $k$, compute the interaction of the Chebyshev nodes of the cluster with the sibling of the cluster using $p$ Chebyshev nodes, i.e., $V_{2i-1,2i}^{(k)}, V_{2i,2i-1}^{(k)}$ where $k \in \{1, 2, \ldots, \kappa\}$ and $i \in \{1, 2, \ldots, 2^{k-1}\}$.

5. For all clusters at level $k$, compute the interpolation matrices from the cluster to its parent using $p$ Chebyshev nodes, i.e., $S_i^{(k)}$ where $k \in \{2, 3, \ldots, \kappa\}$ and $i \in \{1, 2, \ldots, 2^k\}$. This is done by computing $S_i^{(k)}(r, s) = U(\bar{x}_r^{(k)}, \bar{x}_s^{(k-1)})$ where $\bar{x}_r^{(k)}$ is the $r^{th}$ Chebyshev

node of cluster $i$ at level $k$ and $\bar{x}_s^{(k-1)}$ is the $s^{th}$ Chebyshev node at level $(k-1)$ of the parent of the $i^{th}$ cluster.

This gives the desired p-HSS representation. The total cost to construct and store a p-HSS matrix is $\mathcal{O}\left(pN(1+\kappa)\right)$.

We refer the readers to [22, 23, 119] for detailed description of hierarchically semi-separable (HSS) matrices. These representations correspond to increasingly larger set of matrices, i.e.,

$$\boxed{\text{HSS} \subset \text{p-HSS} \subset \text{HODLR}}$$

The $\mathcal{O}(N \log^2 N)$ algorithm is applicable for HODLR matrices while the $\mathcal{O}(N \log N)$ algorithm is applicable for p-HSS matrices. The strategy is presented in a way that the $\mathcal{O}(N \log N)$ algorithm is an extension of the $\mathcal{O}(N \log^2 N)$ algorithm with the additional assumption of a p-HSS structure.

## 6.4 Algorithm

In this section, we present the $\mathcal{O}(N \log^2 N)$ algorithm for HODLR matrices, and $\mathcal{O}(N \log N)$ algorithm for p-HSS matrices. As discussed in the introduction, any direct solver involves two main steps. The first step is the factorization step and the second step is where we use the factorization to obtain the final solution. The difference in the computational cost between the $\mathcal{O}(N \log^2 N)$ and $\mathcal{O}(N \log N)$ algorithm is in the factorization step. Once the desired factorization has been obtained, the computational cost of applying the factorization to solve the system is $\mathcal{O}(N \log N)$ irrespective of the factorization algorithm. For purposes of illustration and analysis, we assume that the off-diagonal sub-blocks at each level are of the same rank and the system is split into two equal halves at each level.

### 6.4.1 Factorization phase

In this section, we discuss the factorization of the HODLR and p-HSS matrices. The overall idea is to factor the underlying matrix, $K \in \mathbb{R}^{N \times N}$, into $\kappa + 1$ block diagonal matrices as in equation (6.8):

$$K = K_\kappa K_{\kappa-1} K_{\kappa-2} \cdots K_1 K_0 \tag{6.8}$$

where $K_k \in \mathbb{R}^{N \times N}$ is a block diagonal matrix with $2^k$ diagonal blocks each of size $\frac{N}{2^k} \times \frac{N}{2^k}$ and each of the diagonal blocks at all levels are low-rank update to the identity matrix.

A $\kappa$-level HODLR matrix, $K^{(\kappa)} \in \mathbb{R}^{N\times N}$ is presented in Equation (6.9).

$$K^{(\kappa)} = \begin{bmatrix} \begin{bmatrix} K_1^{(\kappa)} & U_1^{(\kappa)}V_{1,2}^{(\kappa)^T} \\ U_2^{(\kappa)}V_{2,1}^{(\kappa)^T} & K_2^{(\kappa)} \end{bmatrix} & U_1^{(\kappa-1)}V_{1,2}^{(\kappa-1)^T} & \cdots & \cdots \\ U_2^{(\kappa-1)}V_{2,1}^{(\kappa-1)^T} & \begin{bmatrix} K_3^{(\kappa)} & U_3^{(\kappa)}V_{3,4}^{(\kappa)^T} \\ U_4^{(\kappa)}V_{4,3}^{(\kappa)^T} & K_4^{(\kappa)} \end{bmatrix} & \cdots & \cdots \\ \vdots & \vdots & \ddots & \cdots \\ \vdots & \vdots & \vdots & \begin{bmatrix} K_{2^\kappa}^{(\kappa)} & U_{2^\kappa-1}^{(\kappa)}V_{2^\kappa-1,2^\kappa}^{(\kappa)^T} \\ U_{2^\kappa}^{(\kappa)}V_{2^\kappa,2^\kappa-1}^{(\kappa)^T} & K_{2^\kappa}^{(\kappa)} \end{bmatrix} \end{bmatrix} \tag{6.9}$$

where $K_i^{(\kappa)} \in \mathbb{R}^{N/2^\kappa \times N/2^\kappa}$, $U_j^{(k)}$, $V_{2i-1,2i}^{(k)}, V_{2i,2i-1}^{(k)} \in \mathbb{R}^{N/2^k\times p}$ for $k \in \{1,2,\ldots,\kappa\}$, $j \in \{1,2,\ldots,2^k\}$ and $i \in \{1,2,\ldots,2^{k-1}\}$. Recall that a $\kappa$-level p-HSS matrix not only has the structure described in Equation (6.9) but also has the additional structure mentioned in Equation (6.7).

The first step in the algorithm is to factor the block diagonal matrix shown in equation (6.10).

$$K_\kappa = \begin{bmatrix} K_1^{(\kappa)} & 0 & 0 & 0 & \cdots & \cdots \\ 0 & K_2^{(\kappa)} & 0 & 0 & \cdots & \cdots \\ 0 & 0 & K_3^{(\kappa)} & 0 & \cdots & \cdots \\ 0 & 0 & 0 & K_4^{(\kappa)} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & K_{2^\kappa}^{(\kappa)} \end{bmatrix} \tag{6.10}$$

**Here is the important difference between the factorization of a HODLR and a p-HSS matrix.** For the HODLR structure, when we factor $K_\kappa$, $U_j^{(k)}$ at all levels $k$, need to be updated. However, if we have the p-HSS structure, we only need to update $U_j^{(\kappa)}$, since the p-HSS representation allows us to *obtain* $U_j^{(k)}$ *at all the other levels through the recurrence* (6.7).

Another key observation, valid for both the HODLR and p-HSS matrices and that enables us to reduce the computational cost, is that we *do not need to update* $V_{2i-1,2i}^{(k)}$ *and* $V_{2i,2i-1}^{(k)}$ *for any level.*

Now to factor out $K_\kappa$, we need to multiply by the inverse of $K_i^{(\kappa)}$ the corresponding $N/2^\kappa$ rows of $U_i^{(k)}$ for all $k \in \{1, 2, \ldots, \kappa\}$ (a subset of the $N/2^k$ rows of $U_i^{(k)}$). The multiplication by the inverse is carried out in practice by solving the appropriate linear system.

---

**HODLR case.** Since each $U_i^{(k)}$ has $p$ columns, we need to apply the inverse to a total of $\kappa p$ columns. The computational cost of applying the inverse of $K_i^{(\kappa)}$ to $r$ columns is $\mathcal{O}(p^3 + rp^2)$ and hence to factor $K_\kappa$, the total cost is $\mathcal{O}(\kappa p^2 N)$, where $\kappa = \lfloor \log_2(N/2p) \rfloor$.

**p-HSS case.** It is sufficient to apply the inverse to only the $p$ columns of $U_i^{(\kappa)}$, for the reasons explained above. Hence, the total cost becomes $\mathcal{O}(p^2 N)$.

---

Factoring out $K_\kappa$, we get that $K^{(\kappa)} = K_\kappa \; K^{(\kappa-1)}$, where $K^{(\kappa-1)}$ is of the form in equation (6.11).

$$K^{(\kappa-1)} = \begin{bmatrix} \begin{bmatrix} I & U_1^{(\kappa,1)} V_{1,2}^{(\kappa)^T} \\ U_2^{(\kappa,1)} V_{2,1}^{(\kappa)^T} & I \end{bmatrix} & U_1^{(\kappa-1,1)} V_{1,2}^{(\kappa-1)^T} & \cdots & \cdots \\ U_2^{(\kappa-1,1)} V_{2,1}^{(\kappa-1)^T} & \begin{bmatrix} I & U_3^{(\kappa,1)} V_{3,4}^{(\kappa)^T} \\ U_4^{(\kappa,1)} V_{4,3}^{(\kappa)^T} & I \end{bmatrix} & \cdots & \cdots \\ \vdots & \vdots & \ddots & \cdots \\ \vdots & \vdots & \vdots & \begin{bmatrix} I & U_{2^\kappa-1}^{(\kappa,1)} V_{2^\kappa-1,2^\kappa}^{(\kappa)^T} \\ U_{2^\kappa}^{(\kappa,1)} V_{2^\kappa,2^\kappa-1}^{(\kappa)^T} & I \end{bmatrix} \end{bmatrix} \tag{6.11}$$

Note that $U_i^{(k,1)}$ indicates that $U_i^{(k)}$ has been updated after factoring out $K_\kappa$. Now that we have $K^{(\kappa-1)}$, the plan is to repeat this process as we go up the levels. For ease of understanding, let's define

$$\begin{bmatrix} I & U_{2i-1}^{(\kappa,1)} V_{2i-1,2i}^{(\kappa)^T} \\ U_{2i}^{(\kappa,1)} V_{2i,2i-1}^{(\kappa)^T} & I \end{bmatrix} = K_i^{(\kappa-1,1)}, \tag{6.12}$$

where $I \in \mathbb{R}^{N/2^\kappa \times N/2^\kappa}$ is the identity matrix and $K_i^{(\kappa-1,1)} \in \mathbb{R}^{N/2^{\kappa-1} \times N/2^{\kappa-1}}$. Hence, we

have

$$K^{(\kappa-1)} = \begin{bmatrix} K_1^{(\kappa-1,1)} & U_1^{(\kappa-1,1)} V_{1,2}^{(\kappa-1)^T} & \cdots & \cdots \\ U_2^{(\kappa-1,1)} V_{2,1}^{(\kappa-1)^T} & K_2^{(\kappa-1,1)} & \cdots & \cdots \\ \vdots & \vdots & \ddots & \cdots \\ \vdots & \vdots & \vdots & K_{2^{\kappa-1}}^{(\kappa-1,1)} \end{bmatrix} \tag{6.13}$$

Note that $K^{(\kappa-1)}$ is a $(\kappa-1)$-level HODLR matrix. In addition, if $K^{(\kappa)}$ had a p-HSS structure to begin with, then $K^{(\kappa-1)}$ will also have a p-HSS structure, since $U_i^{(k,1)}$ is related to $U_{2i-1}^{(k+1,1)}$ and $U_{2i-1}^{(k+1,1)}$ through (6.14):

$$U_i^{(k,1)} = \begin{bmatrix} U_{2i-1}^{(k+1,1)} S_{2i-1}^{(k+1)} \\ U_{2i}^{(k+1,1)} S_{2i}^{(k+1)} \end{bmatrix} \tag{6.14}$$

Hence, let us factor $K^{(\kappa-1)}$ as $K_{\kappa-1}\, K^{(\kappa-2)}$, where $K_{\kappa-1}$ is a block diagonal matrix with $2^{\kappa-1}$ blocks each of size $N/2^{\kappa-1} \times N/2^{\kappa-1}$ as shown in equation (6.15), and $K^{(\kappa-2)}$ is a $(\kappa-2)$-level HODLR matrix.

$$K_{\kappa-1} = \begin{bmatrix} K_1^{(\kappa-1,1)} & 0 & 0 & \cdots \\ 0 & K_2^{(\kappa-1,1)} & \cdots & \cdots \\ 0 & \vdots & \ddots & \cdots \\ \vdots & \vdots & \vdots & K_{2^{\kappa-1}}^{(\kappa-1,1)} \end{bmatrix} \tag{6.15}$$

Though the diagonal blocks of $K^{(\kappa-1)}$ are now twice in size compared to the diagonal blocks of $K^{(\kappa)}$, all the diagonal blocks are a low-rank update to an identity matrix, i.e., $K_i^{(\kappa-1,1)} \in \mathbb{R}^{N/2^{\kappa-1} \times N/2^{\kappa-1}}$ in equation (6.12) can be written as $I + \tilde{U}_i^{(\kappa)} \tilde{V}_i^{(\kappa)^T}$, where

$$\tilde{U}_i^{(\kappa)} = \begin{bmatrix} U_{2i-1}^{(\kappa,1)} & 0 \\ 0 & U_{2i}^{(\kappa,1)} \end{bmatrix} \in \mathbb{R}^{(N/2^{\kappa-1}) \times 2p}$$

and

$$\tilde{V}_i^{(\kappa)^T} = \begin{bmatrix} 0 & V_{2i-1,2i}^{(\kappa)^T} \\ V_{2i,2i-1}^{(\kappa)^T} & 0 \end{bmatrix} \in \mathbb{R}^{(N/2^{\kappa-1}) \times 2p}$$

Let us now calculate the computational complexity for the second step in the factorization,

i.e., to obtain $K^{(\kappa-1)} = K_{\kappa-1}\, K^{(\kappa-2)}$. To perform this, we first need to multiply by the inverse of $K_i^{(\kappa-1,1)}$ the corresponding rows of $U_i^{(k,1)}$ for all $k \in \{1, 2, \ldots, \kappa-1\}$.

---

**HODLR case.** Since $K_i^{(\kappa-1,1)} = I + \tilde{U}_i^{(\kappa)} \tilde{V}_i^{(\kappa)^T}$, by Sherman-Morrison-Woodbury formula, the cost to apply the inverse of $K_i^{(\kappa-1,1)}$ to $r$ columns is $\mathcal{O}(N/2^{\kappa-1} \times (2p) \times r)$. Hence, the total cost of this step is $\mathcal{O}((\kappa-1)p^2N)$.

**p-HSS case.** It is sufficient to apply the inverse to just $U_i^{(\kappa-1,1)}$. Hence, the total cost is $\mathcal{O}(p^2N)$.

---

This is repeated till we reach level 0 to get a factorization of the form

$$K = K_\kappa K_{\kappa-1} \cdots K_0 \tag{6.16}$$

where $K_k$ is a block diagonal matrix with $2^k$ diagonal blocks each of size $N/2^k \times N/2^k$. Note that for all $k$, except $\kappa$, all the block diagonal matrices in $K_k$ can be written as a rank $2p$ update to a $\left(N/2^k \times N/2^k\right)$ identity matrix. Hence, the computational complexity at level $k$ for factorizing $K^{(k)}$ as $K_k\, K^{(k-1)}$, for a HODLR matrix is $\mathcal{O}(p^2Nk)$ and for a p-HSS matrix is $\mathcal{O}(p^2N)$. It is important to note that the computational cost for the factorization at level $k$ depends on $k$ for a HODLR matrix, whereas it is independent of $k$ for a p-HSS matrix. Hence, the computational complexity for the factorization is: Figure 6.1 provides a

| Matrix | Computational complexity |
|---|---|
| HODLR | $\sum_{k=1}^{\kappa} \mathcal{O}(kp^2N) = \mathcal{O}(\kappa^2p^2N) = \mathcal{O}(p^2N \log^2 N)$ |
| p-HSS matrix | $\sum_{k=1}^{\kappa} \mathcal{O}(p^2N) = \mathcal{O}(\kappa p^2N) = \mathcal{O}(p^2N \log N)$ |

pictorial description of this factorization.

### 6.4.2 Solving phase

The solving phase is independent of the factorization phase and is the same for both the algorithms. Once we have the factorization of the matrix $K$ in the form $K_\kappa K_{\kappa-1} K_{\kappa-2} \ldots K_0$, then the solution to $Kx = b$ where $K \in \mathbb{R}^{N \times N}$, $b \in \mathbb{R}^{N \times r}$ is given by $x = K_0^{-1} K_1^{-1} \ldots K_\kappa^{-1} b$.

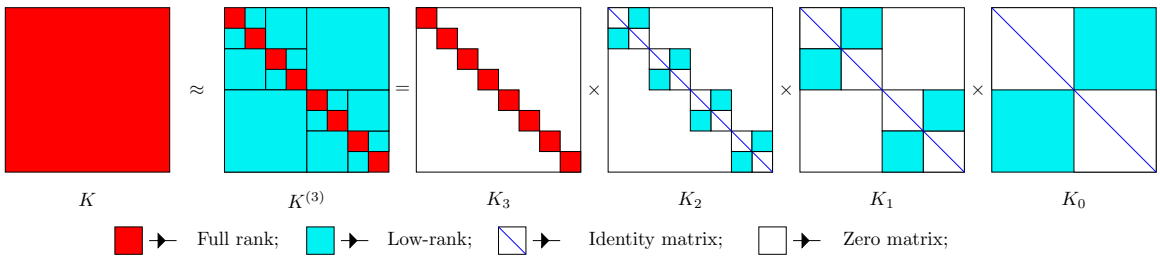



Figure 6.1: Factorization of a three level HODLR/p-HSS matrix

Hence, we need to first apply the inverse of $K_\kappa$ followed by $K_{\kappa-1}$ all the way up to $K_0$. All these inverses need to be applied to an $N \times r$ matrix, where $r$ is the number of right hand sides. In our implementation, the solving phase and factorization phase proceed simultaneously. Hence, in the benchmarks presented the factorization is not constructed explicitly. For instance, in the first step of the factorization phase of both algorithms, when we apply the inverse of $K_\kappa$ to $K^{(\kappa)}$ to get $K^{(\kappa-1)}$, we also apply the inverse to the $r$ right hand sides. Similarly, at every step $k$, when we apply the inverse of $K_k$ to $K^{(k)}$ to get $K^{(k-1)}$, the inverse is also applied to the corresponding $r$ right hand sides. This is analogous to Gaussian elimination where the LU factorization and back substitution proceed together. The additional cost of applying these inverses at each step to the $r$ right hand sides is $\mathcal{O}(rpN)$. Hence, the total cost of the solve phase is $\mathcal{O}(rpN\log(N))$.

The computational complexity for the algorithms discussed are summarized in Table 6.1.

Table 6.1: Computational complexity; $p$: rank; $r$: number of right-hand sides; $N$: size of matrix

| Phase | HODLR | p-HSS |
|---|---|---|
| Factorization | $\mathcal{O}(p^2 N \log^2 N)$ | $\mathcal{O}(p^2 N \log N)$ |
| Solving | $\mathcal{O}(prN \log N)$ | $\mathcal{O}(prN \log N)$ |

## 6.5 Numerical benchmark

In this section, we present results obtained for our test case. The test case considered is a contour deformation problem using radial basis functions.

### 6.5.1 Interpolation using radial basis functions

We briefly discuss interpolation using radial basis functions. The literature on interpolation using radial basis function is vast and we refer the reader to a few [12, 16, 36, 107, 114, 118]. As with other interpolation techniques, the motivation behind interpolation based on radial basis functions is to approximate the given data by a function defined on a large set, ensuring that the function passes through the data points. Let $\{f_k\}_{k=1}^N$ be the given values of the data observed at $N$ distinct points say $\{x_k\}_{k=1}^N$. We consider the case when $x_k$'s lie on a one-dimensional manifold. The motivation behind interpolation using radial basis functions is to find a smooth function $s(x)$ such that $s(x_k) = f_k$, for all $k \in \{1, 2, \ldots, N\}$. To achieve this, the interpolant $s(x)$ is considered to be of the form,

$$s(x) = \sum_{k=1}^{N} \lambda_k \phi(x - x_k) + p(x)$$

where $p(x)$ is a polynomial of degree $l$, $\lambda_k$ are a set of weights and $\phi$ is a function from $\mathbb{R} \to \mathbb{R}$. We set $\phi(0) = 0$. This is termed the nugget effect [3, 6, 34, 37, 95] and helps in making the system reasonably well-conditioned. Without the nugget effect, some of the systems arising out of radial basis interpolation are singular or very close to being singular and hence for most practical applications, a nugget is always chosen.

We have that $p(x) \in \mathcal{P}^l$, where $\mathcal{P}^l$ is the space of polynomials with degree $l$. Let $\{p_0(x), p_1(x), \ldots, p_l(x)\}$ be a basis for $\mathcal{P}^l$. Hence, $p(x)$ can be written as

$$p(x) = \sum_{j=0}^{l} a_j p_j(x).$$

This gives us that the interpolant $s(x)$ must be of the form

$$s(x) = \sum_{k=1}^{N} \lambda_k \phi(x - x_k) + \sum_{j=0}^{l} a_j p_j(x) \tag{6.17}$$

The polynomial $p(x)$ is most often taken to be constant ($l = 0$), linear ($l = 1$) (or) cubic ($l = 3$). Further, in case of interpolation by radial basis functions, $\phi(x - y)$ is a function of

$\|x - y\|_2$. In which case, equation (6.17) can be rewritten as

$$s(x) = \sum_{k=1}^{N} \lambda_k \phi\left(\|x - x_k\|_2\right) + \sum_{j=0}^{l} a_j p_j(x) \tag{6.18}$$

To determine the interpolant, we need to determine the $\lambda_k$'s and the $a_j$'s, a total of $N + l + 1$ unknowns. The interpolant in equation (6.18) is required to satisfy the interpolation conditions.

$$s(x_k) = f_k, \qquad \forall k \in \{1, 2, \ldots, n\} \tag{6.19}$$

Equation (6.19) ensures that the interpolant passes through the data points. The remaining equations are obtained through the *side conditions* given in equation (6.20).

$$\sum_{k=1}^{N} \lambda_k p_j(x_k) = 0, \qquad \forall j \in \{0, 1, \ldots, l\} \tag{6.20}$$

The side conditions in equation (6.20) ensure polynomial reproduction, i.e., if the data arises from a polynomial $q(x) \in \mathcal{P}^l$, then the interpolant is also $q(x)$. In addition, this condition results in the fact that away from the interpolation points $x_k$, the interpolation function will be approximated by $\sum_{j=0}^{l} a_j p_j(x)$.

Hence, we now have $N + l + 1$ equations and $N + l + 1$ unknowns to be determined. These equations can be written as a linear system as shown below

$$\begin{bmatrix} \Phi & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ a \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix} \tag{6.21}$$

where $\Phi(i, j) = \phi\left(\|x_i - x_j\|_2\right)$, $P(k, j) = p_{j-1}(x_k)$, $\lambda(i) = \lambda_i$, $a(i) = a_{i-1}$, $f(i) = f_i$.

### 6.5.2 Problem specification

Given the mapping of points from the boundary of a disc of unit radius to the boundary of the wiggly surface, the goal is to map the interior of the unit disc to the interior of the wiggly surface. The displacement of a set of $N$ points on the unit circle are specified. These $N$ points are chosen uniformly at random on the unit circle, i.e., we sample $\theta$ from a uniform distribution in the interval $[0, 2\pi)$.

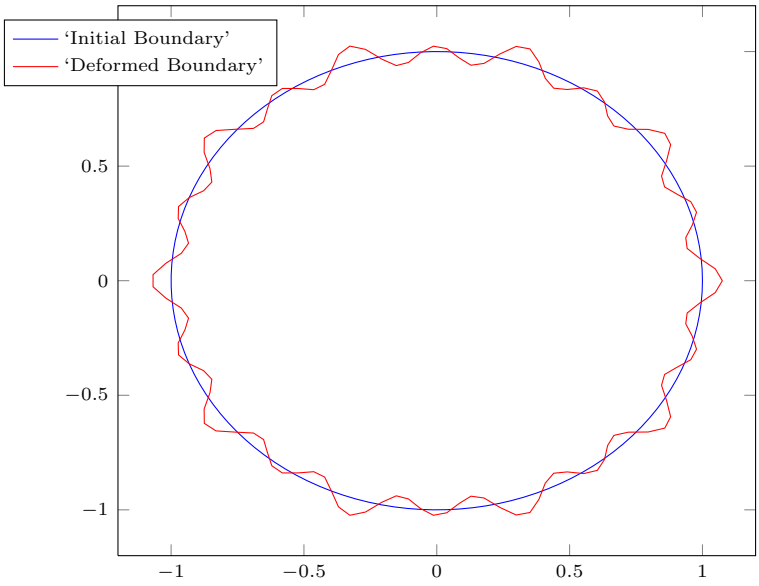



Figure 6.2: Deformation of a one dimensional manifold

The bottleneck in solving these linear interpolation problems using radial basis functions is the solution of the dense linear systems

$$\Phi\,\lambda_1 = P, \qquad \Phi\,\lambda_2 = f \tag{6.22}$$

where $\Phi \in \mathbb{R}^{N\times N}$. To benchmark our algorithm, we compare the time taken by different algorithms to solve equation (6.22) for a variety of commonly used radial basis functions. All the algorithms were implemented in C++. The time taken by the $\mathcal{O}(N \log N)$ and $\mathcal{O}(N \log^2 N)$ algorithms to solve a linear system with one right hand side are compared against the time taken to solve one right hand side using a partially pivoted LU decomposition routine in Eigen [55], to highlight the speedup attained using the proposed algorithm. The relative error was computed by feeding in a known $(\lambda_{\text{exact}}, a_{\text{exact}})$ and comparing these with the results obtained by different algorithms. All the illustrations were run on a machine with a single core 2.66 GHz processor and 8 GB RAM. There was no necessity to parallelize the implementation for the present purpose due to the speedup achieved with the proposed algorithm.

### 6.5.3 Results and discussion

We present a detailed numerical benchmark for the radial basis functions listed in Table 6.2. For all these different radial basis functions, the parameter $a$ was chosen to be the radius of the circle, which in our case is 1. The points on the unit circle are parameterized as $(x, y) = (\cos(\theta), \sin(\theta))$; $r$ denotes the Euclidean distance between two points on the unit circle, i.e., the distance between the points $i$ and $j$ on the circle is given by $r_{ij} = \|x_i - x_j\|_2 = 2\left|\sin\left(\frac{\theta_i - \theta_j}{2}\right)\right|$. Consider for example an off-diagonal block in the matrix

Table 6.2: Radial basis functions $\phi(r)$ considered for numerical benchmarking.

| Quadric | Multi-quadric | Inverse quadric | Inverse multi-quadric |
|---|---|---|---|
| $1 + (r/a)^2$ | $\sqrt{1 + (r/a)^2}$ | $1/(1 + (r/a)^2)$ | $1/\sqrt{1 + (r/a)^2}$ |
| Exponential | Gaussian | Logarithm | |
| $\exp(-r/a)$ | $\exp(-r^2/a^2)$ | $\log(1 + r/a)$ | |

with entries of the form $\Phi_{ij} = \phi(r_{ij}) = \phi(\|x_i - x_j\|_2) = \phi\Big(2\Big|\sin\Big(\frac{\theta_i - \theta_j}{2}\Big)\Big|\Big) = \psi(\theta_i - \theta_j)$, where $\theta_i, \theta_j \in \mathcal{I} = [0, 2\pi)$. The low-rank approximations are constructed by using Chebyshev polynomials that are functions of $\theta$.

In [39, 89], an analysis is presented where the order of Chebyshev polynomials required to build a low-rank approximation is estimated based on the growth of the kernel $\psi$ in the complex plane along an ellipse containing $\mathcal{I} = [0, 2\pi)$. The rank can be shown to be determined primarily by two factors: the distance between the poles of $\psi$ in the complex plane and the interval $\mathcal{I}$, and the growth of $\psi$ in the complex plane. As we chose $a = 1$ (which determines the location of the poles), we therefore expect a rapid decay of the error with the rank.

We performed the following series of tests for each of the radial basis functions in table (6.2). In order to reduce the number of pages, not all our results are shown in the thesis. We, however, ran all the calculations and analyzed all the plots. In cases where many plots were similar, we selected a few representative ones for inclusion. The details of the tests that were performed are given below.

- We looked at how the rank of the off-diagonal blocks grows with $N$ for all the radial basis functions in table (6.2). The system size was made to increase from 1024 to 8192. The decay of the singular-values for the off-diagonal blocks can be found in figure (6.3).

There was no noticeable growth of the rank of the off-diagonal blocks with $N$.

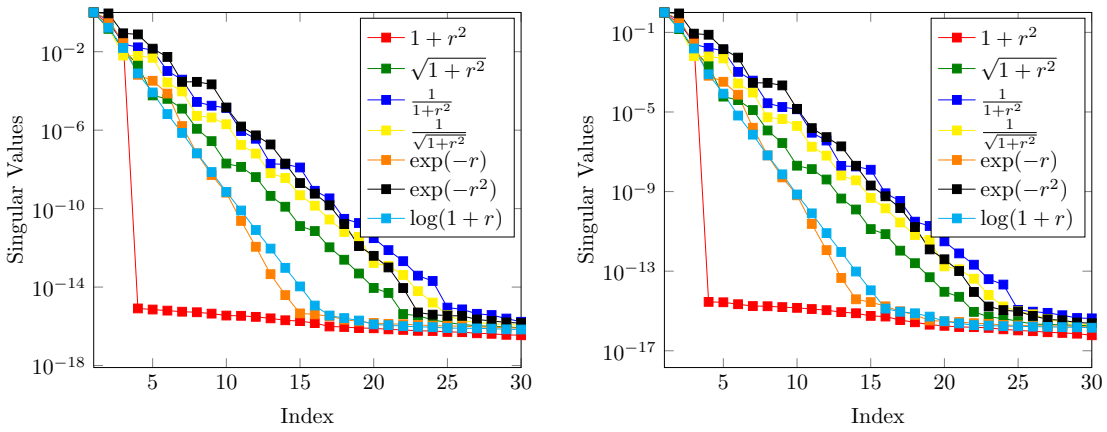



Figure 6.3: Decay of singular values of the off-diagonal blocks for different radial basis functions. The singular values are normalized using the largest singular-value. Left: system size $1024 \times 1024$; Right: system size $8192 \times 8192$.

- We then considered $8192 \times 8192$ matrices. We computed the condition number of the system and the decay of the singular values of the largest off-diagonal block, which is of size $4096 \times 4096$. The condition numbers ranged from $3 \cdot 10^3$ to $2 \cdot 10^7$ as shown in table (6.3). Although a wide range of condition numbers were observed for these matrices, the accuracy of our algorithm was found to be largely independent of the condition number. As explained previously, the accuracy is determined by the decay of the singular values, whose behavior is different from the condition number.

Table 6.3: Condition number of a $8192 \times 8192$ system for different kernels where the points are distributed randomly on a unit circle.

| Kernel | $1+r^2$ | $\sqrt{1+r^2}$ | $1/(1+r^2)$ | $1/\sqrt{1+r^2}$ | $\exp(-r)$ | $\exp(-r^2)$ | $\log(1+r)$ |
|---|---|---|---|---|---|---|---|
| Cond # | $2.5 \cdot 10^4$ | $1.4 \cdot 10^4$ | $1.0 \cdot 10^4$ | $1.5 \cdot 10^4$ | $1.2 \cdot 10^6$ | $3.4 \cdot 10^3$ | $1.7 \cdot 10^7$ |

- For the $8192 \times 8192$ linear system, we computed:
  - The relative error in the solution obtained by using the $\mathcal{O}(N \log^2 N)$ and $\mathcal{O}(N \log N)$ algorithms as a function of the rank of the off-diagonal blocks. As explained

above, the right-hand-side of the system was computed from a known $(\lambda, a)$, so that the exact solution is known.

- The assembly time and solve time taken for both algorithms as a function of the rank.
  - **Assembly time** denotes the time taken to compute the desired entries in the matrix and the desired low-rank factorizations to set up the hierarchical structure. Specifically, this includes the time taken to compute $K_i^{(\kappa)} \in \mathbb{R}^{N/2^\kappa \times N/2^\kappa}$, $U_i^{(\kappa)} \in \mathbb{R}^{N/2^\kappa \times p}$ where $i \in \{1, 2, 3, \ldots, 2^\kappa\}$, $V_{2i-1,2i}^{(k)}$, $V_{2i,2i-1}^{(k)} \in \mathbb{R}^{N/2^k \times p}$ at all levels, $k \in \{1, 2, 3, \ldots, \kappa\}$, $i \in \{1, 2, 3, \ldots, 2^{k-1}\}$ and $S_i^{(k)} \in \mathbb{R}^{p \times p}$ at all levels where $k \in \{2, 3, \ldots, \kappa\}$ and $i \in \{1, 2, 3, \ldots, 2^k\}$.
  - **Solve time** denotes the time taken to compute the solution to the linear system, which includes both the time taken to obtain the factorization (section 6.4.1) and perform the solve (section 6.4.2). Recall that the factorization phase and the solve phase proceed together as mentioned in section 6.4.2.

- Next for all the radial basis functions, we fixed the off-diagonal rank at 30 and used Chebyshev interpolation [39] for $\theta$, to construct a low-rank representation of the off-diagonal blocks. For the $\mathcal{O}(N \log^2 N)$ and $\mathcal{O}(N \log N)$ algorithms, we increase the system size from 128 to 1048576. We compared the two algorithms with the partially pivoted LU algorithm. For the partially pivoted LU algorithm, we increased the system size from 128 to 8192. Beyond this, the partially pivoted LU algorithm took too much time. The following comparisons were made:
  - Relative error in the solution obtained by using the fast algorithms and the partially pivoted LU algorithms as a function of the system size.
  - Assembly time taken for the fast algorithms and the partially pivoted LU algorithms as a function of the system size.
  - Solve time taken for the fast algorithms and the partially pivoted LU algorithms as a function of the system size.

- Next we also analyzed how the cost at each level varied for the two fast algorithms. To do this, we considered a system size of $131072 \times 131072$ and fixed the rank of the off-diagonal blocks at 20. The total number of levels in this case is 13. We measured the time taken to assemble and solve at each level.

All the time taken shown in the figures and tables are in seconds. Most of the tests returned similar results in terms of accuracy and performance. In order to reduce the number of numerical results included, we present detailed tests for $\exp(-r^2)$ only.

#### 6.5.3.1 Gaussian

The Gaussian radial basis function is given by $\phi(r) = \exp(-r^2)$. We present detailed results for this function as it is widely used in many radial basis function interpolation. Another reason is that among all the radial basis functions we considered, the Gaussian and the inverse quadric show the slowest decay of singular-values of the off-diagonal block. For all the radial basis functions, the relative error obtained using the $N \log^2 N$ algorithm is nearly the same as the relative error obtained using the $N \log N$ algorithm. This highlights the fact that in our case the HODLR systems are in fact p-HSS systems as well.

The comparison of these fast algorithms with eigen [55], an efficient C++ linear algebra package, highlights the performance of these fast algorithms. Since we use a partially pivoted LU factorization to solve the linear system using eigen, the total time scales as $O(N^3)$. We observe a huge reduction in running time with these fast new algorithms. The relative error between eigen and the proposed algorithm is also very small because of the rapid decay of the singular values (6.3). Further, between the fast algorithms, the difference in asymptotic scaling between $O(N \log N)$ and $O(N \log^2 N)$ is rather important as the figures (6.4), (6.5) and table (6.4) indicate. A detailed analysis of the assembly time and solve time at each level for the fast algorithms is shown in figure (6.6). Level 0 is the root and level 16 is the leaf. This clearly highlights the difference in the computational cost between the two fast algorithms. For the $\mathcal{O}(N \log^2 N)$ algorithm, the assembly time remains nearly the same across all levels. However, the solve time grows linearly with the number of levels. This is consistent with our analysis in section 6.4.1. In the case of the $\mathcal{O}(N \log N)$ algorithm, the assembly time (except for the last few levels close to the leaf) and the solve time both remain the same at all levels. The fact that the time taken at all levels is almost the same, is again consistent with our analysis in section 6.4.1. The increase in assembly time for the last few levels is due to the fact that there is a proliferation of small problems and hence hardware effects such as the size of the memory cache play a role. Also, at the leaf level, few additional computations are needed to assemble the system and hence this too increases the computation time at the leaf level.

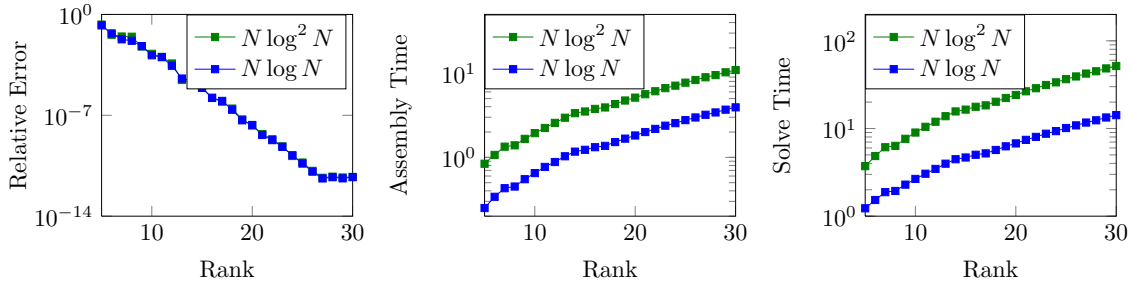



Figure 6.4: Keeping the system size fixed at 8192; Left: Relative error; Middle: Time taken to assemble in seconds; Right: Time taken to solve in seconds; versus rank of the off-diagonal blocks

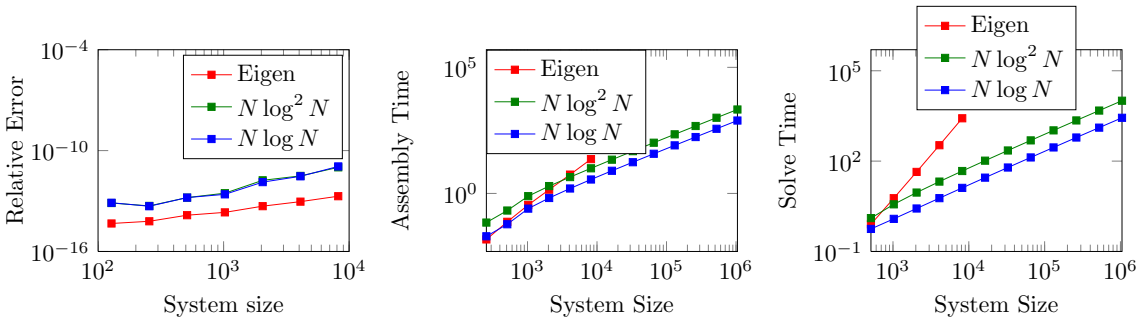



Figure 6.5: Keeping the rank of the off-diagonal block fixed at 30; Left: Relative error; Middle: Time taken to assemble in seconds; Right: Time taken to solve in seconds; versus system size $N$.

Table 6.4: Time taken to solve a linear system as a function of the system size holding the rank of the off-diagonal blocks fixed at 30.

| System Size | Time taken | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Eigen | | | $N \log^2 N$ | | | $N \log N$ | | |
| | Assembly | Solve | Total | Assembly | Solve | Total | Assembly | Solve | Total |
| 256 | $1.5 \cdot 10^{-2}$ | $1.1 \cdot 10^{-1}$ | $1.2 \cdot 10^{-1}$ | $7.1 \cdot 10^{-2}$ | $5.1 \cdot 10^{-1}$ | $5.8 \cdot 10^{-1}$ | $2.0 \cdot 10^{-2}$ | $2.4 \cdot 10^{-1}$ | $2.6 \cdot 10^{-1}$ |
| 512 | $7.5 \cdot 10^{-2}$ | $8.0 \cdot 10^{-1}$ | $8.7 \cdot 10^{-1}$ | $2.1 \cdot 10^{-1}$ | $1.3 \cdot 10^{+0}$ | $1.5 \cdot 10^{+0}$ | $6.2 \cdot 10^{-2}$ | $5.6 \cdot 10^{-1}$ | $6.3 \cdot 10^{-1}$ |
| 1024 | $3.5 \cdot 10^{-1}$ | $5.9 \cdot 10^{+0}$ | $6.2 \cdot 10^{+0}$ | $7.8 \cdot 10^{-1}$ | $3.7 \cdot 10^{+0}$ | $4.5 \cdot 10^{+0}$ | $2.5 \cdot 10^{-1}$ | $1.2 \cdot 10^{+0}$ | $1.5 \cdot 10^{+0}$ |
| 2048 | $1.4 \cdot 10^{+0}$ | $4.4 \cdot 10^{+1}$ | $4.5 \cdot 10^{+1}$ | $2.0 \cdot 10^{+0}$ | $9.1 \cdot 10^{+0}$ | $1.1 \cdot 10^{+1}$ | $6.6 \cdot 10^{-1}$ | $2.7 \cdot 10^{+0}$ | $3.3 \cdot 10^{+0}$ |
| 4096 | $5.6 \cdot 10^{+0}$ | $3.4 \cdot 10^{+2}$ | $3.4 \cdot 10^{+2}$ | $4.5 \cdot 10^{+0}$ | $2.1 \cdot 10^{+1}$ | $2.5 \cdot 10^{+1}$ | $1.6 \cdot 10^{+0}$ | $5.9 \cdot 10^{+0}$ | $7.5 \cdot 10^{+0}$ |
| 8192 | $2.3 \cdot 10^{+1}$ | $2.7 \cdot 10^{+3}$ | $2.7 \cdot 10^{+3}$ | $1.0 \cdot 10^{+1}$ | $4.7 \cdot 10^{+1}$ | $5.7 \cdot 10^{+1}$ | $3.6 \cdot 10^{+0}$ | $1.3 \cdot 10^{+1}$ | $1.7 \cdot 10^{+1}$ |
| 16384 | – | – | – | $2.2 \cdot 10^{+1}$ | $1.0 \cdot 10^{+2}$ | $1.3 \cdot 10^{+2}$ | $7.9 \cdot 10^{+0}$ | $2.8 \cdot 10^{+1}$ | $3.6 \cdot 10^{+1}$ |
| 32768 | – | – | – | $4.7 \cdot 10^{+1}$ | $2.3 \cdot 10^{+2}$ | $2.7 \cdot 10^{+2}$ | $1.7 \cdot 10^{+1}$ | $6.1 \cdot 10^{+1}$ | $7.9 \cdot 10^{+1}$ |
| 65536 | – | – | – | $1.0 \cdot 10^{+2}$ | $4.9 \cdot 10^{+2}$ | $5.9 \cdot 10^{+2}$ | $3.7 \cdot 10^{+1}$ | $1.3 \cdot 10^{+2}$ | $1.7 \cdot 10^{+2}$ |
| 131072 | – | – | – | $2.2 \cdot 10^{+2}$ | $1.1 \cdot 10^{+3}$ | $1.3 \cdot 10^{+3}$ | $8.1 \cdot 10^{+1}$ | $2.8 \cdot 10^{+2}$ | $3.6 \cdot 10^{+2}$ |
| 262144 | – | – | – | $4.7 \cdot 10^{+2}$ | $2.2 \cdot 10^{+3}$ | $2.7 \cdot 10^{+3}$ | $1.7 \cdot 10^{+2}$ | $6.1 \cdot 10^{+2}$ | $7.8 \cdot 10^{+2}$ |
| 524288 | – | – | – | $1.0 \cdot 10^{+3}$ | $4.8 \cdot 10^{+3}$ | $5.8 \cdot 10^{+3}$ | $3.7 \cdot 10^{+2}$ | $1.3 \cdot 10^{+3}$ | $1.7 \cdot 10^{+3}$ |
| 1048576 | – | – | – | $2.1 \cdot 10^{+3}$ | $1.0 \cdot 10^{+4}$ | $1.2 \cdot 10^{+4}$ | $7.8 \cdot 10^{+2}$ | $2.7 \cdot 10^{+3}$ | $3.5 \cdot 10^{+3}$ |

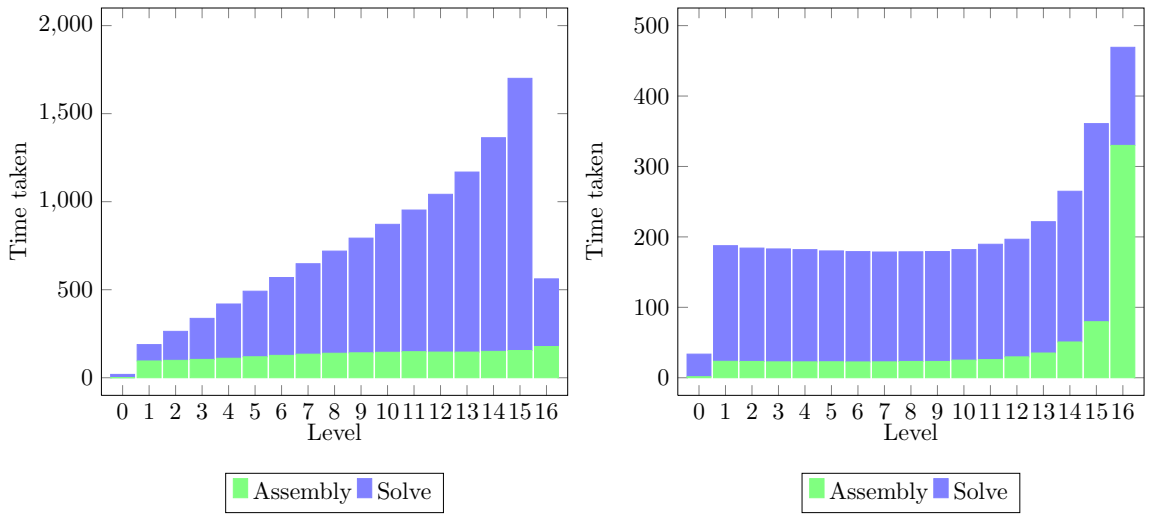



Figure 6.6: Split up of the time taken by the algorithms at each level for a $1,048,576 \times 1,048,576$ system. Left: $N \log^2 N$ algorithm; Right: $N \log N$ algorithm

#### 6.5.3.2 Quadric biharmonic

The quadric biharmonic radial basis function is given by $\phi(r) = 1 + (r/a)^2$. The exact rank of the off-diagonal blocks is 3 since the radial basis function is a quadratic polynomial in $r$, which can also be seen in figure (6.3). However, since we are constructing the low-rank using $\theta$, we have that $\phi(r_{ij}) = 1 + 4\sin^2\left((\theta_i - \theta_j)/2\right)$ and hence around 15 terms are needed to construct a good low-rank approximation. The relative error as a function of rank and system size are plotted in figure (6.7).

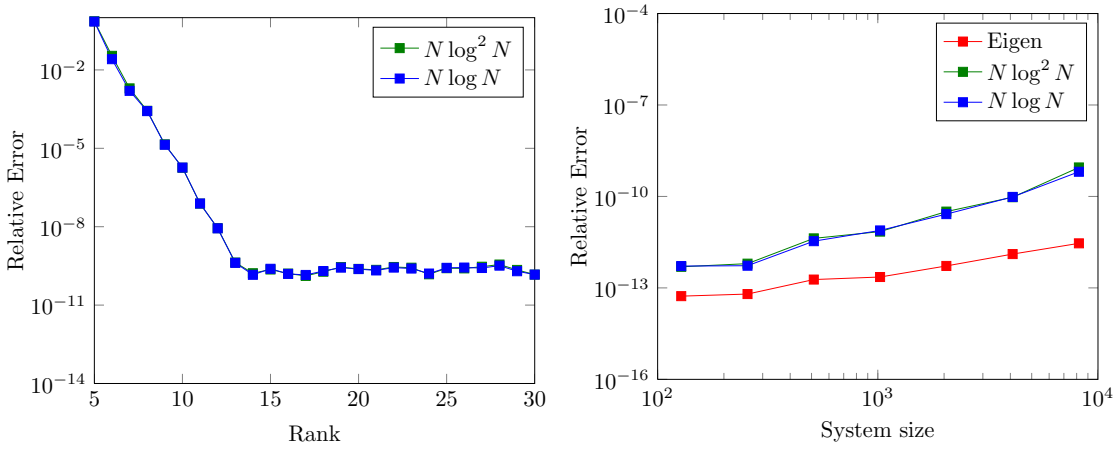



Figure 6.7: Relative error for $1 + (r/a)^2$. Left: versus rank for a system size of 8192 (the green and blue curves overlap); Right: versus system size keeping the off-diagonal rank as 30.

#### 6.5.3.3 Multi-quadric biharmonic

The radial basis function is given by $\phi(r) = \sqrt{1 + (r/a)^2}$. The decay of the singular-values of the off-diagonal blocks is moderate and the $25^{th}$ singular-value is close to machine precision as seen in figure (6.3). The relative error as a function of rank and system size are plotted in figure (6.8).

#### 6.5.3.4 Inverse quadric biharmonic

The radial basis function is given by $\phi(r) = \frac{1}{1+(r/a)^2}$. The decay of the singular-values of the off-diagonal blocks is very similar to the decay of the Gaussian. As before, the $25^{th}$

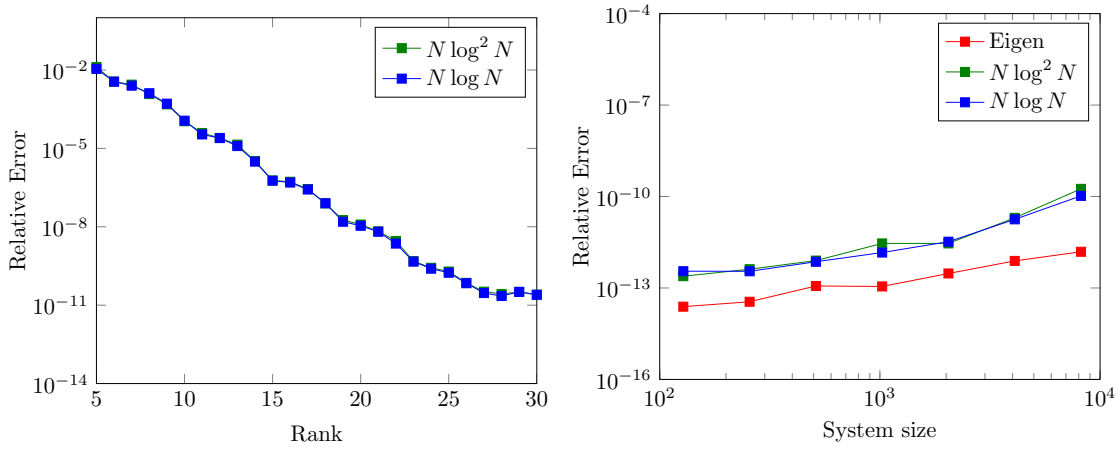



Figure 6.8: Relative error for $\sqrt{1+(r/a)^2}$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30.

singular-value is close to machine precision as seen in figure (6.3). The relative error as a function of rank and system size are plotted in figure (6.9).

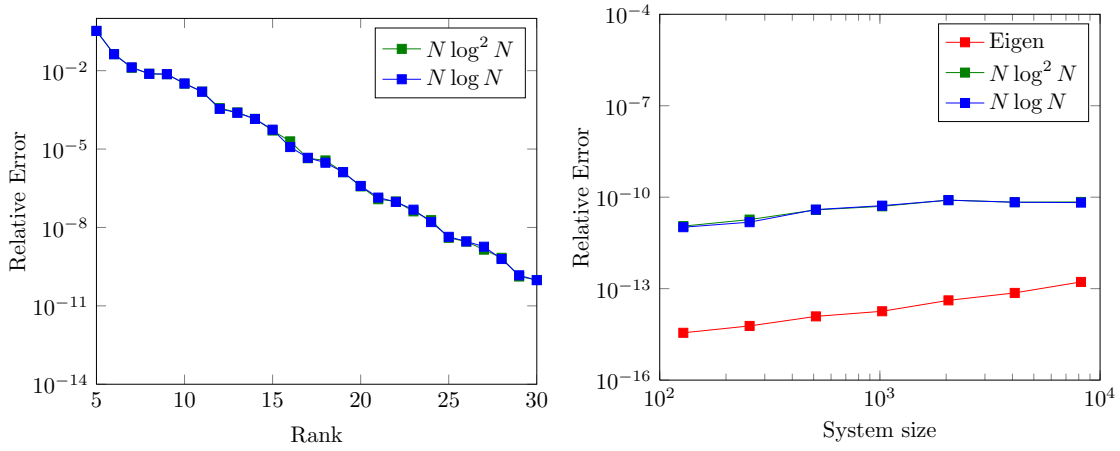



Figure 6.9: Relative error for $\frac{1}{1+(r/a)^2}$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30.

#### 6.5.3.5 Inverse multi-quadric biharmonic

The radial basis function is given by $\phi(r) = \frac{1}{\sqrt{1+(r/a)^2}}$. The decay of the singular-values of the off-diagonal blocks is slightly faster than inverse quadric biharmonic but slower than multi-quadric biharmonic. As before, the $25^{th}$ singular-value is close to machine precision as seen in figure (6.3). The relative error as a function of rank and system size are plotted in figure (6.10).

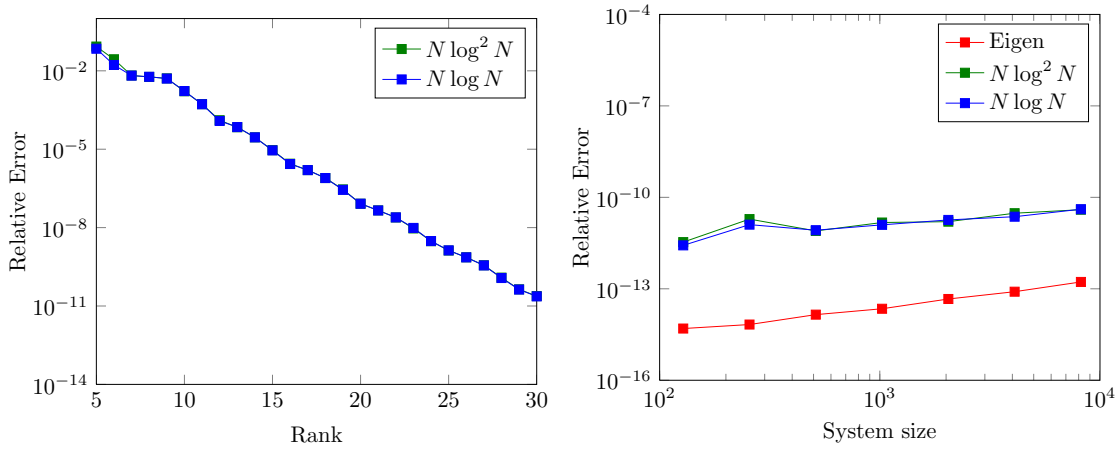



Figure 6.10: Relative error for $\frac{1}{\sqrt{1+(r/a)^2}}$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30.

#### 6.5.3.6 Exponential

The radial basis function is given by $\phi(r) = \exp(-r/a)$. The decay of the singular-values of the off-diagonal blocks is rapid. The $15^{th}$ singular-value is close to machine precision as seen in figure (6.3). It is to be noted that even though the condition number of the system formed using the exponential radial basis function is relatively large, i.e., around $2 \cdot 10^6$, this does not seem to affect the relative error of the solution obtained using the fast algorithm as seen in figure (6.11).

#### 6.5.3.7 Logarithm

The radial basis function is given by $\phi(r) = \log(1 + r/a)$. The decay of the singular-values of the off-diagonal blocks is similar to $\exp(-r/a)$. The $15^{th}$ singular-value is close to machine

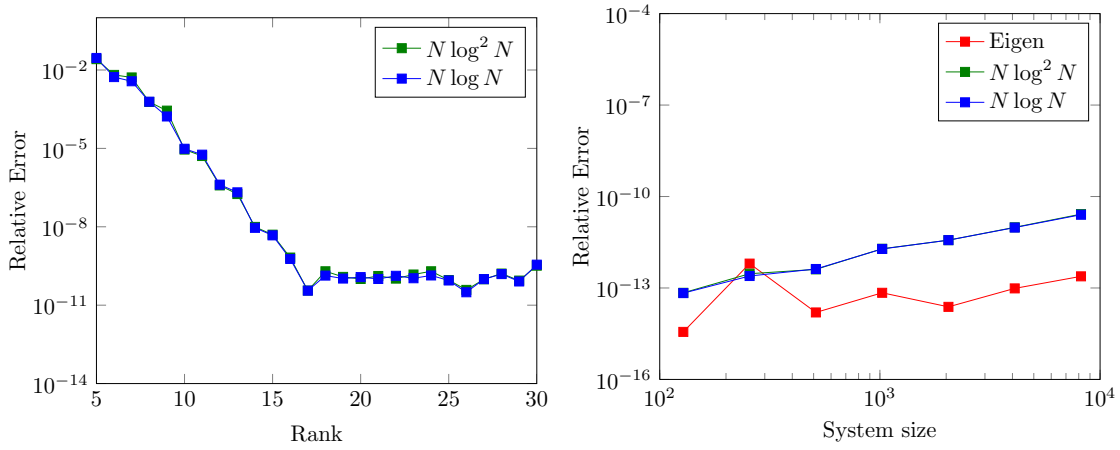



Figure 6.11: Relative error for $\exp(-r/a)$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30.

precision as seen in figure (6.3). It is also to be noted that, similar to the exponential radial basis function, the condition number for the system formed using the logarithm radial basis function is also large, i.e., around $2 \cdot 10^7$. However, this does not seem to affect the relative error of the solution obtained using the fast algorithm as seen in figure (6.12).

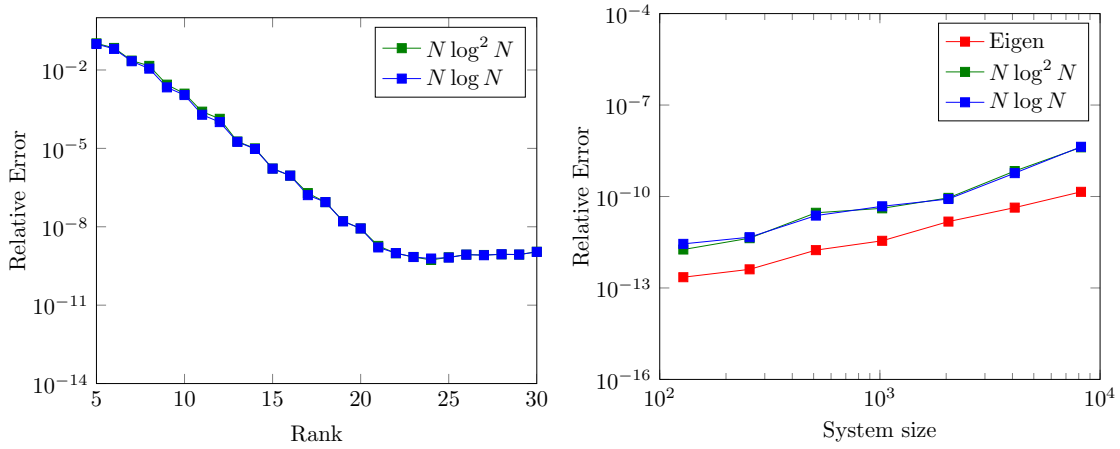



Figure 6.12: Relative error for $\log(1 + r/a)$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30.

## 6.6 Conclusions

We have presented a new algorithm for solving linear systems that have a partial hierarchically semi-separable structure. Such systems occur frequently in many applications, for example when the underlying kernel is smooth and non-oscillatory. The algorithm presented has a computational complexity of $\mathcal{O}(N \log N)$ and a storage cost of $\mathcal{O}(N \log N)$. We have illustrated the application of the solver with detailed numerical benchmarks. These numerical benchmarks validate the fact that the computational complexity is $\mathcal{O}(N \log N)$ and the robustness of the method. The main advantage of this algorithm is that it is not only conceptually easy to understand and implement, but also quite general and robust as the numerical benchmarks indicate. The algorithm was implemented both in C++ and MATLAB.

# Chapter 7

# Extended sparsification based fast direct solvers

## 7.1 Introduction

In this chapter, we discuss another new approach to solve HODLR and HSS matrices. This new approach is promising since it not only constructs an $\mathcal{O}(N)$ linear solver for the HSS linear system but also could also be extended to more general $\mathcal{H}$ and $\mathcal{H}^2$ linear systems. As discussed in chapter 3, all the different hierarchical structures fall into four main classes:

1. Hierarchically Off-Diagonal Low-Rank, abbreviated as HODLR.
2. Hierarchically Semi-Separable, abbreviated as HSS.
3. Generic hierarchical matrix, denoted as $\mathcal{H}$ matrix.
4. Generic hierarchical matrix with nested low-rank structure, $\mathcal{H}^2$ matrix.

Matrix vector products for the above hierarchical matrices can be computed in almost linear complexity 7.1. Also as discussed in chapter 3 matrices arising out of the fast multipole [50, 51] algorithm are a strict subclass of the $\mathcal{H}^2$ matrices. Recently, there has been an increasing focus on constructing fast direct solvers for these hierarchical matrices. This endeavor has been fruitful in the case of HODLR and HSS matrices [2, 21, 23, 65, 79, 88]. Fast direct solvers for $\mathcal{H}$ and $\mathcal{H}^2$ matrices is still active research area and we refer the readers to [59–62] for some progress in this direction.

Table 7.1: Computational cost for matrix vector product for different hierarchical structures.

| HODLR | HSS | $\mathcal{H}$ | $\mathcal{H}^2$ |
|---|---|---|---|
| $\mathcal{O}(N \log N)$ | $\mathcal{O}(N)$ | $\mathcal{O}(N \log N)$ | $\mathcal{O}(N)$ |

The present work attempts to present a unified approach for solving all these different hierarchical matrices. The key, new idea behind this approach is that one form of sparsity can be converted into another form of sparsity, which can then be exploited. To be specific, the data-sparse hierarchical matrix can exactly be represented as a larger structured sparse matrix. (We use the term sparse matrix to mean zero fill-in. This should not be confused with data-sparsity.) The structure of the larger sparse matrix can then be exploited to solve the linear system in almost linear complexity. Table 7.2 presents the computational complexity of the fast direct solvers discussed in this chapter. We also illustrate the algorithm

Table 7.2: Computational cost for the fast direct solver discussed in this chapter.

| HODLR | HSS |
|---|---|
| $\mathcal{O}(N \log^2 N)$ | $\mathcal{O}(N)$ |

by benchmarking it for the different hierarchical structures. The next section is the core of the work. It presents the different hierarchical structures, discusses the algorithm for solving these systems in almost linear complexity and provides numerical benchmark. We briefly discuss constructing the different hierarchical matrices, but we keep the discussion to a minimum and provide references so that the reader can look up constructing the dense hierarchical structures in almost linear complexity. The main focus of this chapter are the following:

1. Converting a hierarchical structure into a **larger sparser matrix**.
2. **Universality of the self-similar sparse matrix structure** for a given hierarchical matrix.
3. **Pattern of fill-in** during fast LU factorization of the larger sparse matrix (equivalently, a fast direct solver for the original hierarchical matrix) by optimal ordering of the unknowns and equations of the larger sparse matrix.

## 7.2 Extended sparsification approach

As shown in the Euler diagram in Figure 3.1, there are different hierarchical structures depending on the tree structure and algorithms to obtain low-rank different matrix. In this chapter, we shall focus on constructing fast solvers for hierarchical matrices with weak admissibility criteria, i.e., HSS and HODLR matrices. It is to be noted that the technique discussed in the next few sections can be extended to hierarchical matrix with strong admissibility criteria, i.e., $\mathcal{H}$ and $\mathcal{H}^2$ matrices, as well. This is the subject of my current work.

Before discussing our algorithm, we present a brief discussion of previous works in this direction and our new contribution. The idea of extended sparsification has been considered before in the article by Chandrasekaran et al. [23], though only in the context of HSS matrices. The algorithm for HSS matrices is fairly easier since in the elimination process there are *no new fill-ins*. In our approach, we also deal with a slightly larger class of HODLR matrices.

It is also to be noted that Greengard et al. [53] present the idea of representing the dense matrix as an extended sparse matrix. The article presents a single-level fast solver whose scaling is not $\mathcal{O}(N)$ and due to which compressing the fill-ins, which is important to extend the strategy in a multi-level setting, is not discussed in detail. In the figures discussed in the next few sub-section, the following color code will be followed. Red color will be used to denote matrices capturing the self-self interaction and the translation (M2L) matrices. Green color will be used to denote the gather/anterpolation/M2M operator and the scatter/interpolation/L2L matrices. Blue color will denote the negative identity matrix.

### 7.2.1 Fast direct solver for HODLR matrices

We are interested in solving an equation of the form

$$K_1^{(0)} x^{(0)} = b^{(0)}$$

where $K_1^{(0)} \in \text{HODLR}(N, r, \kappa)$, $x^{(0)}, b^{(0)} \in \mathbb{R}^{N\times 1}$. We refer the readers to [2, 79] and the earlier chapters 3 and 6, where this structure has been discussed in more detail. This dense HODLR linear system is converted into a larger sparse structured system as follows. Let us

proceed one level into the tree, we then have

$$\begin{bmatrix} K_1^{(1)} & U_1^{(1)}K_{1,2}^{(1)}V_2^{(1)^T} \\ U_2^{(1)}K_{2,1}^{(1)}V_1^{(1)^T} & K_2^{(1)} \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} \tag{7.1}$$

Now introduce the following new variables:

$$y_1^{(1)} = V_1^{(1)^T}x_1^{(1)};\ y_2^{(1)} = V_2^{(1)^T}x_2^{(1)};\ z_1^{(1)} = K_{1,2}^{(1)}y_2^{(1)};\ z_2^{(1)} = K_{2,1}^{(1)}y_1^{(1)}$$

For those familiar, with the FMM, $y_i^{(k)}$'s account for the particle to multipole tranfer, and $z_i^{(k)}$'s account for the multipole to local transfer. In the context of FMM, the $y_i^{(k)}$'s are termed the multipole coefficients and $z_i^{(k)}$'s are termed the local coefficients. We shall borrow the terminology. Introducing these new variables and reordering the variables and equations appropriately, we see that (7.1) can be rewritten as (7.4). Now stepping one more level into the tree, we see that Equation (7.4) becomes (7.5). As with the previous level, let us introduce the following new variables; The variables that account for the particle to multipole transfer:

$$y_1^{(2)} = V_1^{(2)^T}x_1^{(2)};\ y_2^{(2)} = V_2^{(2)^T}x_2^{(2)};\ y_3^{(2)} = V_3^{(2)^T}x_3^{(2)};\ y_4^{(2)} = V_4^{(2)^T}x_4^{(2)};$$

and the variables that account for the multipole to local transfer:

$$z_1^{(2)} = K_{1,2}^{(2)}y_2^{(2)};\ z_2^{(2)} = K_{2,1}^{(2)}y_1^{(2)};\ z_3^{(2)} = K_{3,4}^{(2)}y_4^{(2)};\ z_4^{(2)} = K_{4,3}^{(2)}y_3^{(2)};$$

Again, with these new variables, reordering the equations and variables, we see that (7.5) can be written as (7.6). Hence, solving the linear system $K_1^{(0)}x_1^{(0)} = b_1^{(0)}$ is equivalent to solving the new sparser system in Equation (7.2).

$$K_{\text{sparse}}x_{\text{new}} = b_{\text{new}} \tag{7.2}$$

where $K_{\text{sparse}}$ is the sparsified new matrix as seen in Equations (7.4)– (7.6) and Figure 7.1,

$$x_{\text{new}} = \begin{bmatrix} x_1^{(0)^T} & z^{(\kappa)^T} & y^{(\kappa)^T} & z^{(\kappa-1)^T} & y^{(\kappa-1)^T} & \dots & z^{(1)^T} & y^{(1)^T} \end{bmatrix}^T \text{ and } b_{\text{new}} = \begin{bmatrix} b_1^{(0)} \\ 0 \end{bmatrix},$$

The vectors $z^{(k)}$ and $y^{(k)}$ consists of all the local and multipole coefficients, respectively, at level $k$, i.e.,

$$z^{(k)^T} = \begin{bmatrix} z_1^{(k)} & z_2^{(k)} & \cdots & z_{2^k}^{(k)} \end{bmatrix} \text{ and } y^{(k)^T} = \begin{bmatrix} y_1^{(k)} & y_2^{(k)} & \cdots & y_{2^k}^{(k)} \end{bmatrix}.$$

We can see the pattern arising by looking at Equations (7.4) and (7.6). We will now pictorially describe the pattern of a $\kappa$ level HODLR matrix, i.e, $K_1^{(0)} \in \text{HODLR}(N, r, \kappa)$. In all the pictorial illustration, we will assume that $r = \dfrac{N}{2^{\kappa+1}}$. Figure 7.1 presents the pictorial pattern in the sparse equivalent of the hierarchical matrix (we have taken $\kappa = 4$). The variables and equations are reordered in such a way that the elimination proceeds from the top-left and proceeds along the south-east direction, as shown in the set of figures. To be specific, the ordering of the variables and equations in the matrix are as described in Table 7.3 and Table 7.4 respectively. *It is to be noted that the ordering of the variables and equations*

Table 7.3: Ordering of unknowns

- The first set of variables are the unknowns of the original dense matrix, i.e., $x_i^{(\kappa)}$, where $i \in \{1, 2, \ldots, 2^\kappa\}$. These variables are then followed by
- For level $k = \kappa \to 1$,
  - The $z_i^{(k)}$'s: The local coefficients at level $k$, where $i \in \{1, 2, \ldots, 2^k\}$.
  - The $y_i^{(k)}$'s: The multipole coefficients at level $k$, where $i \in \{1, 2, \ldots, 2^k\}$.

*is crucial not only to maintain the self-similar sparsity pattern as we proceed through the elimination process but more importantly to minimize the fill-in. Another important aspect is that this ordering of unknowns and equations —to minize the fill-in and gain computational mileage —is universal for all hierarchical matrices, irrespective of them being HODLR or HSS or $\mathcal{H}$ or $\mathcal{H}^2$. We will return to this when we discuss the other hierarchical structures.*

The elimination process —the self-similar sparsity structure and the fill-in—is pictorially represented through Figures 7.1– 7.9. Note that after the first set of eliminations, there is a fill-in of $\mathcal{O}(rN\log^2(N))$ as seen in Figure 7.2. However, in the subsequent steps, there are no additional fill-ins. From the set of figures, it is also clear that the computational cost can be obtained using the recurrence (7.3), where $C(N; k, r)$ denotes the computational cost for

Table 7.4: Ordering of equations

- The first set of equations contain the $x_i^{(\kappa)}$'s and the local coefficients at all levels, i.e., $z_i^{(k)}$'s, where $k \in \{1, 2, \ldots, \kappa\}$ and $i \in \{1, 2, \ldots, 2^k\}$, i.e., the particle to local transfer at all levels. This is followed by the equations capturing the relation between
- For level $k = \kappa \to 1$
  - The $x_i^{(\kappa)}$'s and the multipole coefficients, $y_i^{(k)}$, where $i \in \{1, 2, \ldots, 2^k\}$, i.e., the particle to multipole transfer at level $k$.
  - The local coefficients, $z_i^{(k)}$ and the multipole coefficients, $y_i^{(k)}$, where $i \in \{1, 2, \ldots, 2^k\}$, i.e., the multipole to local transfer at level $k$.

the factorization of a $k$–level $N \times N$ HODLR matrix and $r$ is the rank of the off-diagonal blocks at each level.

$$C(N; k, r) = C\left(\frac{N}{2}; k-1, r\right) + \mathcal{O}(Nk^2r^2) \tag{7.3}$$

where $N$ is the size of the matrix at level $k$. Solving the recurence gives us

$$C(N; \kappa, r) = \mathcal{O}\left(N\kappa^2r^2\right) + \mathcal{O}\left(\frac{N}{2}(\kappa-1)^2r^2\right) + \cdots$$

where $\kappa = \log_2\left(\frac{N}{2r}\right)$. Hence,

$$C(N; \kappa, r) = \mathcal{O}\left(\sum_{k=1}^{\kappa} \frac{N}{2^{\kappa-k}}k^2r^2\right) = \mathcal{O}\left(\frac{N}{2^\kappa}2^\kappa\kappa^2r^2\right) = \mathcal{O}\left(r^2N\log^2(N)\right)$$

$$\begin{bmatrix} K_1^{(1)} & 0 & U_1^{(1)} & 0 & 0 & 0 \\ 0 & K_2^{(1)} & 0 & U_2^{(1)} & 0 & 0 \\ V_1^{(1)^T} & 0 & 0 & 0 & -I & 0 \\ 0 & V_2^{(1)^T} & 0 & 0 & 0 & -I \\ 0 & 0 & -I & 0 & 0 & K_{1,2}^{(1)} \\ 0 & 0 & 0 & -I & K_{2,1}^{(1)} & 0 \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ z_1^{(1)} \\ z_2^{(1)} \\ y_1^{(1)} \\ y_2^{(1)} \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{7.4}$$

$$\begin{bmatrix} \begin{bmatrix} K_1^{(2)} & U_1^{(2)} K_{1,2}^{(2)} V_2^{(2)^T} \\ U_2^{(2)} K_{2,1}^{(2)} V_1^{(2)^T} & K_2^{(2)} \end{bmatrix} & 0 & U_1^{(1)} & 0 & 0 & 0 \\ 0 & \begin{bmatrix} K_3^{(2)} & U_3^{(2)} K_{3,4}^{(2)} V_4^{(2)^T} \\ U_4^{(2)} K_{4,3}^{(2)} V_3^{(2)^T} & K_4^{(2)} \end{bmatrix} & 0 & U_2^{(1)} & 0 & 0 \\ V_1^{(1)^T} & 0 & 0 & 0 & -I & 0 \\ 0 & V_2^{(1)^T} & 0 & 0 & 0 & -I \\ 0 & 0 & -I & 0 & 0 & K_{1,2}^{(1)} \\ 0 & 0 & 0 & -I & K_{2,1}^{(1)} & 0 \end{bmatrix} \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \\ x_4^{(2)} \\ z_1^{(1)^T} \\ z_2^{(1)^T} \\ y_1^{(1)^T} \\ y_2^{(1)^T} \end{bmatrix} = \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \\ b_4^{(2)} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{7.5}$$

$$
\begin{bmatrix}
K_1^{(2)} & 0 & 0 & 0 & U_1^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & U_1^{(1)} & 0 & 0 & 0 \\
0 & K_2^{(2)} & 0 & 0 & 0 & U_2^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & U_2^{(1)} & 0 & 0 \\
0 & 0 & K_3^{(2)} & 0 & 0 & 0 & U_3^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & K_4^{(2)} & 0 & 0 & 0 & U_4^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
V_1^{(2)^T} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & V_2^{(2)^T} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & V_3^{(2)^T} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & V_4^{(2)^T} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & K_{1,2}^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & K_{2,1}^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & K_{3,4}^{(2)} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & K_{4,3}^{(2)} & 0 & 0 & 0 & 0 & 0 \\
V_1^{(1)^T} & & 0 & & 0 & & 0 & & 0 & & 0 & & 0 & 0 & -I & 0 \\
0 & & V_2^{(1)^T} & & 0 & & 0 & & 0 & & 0 & & 0 & 0 & 0 & -I \\
0 & & 0 & & 0 & & 0 & & 0 & & 0 & & -I & 0 & 0 & K_{1,2}^{(1)} \\
0 & & 0 & & 0 & & 0 & & 0 & & 0 & & 0 & -I & K_{2,1}^{(1)} & 0
\end{bmatrix}
\begin{bmatrix}
x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \\ x_4^{(2)} \\ z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \\ z_4^{(2)} \\ y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \\ y_4^{(2)} \\ z_1^{(1)^T} \\ z_2^{(1)^T} \\ y_1^{(1)^T} \\ y_2^{(1)^T}
\end{bmatrix}
=
\begin{bmatrix}
b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \\ b_4^{(2)} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0
\end{bmatrix}
\tag{7.6}
$$

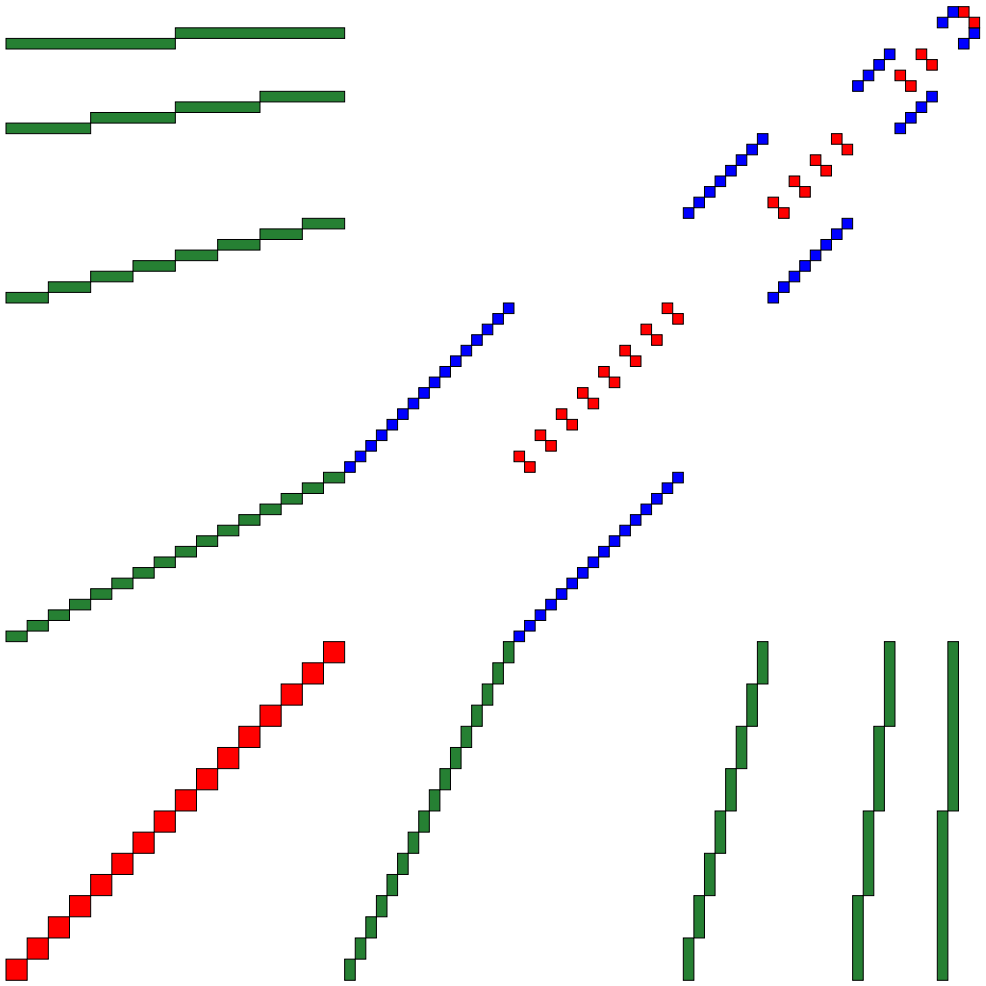


Figure 7.1: The sparsified matrix at the lowest level

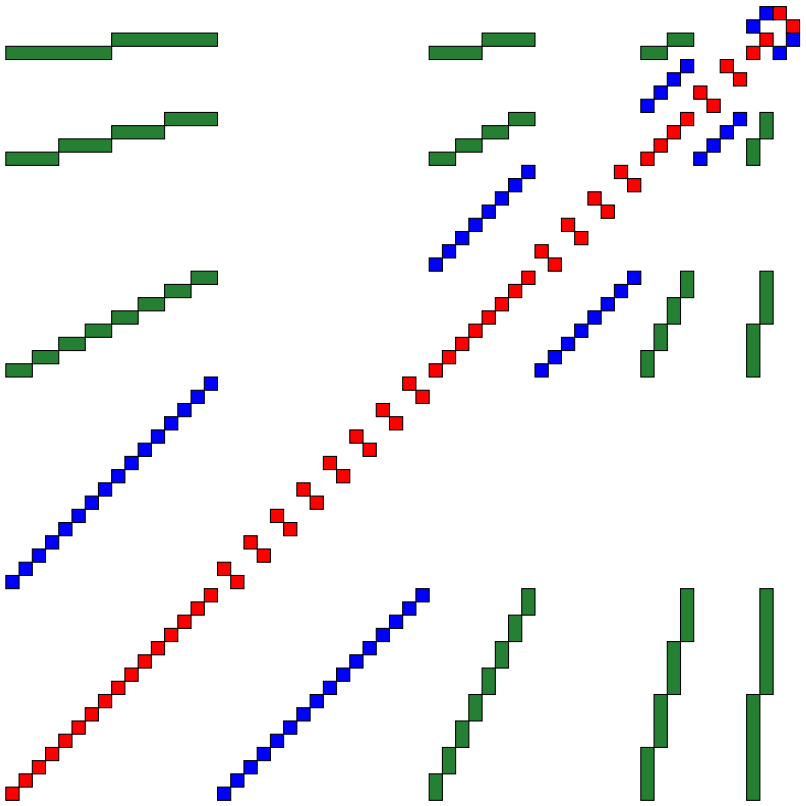


Figure 7.2: The sparsified matrix: After the first set of eliminations.

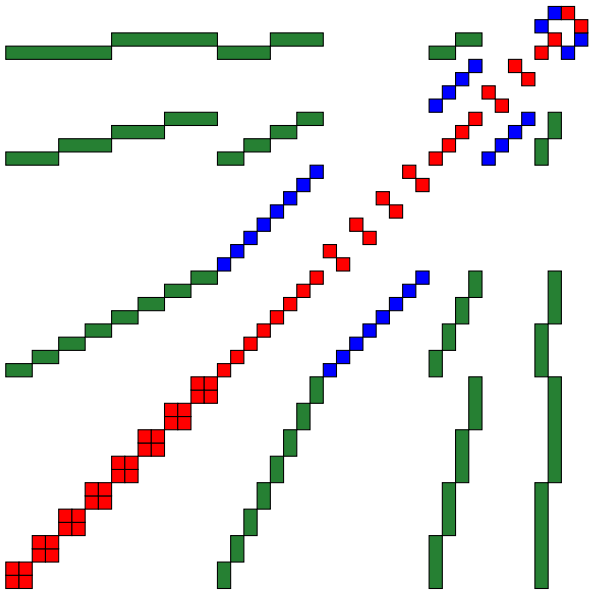


Figure 7.3: The sparsified matrix: After the second set of eliminations.

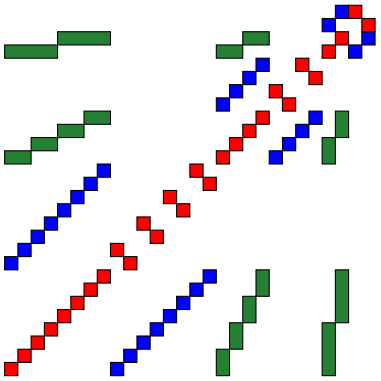


Figure 7.4: The sparsified matrix: After the third set of eliminations.

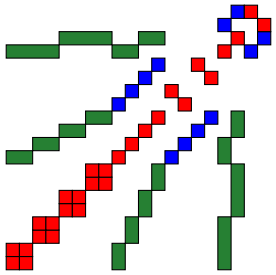


Figure 7.5: The sparsified matrix: After the fourth set of eliminations.

Figure 7.6: The sparsified matrix: After the fifth set of eliminations.



Figure 7.7: The sparsified matrix: After the sixth set of eliminations.



Figure 7.8: The sparsified matrix: After the seventh set of eliminations.



Figure 7.9: The sparsified matrix: After the eigth set of eliminations.

### 7.2.2 Numerical benchmark

We present a numerical benchmark for the above algorithm on HODLR matrices. We consider a set of points distributed uniformly at random in the interval $[-1, 1]$. The matrix of interaction is given by the kernel $K(x_i, x_j) = \exp(-|x_i - x_j|)$. The rank of the off-diagonal blocks is taken as 8 at all levels.

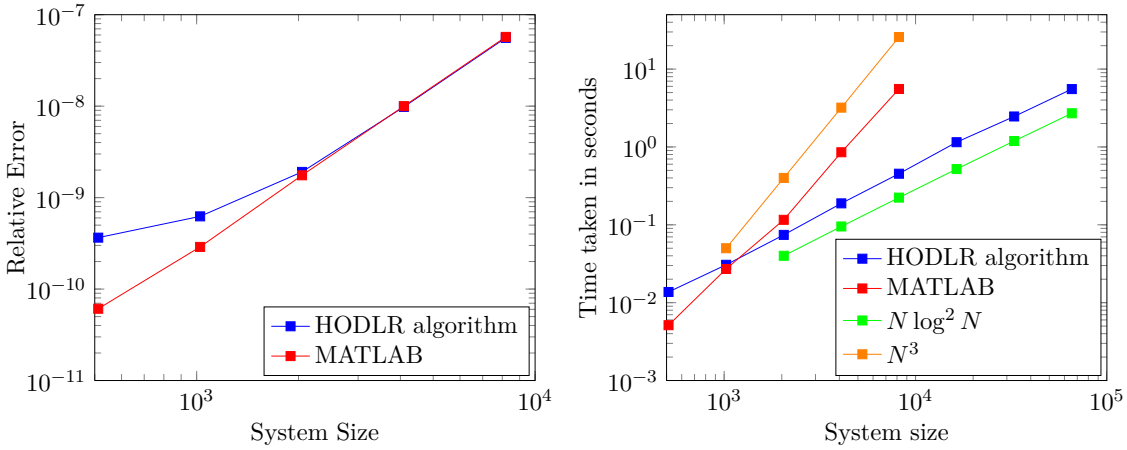



Figure 7.10: Left: Comparison of relative error versus system size; Right: Comparison of time taken versus system size.

### 7.2.3 Fast direct solver for HSS matrices

Consider $K_1^{(0)} x_1^{(0)} = b_1^{(0)}$, where $K_1^{(0)} \in \text{HSS}(N, r, \kappa)$, $x = \begin{bmatrix} x_1^{(\kappa)} & x_2^{(\kappa)} & x_3^{(\kappa)} & \cdots & x_{2^\kappa}^{(\kappa)} \end{bmatrix}^T$ and $b = \begin{bmatrix} b_1^{(\kappa)} & b_2^{(\kappa)} & b_3^{(\kappa)} & \cdots & b_{2^\kappa}^{(\kappa)} \end{bmatrix}^T$. Refer the earlier chapters 3 and 6 for a discussion of HSS matrices. As with the HODLR matrices, let us introduce the new variables. In the HSS case, the new variables will take into account: the particle to multipole transfer at the lowest level, the multipole to multipole transfer at the rest of the levels and the multipole to local/particle transfer at the different levels.

1. Particle to multipole at the leaf level:

$$y_\ell^{(\kappa)} = V_\ell^{(\kappa)^T} x_\ell^{(\kappa)},$$

where $\ell \in \{1, 2, \ldots, 2^\kappa\}$. $y_\ell^{(\kappa)}$'s are the multipole coefficients at the $\kappa^{th}$ level and take into account the particle to multipole tranfer.

2. Multipole to Multipole:

$$y_\ell^{(k)} = R_{2\ell-1}^{(k+1)^T} y_{2\ell-1}^{(k+1)} + R_{2\ell}^{(k+1)^T} y_{2\ell}^{(k+1)},$$

where $\ell \in \{1, 2, \ldots, 2^k\}$ and $k \in \{\kappa-1, \kappa-2, \ldots, 1\}$. $y_\ell^{(k)}$'s are the multipole coefficients at the $k^{th}$ level and take into account the multipole to multipole tranfer.

3. Multipole to Local and Local to local:

$$z_1^{(1)} = K_{1,2}^{(1)} y_2^{(1)};\ z_2^{(1)} = K_{2,1}^{(1)} y_1^{(1)}$$

$$z_{2\ell-1}^{(k)} = \underbrace{K_{2\ell-1,2\ell}^{(k)} y_{2\ell}^{(k)}}_{\text{Multipole to local}} + \underbrace{S_{2\ell-1}^{(k)} z_\ell^{(k-1)}}_{\text{Local to local}};\ z_{2\ell}^{(k)} = \underbrace{K_{2\ell,2\ell-1}^{(k)} y_{2\ell-1}^{(k)}}_{\text{Multipole to local}} + \underbrace{S_{2\ell}^{(k)} z_\ell^{(k-1)}}_{\text{Local to local}}$$

4. Local to particle at the leaf level:

$$b_\ell^{(\kappa)} = K_\ell^{(\kappa)} x_\ell^{(\kappa)} + U_\ell^{(\kappa)} z_\ell^{(\kappa)}$$

where $\ell \in \{1, 2, \ldots, 2^\kappa\}$.

#### 7.2.3.1 Sparse representation of HSS matrices

Let us illustrate this in detail for a 2 level HSS tree and pictorially explain this for a 4 level HSS tree.

$$K = \begin{bmatrix} K_1^{(1)} & U_1^{(1)} K_{1,2}^{(1)} V_2^{(1)^T} \\ U_2^{(1)} K_{2,1}^{(1)} V_1^{(1)^T} & K_2^{(1)} \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} K_1^{(2)} & U_1^{(2)} K_{1,2}^{(2)} V_2^{(2)^T} \\ U_2^{(2)} K_{2,1}^{(2)} V_1^{(2)^T} & K_2^{(2)} \end{bmatrix} & \begin{bmatrix} U_1^{(2)} S_1^{(2)} \\ U_2^{(2)} S_2^{(2)} \end{bmatrix} K_{1,2}^{(1)} \begin{bmatrix} V_3^{(2)} R_3^{(2)} \\ V_4^{(2)} R_4^{(2)} \end{bmatrix}^T \\ \begin{bmatrix} U_3^{(2)} S_3^{(2)} \\ U_4^{(2)} S_4^{(2)} \end{bmatrix} K_{2,1}^{(1)} \begin{bmatrix} V_1^{(2)} R_1^{(2)} \\ V_2^{(2)} R_2^{(2)} \end{bmatrix}^T & \begin{bmatrix} K_3^{(2)} & U_3^{(2)} K_{3,4}^{(2)} V_4^{(2)^T} \\ U_4^{(2)} K_{4,3}^{(2)} V_3^{(2)^T} & K_4^{(2)} \end{bmatrix} \end{bmatrix}$$

1. Particle to multipole tranfer:

$$y_1^{(2)} = V_1^{(2)^T} x_1^{(2)};\ y_2^{(2)} = V_2^{(2)^T} x_2^{(2)};\ y_3^{(2)} = V_3^{(2)^T} x_3^{(2)};\ y_4^{(2)} = V_4^{(2)^T} x_4^{(2)}$$

2. Multipole to multipole tranfer:

$$y_1^{(1)} = R_1^{(2)^T} y_1^{(2)} + R_2^{(2)^T} y_2^{(2)};\ y_2^{(1)} = R_3^{(2)^T} y_3^{(2)} + R_4^{(2)^T} y_4^{(2)}$$

3. Multipole to local and local to local transfers:

$$z_1^{(1)} = K_{1,2}^{(1)} y_2^{(1)};\ z_2^{(1)} = K_{2,1}^{(1)} y_1^{(1)}$$

$$\begin{aligned} z_1^{(2)} = K_{1,2}^{(2)} y_2^{(2)} + S_1^{(2)} z_1^{(1)};\ z_2^{(2)} = K_{2,1}^{(2)} y_1^{(2)} + S_2^{(2)} z_1^{(1)};\\ z_3^{(2)} = K_{3,4}^{(2)} y_4^{(2)} + S_3^{(2)} z_2^{(1)};\ z_4^{(2)} = K_{4,3}^{(2)} y_3^{(2)} + S_4^{(2)} z_2^{(1)} \end{aligned}$$

4. Local to particle transfer:

$$\begin{aligned} b_1^{(2)} = K_1^{(2)} x_1^{(2)} + U_1^{(2)} z_1^{(2)};\ b_2^{(2)} = K_2^{(2)} x_2^{(2)} + U_2^{(2)} z_2^{(2)};\\ b_3^{(2)} = K_3^{(2)} x_3^{(2)} + U_3^{(2)} z_3^{(2)};\ b_4^{(2)} = K_4^{(2)} x_4^{(2)} + U_4^{(2)} z_4^{(2)} \end{aligned}$$

Introducing these variables and with appropriate ordering of equations and unknowns we obtain the Equation (7.8). The ordering of the variables remains the same as shown in Table 7.3. The ordering of the equations remain more or less the same and this has been discussed in Table 7.5. *It is to be noted that, as in the HODLR case, the ordering of the variables and equations is crucial for the same reasons.* Hence, solving $K_1^{(0)} x_1^{(0)} = b_1^{(0)}$ is equivalent to solving $K_{\text{sparse}} x_{\text{new}} = b_{\text{new}}$, where a 2-level $K_{\text{sparse}}$ is as shown in Equation (7.8) and a 4-level $K_{\text{sparse}}$ is as shown in Figure 7.11,

$$x_{\text{new}} = \begin{bmatrix} x_1^{(0)^T} & z^{(\kappa)^T} & y^{(\kappa)^T} & z^{(\kappa-1)^T} & y^{(\kappa-1)^T} & \dots & z^{(1)^T} & y^{(1)^T} \end{bmatrix}^T \text{ and } b_{\text{new}} = \begin{bmatrix} b_1^{(0)} \\ 0 \end{bmatrix},$$

The vectors $z^{(k)}$ and $y^{(k)}$ consists of all the local and multipole coefficients, respectively, at level $k$, i.e.,

$$z^{(k)^T} = \begin{bmatrix} z_1^{(k)} & z_2^{(k)} & \dots & z_{2^k}^{(k)} \end{bmatrix} \text{ and } y^{(k)^T} = \begin{bmatrix} y_1^{(k)} & y_2^{(k)} & \dots & y_{2^k}^{(k)} \end{bmatrix}.$$

Table 7.5: Ordering of equations

- The first set of equations contain the $x_i^{(\kappa)}$'s and the local coefficients at all levels, i.e., $z_i^{(k)}$'s, where $k \in \{1, 2, \ldots, \kappa\}$ and $i \in \{1, 2, \ldots, 2^k\}$, i.e., the local to particle tranfer.
- This is followed by the equations capturing the relation between $x_i^{(\kappa)}$'s and the multipole coefficients, $y_i^{(k)}$, where $i \in \{1, 2, \ldots, 2^k\}$, i.e, the particle to multipole tranfer. This is then followed by the equation capturing the relation between
- For level $k = \kappa \to 2$
  - The local coefficients, $z_i^{(k)}$ and the multipole coefficients, $y_i^{(k)}$, and the local coefficients $z_j^{(k-1)}$, where $i \in \{1, 2, \ldots, 2^k\}$ and $j \in \{1, 2, \ldots, 2^{k-1}\}$, i.e., the multipole to local and local to local transfers at level $k$.
  - The multipole coefficients, $y_i^{(k)}$ and the multipole coefficients, $y_i^{(k-1)}$, i.e., the multipole to multipole transfers at level $k$.
- The local coefficients $z_i^{(1)}$ at level 1 and the multipole coefficients at level 1, where $i \in \{1, 2\}$, i.e., the multipole to local transfers at level 1.

The elimination is explained pictorially through Figures 7.11 – 7.19. As with the sparse HODLR matrix, the elimination proceeds from the top-left and proceeds along the south-east direction, as shown in Figures 7.11– 7.19.

Note that as the elimination proceeds the self-similar sparsity structure and the fill-in is preserved. There is no additional fill-in as we proceed through the elimination process as shown in Figure 7.11 through Figure 7.19. From the set of Figures 7.11– 7.19, the computational cost can be obtained through the recurrence (7.7), where $C(N;k,r)$ denotes the computational cost for the factorization of a $k$-level $N \times N$ HSS matrix and $r$ is the rank of the off-diagonal blocks at each level.

$$C(N;k,r) = C\left(\frac{N}{2};k-1,r\right) + \mathcal{O}(Nr^2) \tag{7.7}$$

Solving the above recurence with $\kappa = \log_2\left(\frac{N}{2r}\right)$ gives us

$$C(N;\kappa,r) = \mathcal{O}\left(Nr^2\right) + \mathcal{O}\left(\frac{N}{2}r^2\right) + \mathcal{O}\left(\frac{N}{4}r^2\right) + \cdots = \mathcal{O}\left(\sum_{k=1}^{\kappa}\frac{N}{2^{\kappa-k}}r^2\right) = \mathcal{O}(r^2N)$$

$$
\begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \\ b_4^{(2)} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \underbrace{\begin{bmatrix}
K_1^{(2)} & 0 & 0 & 0 & U_1^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & K_2^{(2)} & 0 & 0 & 0 & U_2^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & K_3^{(2)} & 0 & 0 & 0 & U_3^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & K_4^{(2)} & 0 & 0 & 0 & U_4^{(2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
V_1^{(2)^T} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & V_2^{(2)^T} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & V_3^{(2)^T} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & V_4^{(2)^T} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & K_{1,2} & 0 & 0 & S_1^{(2)} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & K_{2,1} & 0 & 0 & 0 & S_2^{(2)} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & 0 & 0 & K_{3,4} & 0 & S_3^{(2)} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & K_{4,3} & 0 & 0 & S_4^{(2)} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_1^{(2)^T} & R_2^{(2)^T} & 0 & 0 & 0 & 0 & -I & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & R_3^{(2)^T} & R_4^{(2)^T} & 0 & 0 & 0 & -I \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -I & 0 & 0 & K_{1,2}^{(1)} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -I & K_{2,1}^{(1)} & 0
\end{bmatrix}}_{\text{The matrix } A} \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \\ x_4^{(2)} \\ z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \\ z_4^{(2)} \\ y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \\ y_4^{(2)} \\ z_1^{(1)} \\ z_2^{(1)} \\ y_1^{(1)} \\ y_2^{(1)} \end{bmatrix} \tag{7.8}
$$

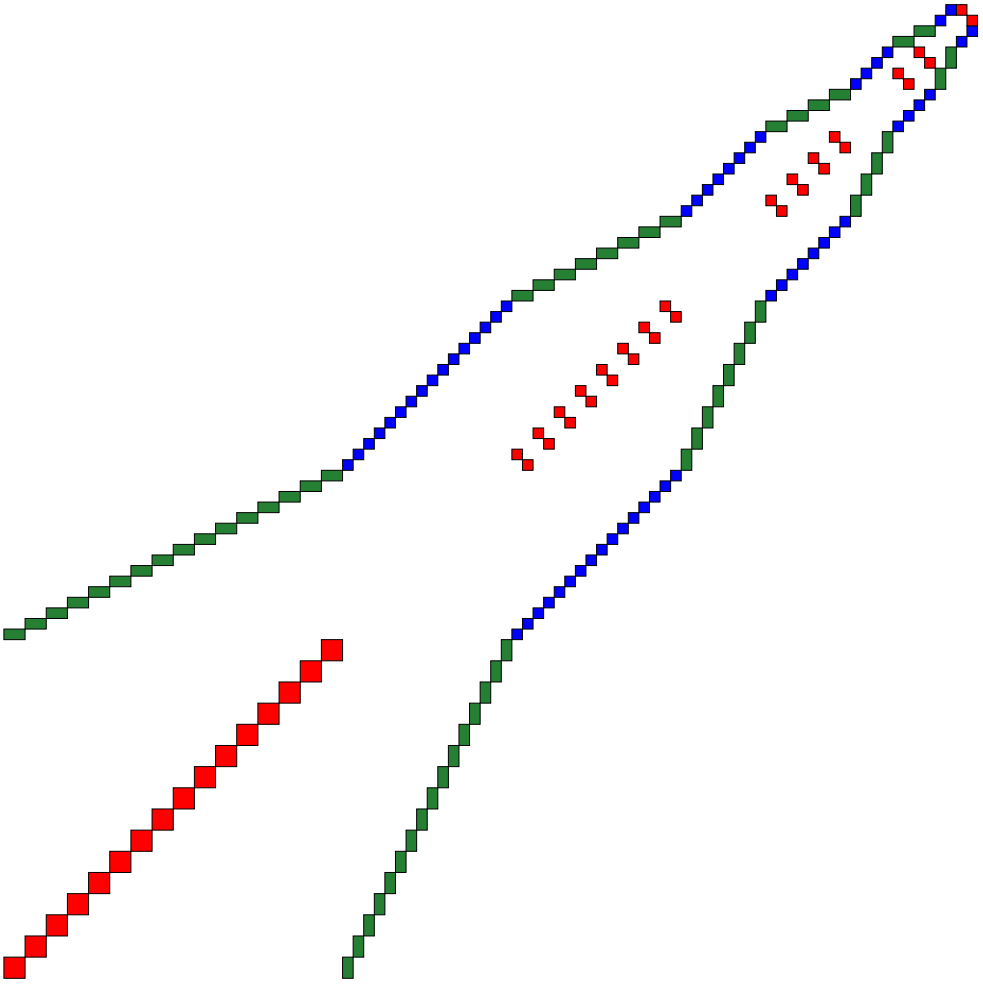


Figure 7.11: The sparsified matrix at the lowest level

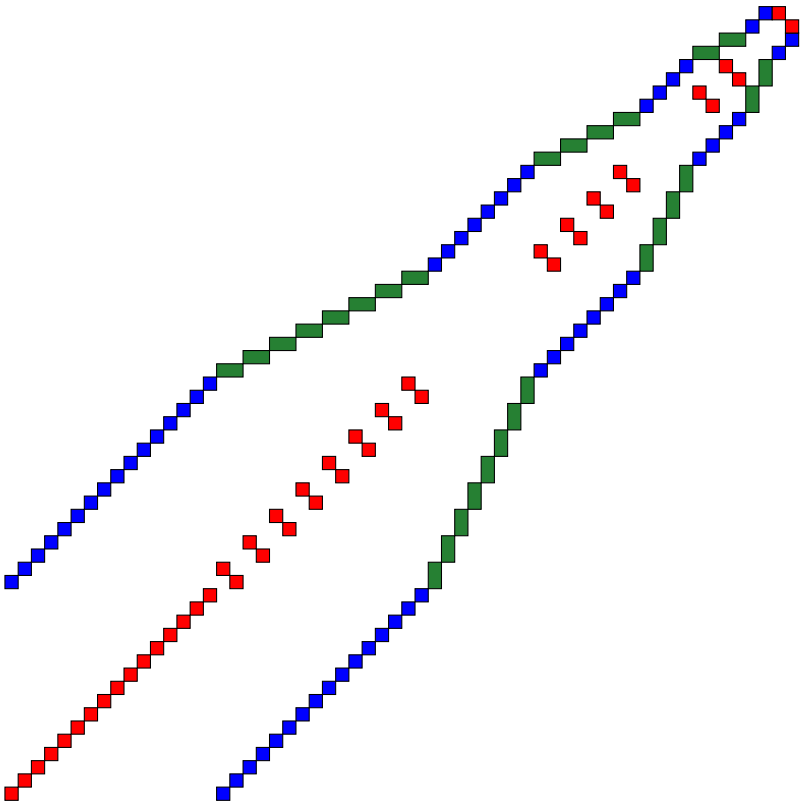


Figure 7.12: The sparsified matrix: After the first set of eliminations.

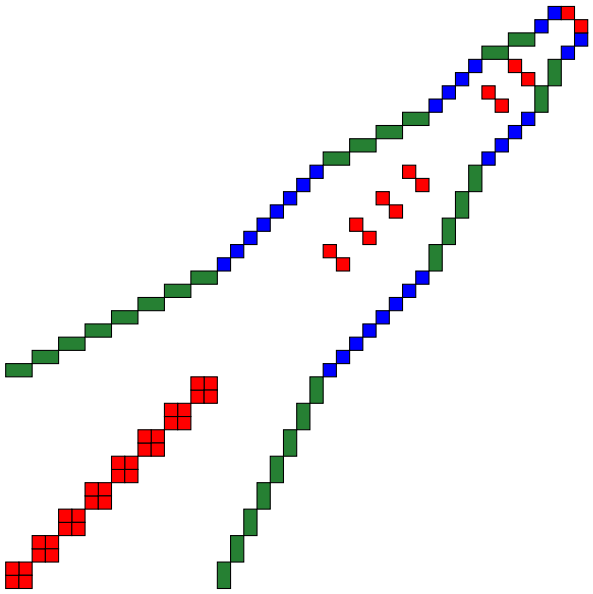


Figure 7.13: The sparsified matrix: After the second set of eliminations.

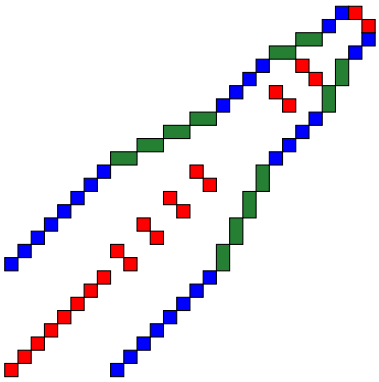


Figure 7.14: The sparsified matrix: After the third set of eliminations.

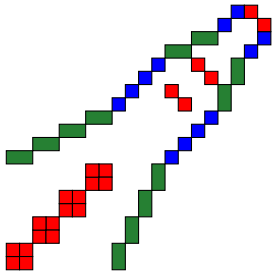


Figure 7.15: The sparsified matrix: After the fourth set of eliminations.

Figure 7.16: The sparsified matrix: After the fifth set of eliminations.



Figure 7.17: The sparsified matrix: After the sixth set of eliminations.



Figure 7.18: The sparsified matrix: After the seventh set of eliminations.



Figure 7.19: The sparsified matrix: After the eigth set of eliminations.

### 7.2.4 Numerical benchmark

We present a numerical benchmark for the above algorithm on HSS matrices. We consider a set of points distributed uniformly at random in the interval $[-1, 1]$. The matrix of interaction is given by the kernel $K(x_i, x_j) = \exp(-|x_i - x_j|)$. The rank of the off-diagonal blocks is taken as 8 at all levels.

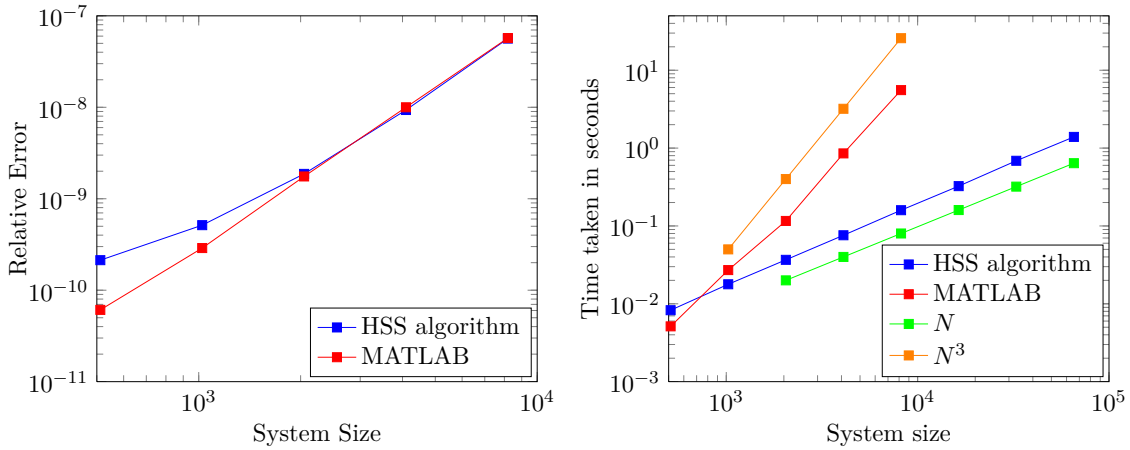



Figure 7.20: Left: Comparison of relative error versus system size; Right: Comparison of time taken versus system size.

## 7.3 Conclusions

In this chapter, we presented a new algorithm based on extended sparsification approach for constructing fast direct solvers. The algorithm was presented in the context of HSS and HODLR matrices. The advantage of this approach as opposed to the factorization based fast direct solvers is that this can be extended to the more general class of $\mathcal{H}$ and $\mathcal{H}^2$ matrices, though this is slightly non-trivial. To achieve linear complexity for $\mathcal{H}$ and $\mathcal{H}^2$ matrices, *it is to be emphasized that, when performing the elimination of unknowns to solve the extended sparse linear system, the interaction between the unknowns corresponding to the well-separated clusters at all stages in the elimination process can be efficiently compressed as low-rank. This implies, after an appropriate ordering of equations and unknowns, while we perform the elimination, the fill-in that occurs in the elimination process corresponding to well-separated clusters can be compressed and efficiently represented as a low-rank matrix.*

# Chapter 8

# Summary and Conclusions

The main contribution of the thesis is the pleasure I derived from working on these problems and algorithms. The other contributions of the thesis include a new set of fast direct solvers for hierarchical matrices with weak admissibility criteria, addressing challenging computational issues in the context of large-scale linear inverse problems and a novel computationally efficient Kalman filtering algorithm. All the fast algorithms discussed in this thesis rely on exploiting the underlying symmetry and hierarchical nature of problems arising from physical applications.

The idea of extended sparsification in constructing fast direct solvers for hierarchical matrices with weak admissibility criteria can also be extended to the more general class of hierarchical matrices, i.e., $\mathcal{H}$ and $\mathcal{H}^2$ matrices. However, there are a few additional operations involved. For instance, as we proceed through with the elimination, fill-ins occur at blocks, where ideally we do not want fill-ins. However, these fill-ins are low-rank, i.e., can be compressed again. This is because these fill-ins occur at blocks corresponding to well-separated clusters. The extended sparse matrix approach of viewing hierarchical matrices is a promising approach and could probably be extended to perform all $\mathcal{H}$-matrix operations in (almost) linear complexity. In general, hierarchical matrices open up new paths to many of the dense matrix algebra problems, which were inaccessible before. More over, the sparse matrix approach based on extended variables provides a:

- New conceptual framework to view hierarchical matrices.
- Interesting interplay between low-rank sparsity and zero fill-in sparsity.
- Template for different dense hierarchical systems

- Ideal platform for developing linear algebra packages.
- More scalable approach on parallel machines than recursive algorithms.

The key take home ideas from this dissertation are the following: Low-rank, Hierarchical structure, and extended sparsity. The motivation for the research in the dissertation is the following mantra:

"A *good computation* is one that does the *least computation* to obtain the right answer."

# Bibliography


[1] J.B. Ajo-Franklin. Optimal experiment design for time-lapse traveltime tomography. *Geophysics*, 74(4):Q27–Q40, 2009.

[2] Sivaram Ambikasaran and Eric Darve. An $\mathcal{O}(n \log n)$ fast direct solver for partial hierarchically semi-separable matrices. *Journal of Scientific Computing*, pages 1–25, 2013.

[3] I. Andrianakis and P.G. Challenor. The effect of the nugget on gaussian process emulators of computer models. *Computational Statistics & Data Analysis*, 2012.

[4] W.E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quart. Appl. Math*, 9(1):17–29, 1951.

[5] J. Barnes and P. Hut. A hierarchical $\mathcal{O}(N \log N)$ force-calculation algorithm. *Nature*, 324(4):446–449, 1986.

[6] B. Baxter. The interpolation theory of radial basis functions. *arXiv preprint arXiv:1006.2443*, 2010.

[7] R.K. Beatson and Leslie Greengard. A short course on fast multipole methods. *Wavelets, multilevel methods and elliptic PDEs*, pages 1–37, 1997.

[8] R.K. Beatson and G.N. Newsam. Fast evaluation of radial basis functions: I. *Computers & Mathematics with Applications*, 24(12):7–19, 1992.

[9] R.K. Beatson, J.B. Cherrie, and C.T. Mouat. Fast fitting of radial basis functions: Methods based on preconditioned GMRES iteration. *Advances in Computational Mathematics*, 11(2):253–270, 1999.

[10] M. Bebendorf. Approximation of boundary element matrices. *Numerische Mathematik*, 86(4):565–589, 2000.

[11] M. Bebendorf and S. Rjasanow. Adaptive low-rank approximation of collocation matrices. *Computing*, 70(1):1–24, 2003. ISSN 0010-485X.

[12] S.D. Billings, R.K. Beatson, and G.N. Newsam. Interpolation of geophysical data using continuous global surfaces. *Geophysics*, 67(6):1810, 2002.

[13] Christian H Bischof and Per Christian Hansen. Structure-preserving and rank-revealing qr-factorizations. *SIAM Journal on Scientific and Statistical Computing*, 12(6):1332–1350, 1991.

[14] O.N. Bjørnstad and W. Falck. Nonparametric spatial covariance functions: estimation and testing. *Environmental and Ecological Statistics*, 8(1):53–70, 2001.

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

[16] M.D. Buhmann. *Radial basis functions: theory and implementations*, volume 12. Cambridge University Press, 2003.

[17] C Sidney Burrus, Ramesh A Gopinath, and Haitao Guo. *Introduction to Wavelets and Wavelet Transforms: A Primer.* Prentice-Hall, Inc., 1998.

[18] Tony F Chan. On the existence and computation of $lu$-factorizations with small pivots. *Mathematics of computation*, 42(166):535–547, 1984.

[19] Tony F Chan. Rank revealing QR factorizations. *Linear Algebra and Its Applications*, 88:67–82, 1987.

[20] Shivkumar Chandrasekaran and Ilse CF Ipsen. On rank-revealing factorisations. *SIAM Journal on Matrix Analysis and Applications*, 15(2):592–622, 1994.

[21] Shivkumar Chandrasekaran, P Dewilde, Ming Gu, W Lyons, and T Pals. A fast solver for hss representations via sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 29(1):67–81, 2006.

[22] Shivkumar Chandrasekaran, P. Dewilde, Ming Gu, T. Pals, X. Sun, A. van der Veen, and D. White. Some fast algorithms for sequentially semiseparable representations. *SIAM Journal on Matrix Analysis and Applications*, 27(2):341, 2006.

[23] Shivkumar Chandrasekaran, Ming Gu, and T. Pals. A fast ULV decomposition solver for hierarchically semiseparable representations. *SIAM Journal on Matrix Analysis and Applications*, 28(3):603–622, 2006.

[24] Ke. Chen. An analysis of sparse approximate inverse preconditioners for boundary integral equations. *SIAM Journal on Matrix Analysis and Applications*, 22(4):1058–1078, 2001.

[25] H. Cheng, Leslie Greengard, and Vladimir Rokhlin. A fast adaptive multipole algorithm in three dimensions. *Journal of Computational Physics*, 155(2):468–498, 1999.

[26] H. Cheng, Z. Gimbutas, Per-Gunnar Martinsson, and Vladimir Rokhlin. On the compression of low-rank matrices. *SIAM Journal on Scientific Computing*, 26(4): 1389–1404, 2005.

[27] R. Coifman, Vladimir Rokhlin, and S. Wandzura. The fast multipole method for the wave equation: A pedestrian prescription. *Antennas and Propagation Magazine, IEEE*, 35(3):7–12, 1993.

[28] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comput*, 19(90):297–301, 1965.

[29] D. Cornford, L. Csató, and M. Opper. Sequential, bayesian geostatistics: a principled method for large data sets. *Geographical Analysis*, 37(2):183–199, 2005.

[30] T.M. Daley, R.D. Solbau, J.B. Ajo-Franklin, and S.M. Benson. Continuous active-source seismic monitoring of formula injection in a brine aquifer. *Geophysics*, 72(5): A57, 2007.

[31] T.M. Daley, J.B. Ajo-Franklin, and C.M. Doughty. Integration of crosswell CASSM (Continuous Active Source Seismic Monitoring) and flow modeling for imaging of a CO2 plume in a brine aquifer. In *2008 SEG Annual Meeting*, 2008.

[32] E. Darve. The fast multipole method: numerical implementation. *Journal of Computational Physics*, 160(1):195–240, 2000.

[33] Eric Darve. The fast multipole method i: Error analysis and asymptotic complexity. *SIAM Journal on Numerical Analysis*, 38(1):98–128, 2000.

[34] G.J. Davis and M.D. Morris. Six factors which affect the condition number of matrices associated with kriging. *Mathematical geology*, 29(5):669–683, 1997.

[35] T.A. Davis. *Direct methods for sparse linear systems*, volume 2. Society for Industrial Mathematics, 2006.

[36] A. De Boer, MS Van der Schoot, and H. Bijl. Mesh deformation based on radial basis function interpolation. *Computers & Structures*, 85(11-14):784–795, 2007.

[37] CR Dietrich and GN Newsam. Efficient generation of conditional simulations by Chebyshev matrix polynomial approximations to the symmetric square root of the covariance matrix. *Mathematical geology*, 27(2):207–228, 1995.

[38] C. Doughty, B.M. Freifeld, and R.C. Trautz. Site characterization for $CO_2$ geologic storage and vice versa: the Frio brine pilot, Texas, USA as a case study. *Environmental Geology*, 54(8):1635–1656, 2008.

[39] W. Fong and E. Darve. The black-box fast multipole method. *Journal of Computational Physics*, 228(23):8712–8725, 2009.

[40] Leslie V Foster. Rank and null space calculations using matrix decomposition without column interchanges. *Linear Algebra and its Applications*, 74:47–71, 1986.

[41] R.W. Freund. A transpose-free quasi-minimal residual algorithm for non-hermitian linear systems. *SIAM Journal on Scientific Computing*, 14:470, 1993.

[42] R.W. Freund and N.M. Nachtigal. QMR: a quasi-minimal residual method for non-hermitian linear systems. *Numerische Mathematik*, 60(1):315–339, 1991.

[43] A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. *Journal of the ACM (JACM)*, 51(6):1025–1041, 2004.

[44] J. Fritz, I. Neuweiler, and W. Nowak. Application of FFT-based algorithms for large-scale universal kriging problems. *Mathematical Geosciences*, 41(5):509–533, 2009.

[45] A. Gillman, P. Young, and Per-Gunnar Martinsson. A direct solver with $\mathcal{O}(N)$ complexity for integral equations on one-dimensional domains. *arXiv preprint arXiv:1105.5372*, 2011.

[46] G Golub. Numerical methods for solving linear least squares problems. *Numerische Mathematik*, 7(3):206–216, 1965.

[47] G.H. Golub and C.F. Van Loan. *Matrix computations*, volume 3. Johns Hopkins Univ Press, 1996.

[48] S.A. Goreinov, EE Tyrtyshnikov, and NL Zamarashkin. A theory of pseudoskeleton approximations. *Linear Algebra and its Applications*, 261(1-3):1–21, 1997.

[49] Lars Grasedyck and Wolfgang Hackbusch. Construction and arithmetics of h-matrices. *Computing*, 70(4):295–334, 2003.

[50] Leslie Greengard. *The rapid evaluation of potential fields in particle systems*, volume 1987. the MIT Press, 1988.

[51] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348, 1987.

[52] Leslie Greengard and Vladimir Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta Numerica*, 6(1):229–269, 1997.

[53] Leslie Greengard, D. Gueyffier, Per-Gunnar Martinsson, and Vladimir Rokhlin. Fast direct solvers for integral equations in complex three-dimensional domains. *Acta Numerica*, 18(1):243–275, 2009.

[54] Ming Gu and S.C. Eisenstat. Efficient algorithms for computing a strong rank-revealing QR factorization. *Society*, 17(4):848–869, 1996.

[55] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[56] N.A. Gumerov and R. Duraiswami. Fast radial basis function interpolation via preconditioned Krylov iteration. *SIAM Journal on Scientific Computing*, 29(5):1876–1899, 2007.

[57] W. Hackbusch and S. Börm. Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. *Computing*, 69(1):1–35, 2002.

[58] W. Hackbusch and Z.P. Nowak. On the fast matrix multiplication in the boundary element method by panel clustering. *Numerische Mathematik*, 54(4):463–491, 1989.

[59] Wolfgang Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part i: Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

[60] Wolfgang Hackbusch and Boris N Khoromskij. A sparse $\mathcal{H}$-matrix arithmetic. *Computing*, 64(1):21–47, 2000.

[61] Wolfgang Hackbusch and Boris N Khoromskij. A sparse $\mathcal{H}$-matrix arithmetic: general complexity estimates. *Journal of Computational and Applied Mathematics*, 125(1): 479–501, 2000.

[62] Wolfgang Hackbusch, Boris Khoromskij, and Stefan A Sauter. On $\mathcal{H}^2$-matrices. In Hans-Joachim Bungartz, RonaldH.W. Hoppe, and Christoph Zenger, editors, *Lectures on Applied Mathematics*, pages 9–29. Springer Berlin Heidelberg, 2000. ISBN 978-3-642-64094-0.

[63] W.W. Hager. Updating the inverse of a matrix. *SIAM review*, pages 221–239, 1989.

[64] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

[65] Kenneth L Ho and Leslie Greengard. A fast direct solver for structured linear systems by recursive skeletonization. *SIAM Journal on Scientific Computing*, 34(5):2507–2532, 2012.

[66] YP Hong and CT Pan. Rank-revealing QR factorizations and the singular value decomposition. *Mathematics of Computation*, 58(197):213–232, 1992.

[67] Tsung-Min Hwang, Wen-Wei Lin, and Eugene K Yang. Rank revealing lu factorizations. *Linear algebra and its applications*, 175:115–141, 1992.

[68] Tsung-Min Hwang, Wen-Wei Lin, and Daniel Pierce. Improved bound for rank revealing lu factorizations. *Linear algebra and its applications*, 261(1):173–186, 1997.

[69] William Kahan. Numerical linear algebra. *Canadian Math. Bull*, 9(6):757–801, 1966.

[70] Ilkka Karasalo. A criterion for truncation of theqr-decomposition algorithm for the singular linear least squares problem. *BIT Numerical Mathematics*, 14(2):156–166, 1974.

[71] P. K. Kitanidis. Statistical estimation of polynomial generalized covariance functions and hydrologic applications. *Water Resources Research*, 19(4):909–921, 1983.

[72] P. K. Kitanidis. Generalized covariance functions in estimation. *Mathematical Geology*, 25(5):525–540, 1993.

[73] P. K. Kitanidis. Quasi-linear geostatistical theory for inversing. *Water Resources Research*, 31(10):2411–2419, 1995.

[74] P. K. Kitanidis. On the geostatistical approach to the inverse problem. *Advances in Water Resources*, 19(6):333–342, 1996.

[75] P. K. Kitanidis. Generalized covariance functions associated with the Laplace equation and their use in interpolation and inverse problems. *Water Resources Research*, 35(5): 1361–1367, 1999.

[76] P. K. Kitanidis. On stochastic inverse modeling. *Geophysical Monograph-American Geophysical Union*, 171:19, 2007.

[77] P. K. Kitanidis and E. G. Vomvoris. A geostatistical approach to the inverse problem in groundwater modeling (steady state) and one-dimensional simulations. *Water Resources Research*, 19(3):677–690, 1983.

[78] Peter K. Kitanidis. *Introduction to geostatistics: applications to hydrogeology*. Cambridge Univ Pr, 1997.

[79] Wai Yip Kong, James Bremer, and Vladimir Rokhlin. An adaptive fast direct solver for boundary integral equations in two dimensions. *Applied and Computational Harmonic Analysis*, 31(3):346–369, 2011.

[80] W. Li and O. A. Cirpka. Efficient geostatistical inverse methods for structured and unstructured grids. *Water Resources Research*, 42(6):W06402, 2006.

[81] E. Liberty, F. Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and M. Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167, 2007.

[82] X. Liu and P. K Kitanidis. Large-scale inverse modeling with an application in hydraulic tomography. *Water Resources Research*, 47(2):W02501, 2011.

[83] X. Liu, WA Illman, AJ Craig, J. Zhu, and TCJ Yeh. Laboratory sandbox validation of transient hydraulic tomography. *Water Resources Research*, 43(5):W05404, 2007.

[84] S.G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(7): 674–693, 1989.

[85] S.G. Mallat. *A wavelet tour of signal processing*. Academic Press, 1999.

[86] Thomas A Manteuffel. An interval analysis approach to rank determination in linear least squares problems. *SIAM Journal on Scientific and Statistical Computing*, 2(3): 335–348, 1981.

[87] Per-Gunnar Martinsson. A fast direct solver for a class of elliptic partial differential equations. *Journal of Scientific Computing*, 38(3):316–330, 2009.

[88] Per-Gunnar Martinsson and Vladimir Rokhlin. A fast direct solver for boundary integral equations in two dimensions. *Journal of Computational Physics*, 205(1):1–23, 2005.

[89] M. Messner, M. Schanz, and E. Darve. Fast directional multilevel summation for oscillatory kernels based on Chebyshev interpolation. *Journal of Computational Physics*, 2011.

[90] L. Miranian and Ming Gu. Strong rank revealing LU factorizations. *Linear Algebra and its Applications*, 367:1–16, 2003.

[91] Esmond Ng. A scheme for handling rank-deficiency in the solution of sparse linear least squares problems. *SIAM journal on scientific and statistical computing*, 12(5): 1173–1183, 1991.

[92] N. Nishimura. Fast multipole accelerated boundary integral equation methods. *Applied Mechanics Reviews*, 55(4):299–324, 2002.

[93] W. Nowak and O. A. Cirpka. Geostatistical inference of hydraulic conductivity and dispersivities from hydraulic heads and tracer data. *Water Resources Research*, 42(8): 8416, 2006.

[94] W. Nowak, S. Tenkleve, and O.A. Cirpka. Efficient computation of linearized cross-covariance and auto-covariance matrices of interdependent quantities. *Mathematical geology*, 35(1):53–66, 2003.

[95] R. J O'Dowd. Conditioning of coefficient matrices of ordinary kriging. *Mathematical Geology*, 23(5):721–739, 1991.

[96] Christopher C Paige and Michael A Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, 1975.

[97] C-T Pan. On the existence and computation of rank-revealing LU factorizations. *Linear Algebra and its Applications*, 316(1):199–222, 2000.

[98] Ching-Tsuan Pan and Ping Tak Peter Tang. Bounds on singular values revealed by qr factorizations. *BIT Numerical Mathematics*, 39(4):740–756, 1999.

[99] D. Pollock and O.A. Cirpka. Fully coupled hydrogeophysical inversion of synthetic salt tracer experiments. *Water Resources Research*, 46(7):W07501, 2010.

[100] K. Pruess. *ECO2N: A TOUGH2 fluid property module for mixtures of water, NaCl, and CO2*. Lawrence Berkeley National Laboratory, 2005.

[101] K. Pruess, C. Oldenburg, and G. Moridis. TOUGH2 user's guide, version 2.0, 1999.

[102] S. Rjasanow. Adaptive cross approximation of dense matrices. *IABEM 2002, International Association for Boundary Element Methods*, 2002.

[103] S. Rjasanow and O. Steinbach. *The fast solution of boundary integral equations. Mathematical and Analytical Techniques with Applications to Engineering*. Springer, New York, 2007.

[104] Vladimir Rokhlin. Rapid solution of integral equations of classical potential theory. *Journal of Computational Physics*, 60(2):187–207, 1985.

[105] Y. Saad. *Iterative methods for sparse linear systems*, volume 20. PWS publishing company Boston, 1996.

[106] Y. Saad and M.H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[107] Robert Schaback. Creating surfaces from scattered data using radial basis functions. *Mathematical methods for curves and surfaces*, pages 477–496, 1995.

[108] P.G. Schmitz and L. Ying. A fast direct solver for elliptic problems on general meshes in 2D. *Journal of Computational Physics*, 2011.

[109] P.G. Schmitz and L. Ying. A fast direct solver for elliptic problems on Cartesian meshes in 3D. In review, 2012.

[110] TH Starks and JH Fang. On the estimation of the generalized covariance function. *Mathematical Geology*, 14(1):57–64, 1982.

[111] Henk A Van der Vorst. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing*, 13(2):631–644, 1992.

[112] R. Vandebril, M.V. Barel, G. Golub, and N. Mastronardi. A bibliography on semiseparable matrices. *Calcolo*, 42(3):249–270, 2005.

[113] Stephen A Vavasis. Preconditioning for boundary integral equations. *SIAM journal on matrix analysis and applications*, 13(3):905–925, 1992.

[114] JG Wang and GR Liu. A point interpolation meshless method based on radial basis functions. *International Journal for Numerical Methods in Engineering*, 54(11): 1623–1648, 2002.

[115] JE White. Computed seismic speeds and attenuation in rocks with partial gas saturation. *Geophysics*, 40(2):224–232, 1975.

[116] Max A Woodbury. Inverting modified matrices. Statistical Research Group, Memo. Rep. no. 42, Princeton University, 1950.

[117] F. Woolfe, E. Liberty, Vladimir Rokhlin, and M. Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25 (3):335–366, 2008.

[118] Z. Wu and R. Schaback. Local error estimates for radial basis function interpolation of scattered data. *IMA Journal of Numerical Analysis*, 13(1):13–27, 1993.

[119] J. Xia, Shivkumar Chandrasekaran, Ming Gu, and X.S. Li. Fast algorithms for hierarchically semiseparable matrices. *Numerical Linear Algebra with Applications*, 17 (6):953–976, 2010.

[120] L. Ying, G. Biros, and D. Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics*, 196(2):591–626, 2004.

[121] D.L. Zimmerman. Computationally efficient restricted maximum likelihood estimation of generalized covariance functions. *Mathematical Geology*, 21(7):655–672, 1989.

[122] D.L. Zimmerman. Computationally exploitable structure of covariance matrices and generalized covariance matrices in spatial models. *Journal of Statistical Computation and Simulation*, 32(1-2):1–15, 1989.