An $\mathcal{O}(N \log N)$ fast direct solver for partial Hierarchically Semi-Separable matrices *

With application to radial basis function interpolation

Sivaram Ambikasaran† Eric Darve*‡

Received: date / Accepted: date

This article describes a fast direct solver (i.e., not iterative) for partial hierarchically semi-separable systems. This solver requires a storage of $\mathcal{O}(N \log N)$ and has a computational complexity of $\mathcal{O}(N \log N)$ arithmetic operations. The numerical benchmarks presented illustrate the method in the context of interpolation using radial basis functions. The key ingredients behind this fast solver are recursion, efficient low-rank factorization using Chebyshev interpolation, and the Sherman-Morrison-Woodbury formula. The algorithm and the analysis are worked out in detail. The performance of the algorithm is illustrated for a variety of radial basis functions and target accuracies.

Keywords. Fast direct solver Numerical linear algebra Partial hierarchically semi-separable representation Hierarchical matrix Radial basis function

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 [2], Conjugate Gradient [41], GMRES [53], MINRES [50], Biconjugate Gradient Stabilized Method [59], QMR [24], TFQMR [23], 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) [6, 16, 48], the Barnes-Hut algorithm [3], panel clustering [39], 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 [5, 14, 17, 18, 22, 30, 31, 65]. These fast iterative solvers accelerate the solution procedure and are able to solve (with some prescribed error tolerance ε) a system in linear or almost linear time.

In this paper 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).

*The authors would like to thank the "Army High Performance Computing Research Center" (AHPCRC) and "The Global Climate and Energy Project" (GCEP) at Stanford for supporting the project.

†Institute for Computational and Mathematical Engineering, Stanford University

‡Department of Mechanical Engineering, Stanford University

- For iterative solvers to be efficient, choosing a good preconditioner [7, 13, 35, 58] 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. [9, 29, 36–38] 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. [11, 12, 64] as a generalization of semi-separable matrices. A semi-separable matrix [57] 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 [11, 12, 64] for more details regarding the HSS representation.

In the last few years, Greengard [32], Rokhlin [42], Martinsson [44, 45], Lexing Ying [55, 56] 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 3.3.2. Linear systems arising from the discretization of boundary integral equations of potential theory [42, 45] in two dimensions, interpolation by radial basis function along a curve [6, 35], 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. [26], Martinsson [44], Martinsson and Rokhlin [45], and Kong et al. [42].

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 [26, 42, 44, 45] — the method in its current form still applies but no longer scales as $\mathcal{O}(N \log N)$. For example, as presented in this paper, 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 $O(N^{1/2})$.

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 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 [27, 40, 61]. 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 5.

We now discuss the algorithm presented in this paper 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. [12] 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. [22]. The solver described in the present manuscript constructs a different one-sided factorization of the matrix. Further, this article works with a p-HSS matrix, which is a superset of HSS matrices, as will be explained in detail in section 3.3.

The work by Rokhlin and Martinsson [45] 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 [12], 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 [45] builds a compressed factorization of the inverse matrix.

A recently published work, while this manuscript was under preparation, by Kong et al. [42] 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. [42] 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 the paper 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 4 motivates and provides an overview of the algorithm, and discusses the computational complexity of the algorithm. Section 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 paper by highlighting the capabilities of this solver.

3 Preliminary ideas

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

3.1 Sherman-Morrison-Woodbury formula

The key ingredient in our algorithm is the Sherman-Morrison-Woodbury formula [40, 61], 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

$$\left(I + UV^T\right) x = b \tag{1}$$

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

$$x = b - U\left(I + V^T U\right)^{-1} V^T b \tag{2}$$

The computational cost for solving (1) using (2) is $\mathcal{O}(prN)$. We refer the readers to [40, 61] for the derivation of the above result and the computational cost, though the direct proof takes only a few steps:

$$\begin{aligned}
\left(I + UV^T\right) x &= \left(I + UV^T\right) b - \left(I + UV^T\right) U \left(I + V^T U\right)^{-1} V^T b \quad \text{[Eq. (2)]} \\
&= \left(I + UV^T\right) b - U \left(I + V^T U\right) \left(I + V^T U\right)^{-1} V^T b \quad \text{[Refactor the second term]} \\
&= \left(I + UV^T\right) b - UV^T b = b
\end{aligned}$$

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 [27]. 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 [15, 25, 33, 47] on constructing fast approximate low-rank factorizations for matrices. Techniques like adaptive cross approximation [52] (ACA), pseudo-skeletal approximations, [28] interpolatory decomposition, [22] randomized algorithms [25, 43, 62], rank-revealing LU [47, 51] and QR [33] 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.

In the numerical illustrations discussed in section 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 [22, 46]. 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.

3.2.1 Low-rank using interpolation

Consider a function $g(x)$ on the closed interval $[-1,1]$. A p-point interpolant, $P_{p-1}(x)$, that approximates $g(x)$ can be written as

$$P_{p-1}(x) = \sum_{k=1}^{p} g(\bar{x}_k) w_k(x) \tag{3}$$

where $\bar{x}_k$'s are the p interpolation nodes and $w_k(x)$ is the interpolating function corresponding to the node $\bar{x}_k$. A low-rank approximation for the kernel $K(x,y)$ is constructed as a p-point interpolant as shown below.

$$K(x,y) \approx \sum_{k=1}^{p} w_k(x) K(\bar{x}_k, y) \tag{4}$$

Note that this p-point interpolant is obtained by having p interpolation nodes for x only (we could interpolate along y as well by having interpolation nodes along y). This gives us a low-rank representation for the kernel $K(x,y)$. Although any interpolation scheme can be used to construct a low-rank approximation as described above, the Chebyshev polynomials will serve as the interpolation basis and their roots will serve as the interpolation nodes. We will briefly recall some properties of the Chebyshev polynomials before proceeding further.

The Chebyshev polynomial of the first kind of degree p, denoted by $T_p(x)$, is defined as

$$T_p(x) = \cos(p \arccos(x)) \tag{5}$$

The domain of $T_p(x)$ is the closed interval $[-1,1]$. $T_p(x)$ has p roots located at

$$\bar{x}_k = \cos\left(\frac{(2k-1)\pi}{2p}\right), k \in \{1,2,\ldots,p\}. \tag{6}$$

The set of roots $\{\bar{x}_k\}$ are called Chebyshev nodes. Using the Chebyshev nodes of $T_p(x)$ as the interpolation nodes, the low-rank approximation to $K(x,y)$ can be written as

$$K_{p-1}(x,y) = \sum_{k=0}^{p-1} T_k(x) c_k(y) \tag{7}$$

where

$$c_k(y) = \begin{cases} \frac{2}{p}\sum_{l=1}^{p} K(\bar{x}_l, y) T_k(\bar{x}_l) & \text{if } k > 0, \\ \frac{1}{p}\sum_{l=1}^{p} K(\bar{x}_l, y) & \text{if } k = 0. \end{cases}$$

By rearranging terms, we get

$$K_{p-1}(x,y) = \sum_{k=1}^{p} U(x,\bar{x}_k) K(\bar{x}_k, y) \tag{8}$$

where

$$U(x,\bar{x}_k) = \frac{1}{p} + \frac{2}{p} \sum_{l=1}^{p-1} T_l(x) T_l(\bar{x}_k) \tag{9}$$

Equation (8) gives us the desired low-rank representation for $K(x,y)$. If we let $K_{mat} \in \mathbb{R}^{M \times N}$ be the matrix $K_{mat}(i,k) = K(x_i, y_k)$, then equation (8) can be written as

$$K_{mat} = U_{mat} V_{mat}^T \tag{10}$$

where $U_{mat} \in \mathbb{R}^{M \times p}, V_{mat} \in \mathbb{R}^{N \times p}$ with $U_{mat}(i,j) = U(x_i, \bar{x}_j)$ and $V_{mat}(j,k) = K(\bar{x}_j, y_k)$. U_{mat} is termed the anterpolation matrix since this anterpolates the information from the cluster containing the x's to the interpolation nodes of the cluster. The matrix V_{mat} computes the kernel interaction between the interpolation nodes of the cluster containing the x's and the y's.

3.3 Hierarchical representations

We briefly discuss a couple of hierarchical representations, which we will be dealing with in this article.

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 (11).

$$K = \begin{bmatrix} K_1^{(1)} & U_1^{(1)} V_{1,2}^{(1)^T} \\ U_2^{(1)} V_{2,1}^{(1)^T} & K_2^{(1)} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} K_1^{(2)} & U_1^{(2)} V_{1,2}^{(2)^T} \\ U_2^{(2)} V_{2,1}^{(2)^T} & K_2^{(2)} \end{bmatrix} & U_1^{(1)} V_{1,2}^{(1)^T} \\ U_2^{(1)} V_{2,1}^{(1)^T} & \begin{bmatrix} K_3^{(2)} & U_3^{(2)} V_{3,4}^{(2)^T} \\ U_4^{(2)} V_{4,3}^{(2)^T} & K_4^{(2)} \end{bmatrix} \end{bmatrix} \tag{11}$$

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 κ-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

$$K_i^{(k)} = \begin{bmatrix} K_{2i-1}^{(k+1)} & U_{2i-1}^{(k+1)} V_{2i-1,2i}^{(k+1)^T} \\ U_{2i}^{(k+1)} V_{2i,2i-1}^{(k+1)^T} & K_{2i}^{(k+1)} \end{bmatrix} \tag{12}$$

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, κ, is $\lfloor \log_2(N/2p) \rfloor$. The construction of a κ-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 κ, 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 anterpolation 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 $\mathcal{O}(pN(1+\kappa))$.

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 [11, 12]. 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 4×4 block partitioning of a p-HSS matrix, K. The two-level p-HSS representation is as shown in equation (13).

$$K = \begin{bmatrix} K_1^{(1)} & U_1^{(1)} V_{1,2}^{(1)^T} \\ U_2^{(1)} V_{2,1}^{(1)^T} & K_2^{(1)} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} K_1^{(2)} & U_1^{(2)} V_{1,2}^{(2)^T} \\ U_2^{(2)} V_{2,1}^{(2)^T} & K_2^{(2)} \end{bmatrix} & \begin{bmatrix} U_1^{(2)} S_1^{(2)} \\ U_2^{(2)} S_2^{(2)} \end{bmatrix} V_{1,2}^{(1)^T} \\ \begin{bmatrix} U_3^{(2)} S_3^{(2)} \\ U_4^{(2)} S_4^{(2)} \end{bmatrix} V_{2,1}^{(1)^T} & \begin{bmatrix} K_3^{(2)} & U_3^{(2)} V_{3,4}^{(2)^T} \\ U_4^{(2)} V_{4,3}^{(2)^T} & K_4^{(2)} \end{bmatrix} \end{bmatrix} \tag{13}$$

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

$$U_1^{(1)} = \begin{bmatrix} U_1^{(2)} S_1^{(2)} \\ U_2^{(2)} S_2^{(2)} \end{bmatrix}, U_2^{(1)} = \begin{bmatrix} U_3^{(2)} S_3^{(2)} \\ U_4^{(2)} S_4^{(2)} \end{bmatrix}. \tag{14}$$

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 κ-level p-HSS representation, if we denote the i^{th} diagonal block at level k as $K_i^{(k)}$ (12), then $U_i^{(k)}$ is constructed from $U_{2i-1}^{(k+1)}$ and $U_{2i}^{(k+1)}$, i.e., we have

$$U_i^{(k)} = \begin{bmatrix} U_{2i-1}^{(k+1)} S_{2i-1}^{(k+1)} \\ U_{2i}^{(k+1)} S_{2i}^{(k+1)} \end{bmatrix} \tag{15}$$

The maximum number of levels, κ, is $\lfloor \log_2(N/2p) \rfloor$. The construction of a κ-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 κ, 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 κ, compute the anterpolation 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 anterpolation 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 $S_i^{(k)}(r, s) = U\left(\bar{x}_r^{(k)}, \bar{x}_s^{(k-1)}\right)$ 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, and $U\left(\bar{x}_r^{(k)}, \bar{x}_s^{(k-1)}\right)$ is given by equation (9).

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 [11, 12, 64] 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.

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.

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 (16):

$$K = K_\kappa K_{\kappa-1} K_{\kappa-2} \cdots K_1 K_0 \tag{16}$$

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 κ-level HODLR matrix, $K^{(\kappa)} \in \mathbb{R}^{N \times N}$ is presented in equation (17).

$$K^{(\kappa)} = \begin{bmatrix} \begin{bmatrix} K_1^{(\kappa)} & U_1^{(\kappa)} V_{1,2}^{(\kappa)^T} \\ U_2^{(\kappa)} V_{2,1}^{(\kappa)^T} & K_2^{(\kappa)} \end{bmatrix} & U_1^{(\kappa-1)} V_{1,2}^{(\kappa-1)^T} & \cdots & \cdots \\ U_2^{(\kappa-1)} V_{2,1}^{(\kappa-1)^T} & \begin{bmatrix} K_3^{(\kappa)} & U_3^{(\kappa)} V_{3,4}^{(\kappa)^T} \\ U_4^{(\kappa)} V_{4,3}^{(\kappa)^T} & K_4^{(\kappa)} \end{bmatrix} & \cdots & \cdots \\ \vdots & \vdots & \ddots & \cdots \\ \vdots & \vdots & \vdots & \begin{bmatrix} K_{2^\kappa}^{(\kappa)} & U_{2^\kappa-1}^{(\kappa)} V_{2^\kappa-1,2^\kappa}^{(\kappa)^T} \\ U_{2^\kappa}^{(\kappa)} V_{2^\kappa,2^\kappa-1}^{(\kappa)^T} & K_{2^\kappa}^{(\kappa)} \end{bmatrix} \end{bmatrix} \tag{17}$$

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 κ-level p-HSS matrix not only has the structure described in equation (17) but also has the additional structure mentioned in equation (15).

The first step in the algorithm is to factor the block diagonal matrix shown in equation (18).

$$K_\kappa = \begin{bmatrix} K_1^{(\kappa)} & 0 & 0 & 0 & \cdots & \cdots \\ 0 & K_2^{(\kappa)} & 0 & 0 & \cdots & \cdots \\ 0 & 0 & K_3^{(\kappa)} & 0 & \cdots & \cdots \\ 0 & 0 & 0 & K_4^{(\kappa)} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & K_{2^\kappa}^{(\kappa)} \end{bmatrix} \tag{18}$$

Here is the important difference between the factorization of a HODLR and a p-HSS matrix. For the HODLR structure, when we factor K_κ, $U_j^{(k)}$ 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* (15).

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_κ, 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 κp 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_κ, 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_κ, we get that $K^{(\kappa)} = K_\kappa\, K^{(\kappa-1)}$, where $K^{(\kappa-1)}$ is of the form in equation (19).

$$K^{(\kappa-1)} = \begin{bmatrix} \begin{bmatrix} I & U_1^{(\kappa,1)}V_{1,2}^{(\kappa)^T} \\ U_2^{(\kappa,1)}V_{2,1}^{(\kappa)^T} & I \end{bmatrix} & U_1^{(\kappa-1,1)}V_{1,2}^{(\kappa-1)^T} & \cdots & \cdots \\ U_2^{(\kappa-1,1)}V_{2,1}^{(\kappa-1)^T} & \begin{bmatrix} I & U_3^{(\kappa,1)}V_{3,4}^{(\kappa)^T} \\ U_4^{(\kappa,1)}V_{4,3}^{(\kappa)^T} & I \end{bmatrix} & \cdots & \cdots \\ \vdots & \vdots & \ddots & \cdots \\ \vdots & \vdots & \vdots & \begin{bmatrix} I & U_{2^\kappa-1}^{(\kappa,1)}V_{2^\kappa-1,2^\kappa}^{(\kappa)^T} \\ U_{2^\kappa}^{(\kappa,1)}V_{2^\kappa,2^\kappa-1}^{(\kappa)^T} & I \end{bmatrix} \end{bmatrix} \tag{19}$$

Note that $U_i^{(k,1)}$ indicates that $U_i^{(k)}$ has been updated after factoring out K_κ. 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

$$\begin{bmatrix} I & U_{2i-1}^{(\kappa,1)}V_{2i-1,2i}^{(\kappa)^T} \\ U_{2i}^{(\kappa,1)}V_{2i,2i-1}^{(\kappa)^T} & I \end{bmatrix} = K_i^{(\kappa-1,1)}, \tag{20}$$

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

$$K^{(\kappa-1)} = \begin{bmatrix} K_1^{(\kappa-1,1)} & U_1^{(\kappa-1,1)}V_{1,2}^{(\kappa-1)^T} & \cdots & \cdots \\ U_2^{(\kappa-1,1)}V_{2,1}^{(\kappa-1)^T} & K_2^{(\kappa-1,1)} & \cdots & \cdots \\ \vdots & \vdots & \ddots & \cdots \\ \vdots & \vdots & \vdots & K_{2^{\kappa-1}}^{(\kappa-1,1)} \end{bmatrix} \tag{21}$$

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 (22):

$$U_i^{(k,1)} = \begin{bmatrix} U_{2i-1}^{(k+1,1)}S_{2i-1}^{(k+1)} \\ U_{2i}^{(k+1,1)}S_{2i}^{(k+1)} \end{bmatrix} \tag{22}$$

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 (23), and $K^{(\kappa-2)}$ is a $(\kappa-2)$-level HODLR matrix.

$$K_{\kappa-1} = \begin{bmatrix} K_1^{(\kappa-1,1)} & 0 & 0 & \cdots \\ 0 & K_2^{(\kappa-1,1)} & \cdots & \cdots \\ 0 & \vdots & \ddots & \cdots \\ \vdots & \vdots & \vdots & K_{2^{\kappa-1}}^{(\kappa-1,1)} \end{bmatrix} \tag{23}$$

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 (20) 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

$$K = K_\kappa K_{\kappa-1} \cdots K_0 \tag{24}$$

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 κ, 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:

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	$\sum_{k=1}^{\kappa} \mathcal{O}(p^2N) = \mathcal{O}(\kappa p^2N) = \mathcal{O}(p^2N\log N)$

Figure 1 provides a pictorial description of this factorization.

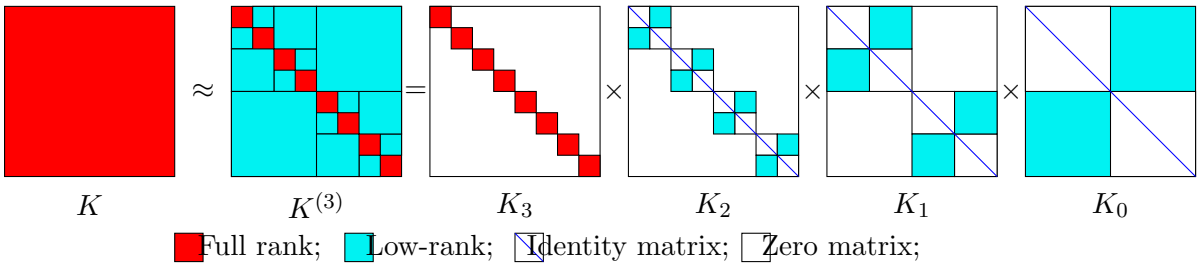

Figure 1: Factorization of a three level HODLR/p-HSS matrix

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 $x = K_0^{-1}K_1^{-1}\ldots K_\kappa^{-1}b$. Hence, we need to first apply the inverse of K_κ 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_κ 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 1.

Table 1: Computational complexity; p: rank; r: number of right-hand sides; N: size of matrix

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

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.

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 [8, 10, 20, 54, 60, 63]. 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, λ_k are a set of weights and ϕ is a function from $\mathbb{R} \to \mathbb{R}$. We set $\phi(0) = 0$. This is termed the nugget effect [1, 4, 19, 21, 49] 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

$$s(x) = \sum_{k=1}^{N} \lambda_k \phi(x - x_k) + \sum_{j=0}^{l} a_j p_j(x) \tag{25}$$

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 (25) can be rewritten as

$$s(x) = \sum_{k=1}^{N} \lambda_k \phi\left(\|x - x_k\|_2\right) + \sum_{j=0}^{l} a_j p_j(x) \tag{26}$$

To determine the interpolant, we need to determine the λ_k's and the a_j's, a total of $N + l + 1$ unknowns. The interpolant in equation (26) is required to satisfy the interpolation conditions.

$$s(x_k) = f_k, \qquad \forall k \in \{1, 2, \ldots, n\} \tag{27}$$

Equation (27) ensures that the interpolant passes through the data points. The remaining equations are obtained through the *side conditions* given in equation (28).

$$\sum_{k=1}^{N} \lambda_k p_j(x_k) = 0, \qquad \forall j \in \{0, 1, \ldots, l\} \tag{28}$$

The side conditions in equation (28) 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

$$\begin{bmatrix} \Phi & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ a \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix} \tag{29}$$

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

5.2 Problem specification

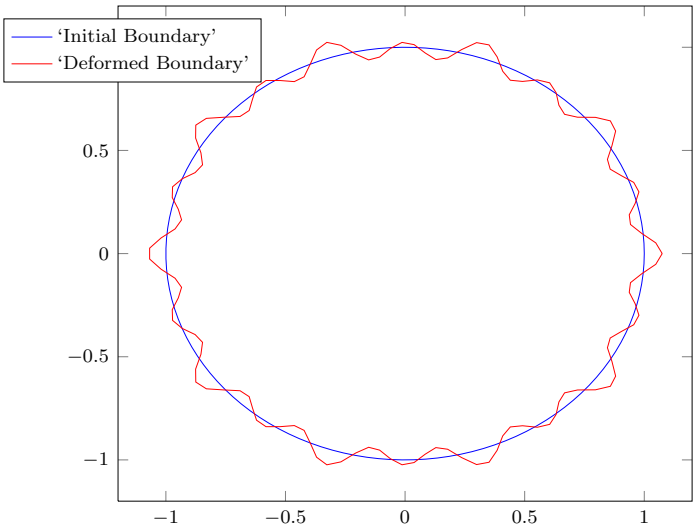

Figure 2: Deformation of a one dimensional manifold

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 θ from a uniform distribution in the interval $[0, 2\pi)$.

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

$$\Phi\,\lambda_1 = P, \qquad \Phi\,\lambda_2 = f \tag{30}$$

where $\Phi \in \mathbb{R}^{N\times N}$. To benchmark our algorithm, we compare the time taken by different algorithms to solve equation (30) 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 [34], 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.

5.3 Results and discussion

We present a detailed numerical benchmark for the radial basis functions listed in Table 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 $r_{ij} = \|x_i - x_j\|_2 = 2\left|\sin\left(\dfrac{\theta_i - \theta_j}{2}\right)\right|$.

Table 2: Radial basis functions $\phi(r)$ for which numerical benchmarking was performed.

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

Consider for example an off-diagonal block in the matrix with entries of the form $\Phi_{ij} = \phi(r_{ij}) = \phi(\|x_i - x_j\|_2) = \phi\left(2\left|\sin\left(\dfrac{\theta_i - \theta_j}{2}\right)\right|\right) = \psi(\theta_i - \theta_j)$, 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 θ.

In [22, 46], 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 ψ 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 ψ in the complex plane and the interval $\mathcal{I}$, and the growth of ψ 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 (2). In order to reduce the number of pages, not all our results are shown in this paper. 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 in this paper. 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 (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 (3). There was no noticeable growth of the rank of the off-diagonal blocks with N.

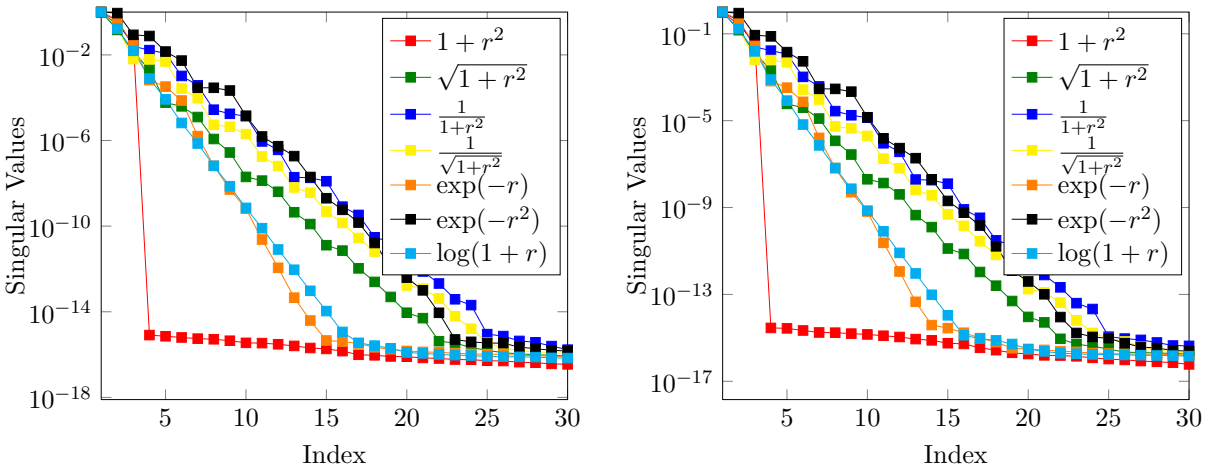

Figure 3: Decay of singular values of the off-diagonal blocks for different radial basis functions. The singular values are normalized using the largest singular-value. Left: system size 1024×1024; Right: system size 8192×8192.

- We then considered 8192×8192 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 4096×4096. The condition numbers ranged from $3 \cdot 10^3$ to $2 \cdot 10^7$ as shown in table (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 3: Condition number of a 8192×8192 system for different kernels where the points are distributed randomly on a unit circle.

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

- For the 8192×8192 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 (λ, 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 4.1) and perform the solve (section 4.2).

Recall that the factorization phase and the solve phase proceed together as mentioned in section 4.2.

- Next for all the radial basis functions, we fixed the off-diagonal rank at 30 and used Chebyshev interpolation [22] for θ, 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 131072×131072 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.

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 [34], 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 (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 (4), (5) and table (4) indicate. A detailed analysis of the assembly time and solve time at each level for the fast algorithms is shown in figure (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 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 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.

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 (3). However, since we are constructing the low-rank using θ, 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 (7).

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 (3). The relative error as a function of rank and system size are plotted in figure (8).

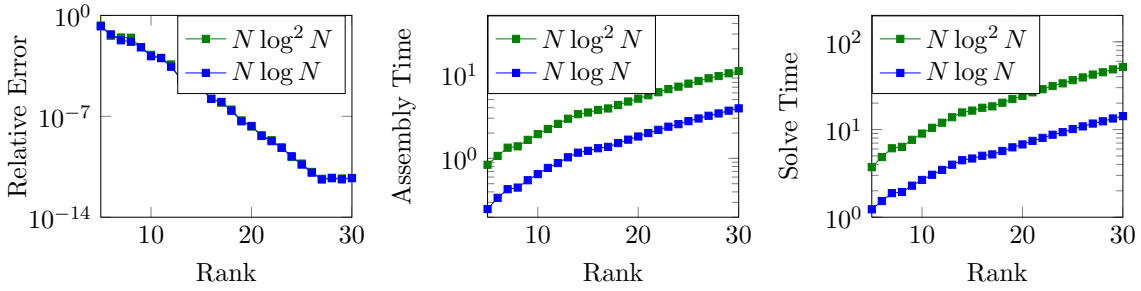

Figure 4: Keeping the system size fixed at 8192; Left: Relative error; Middle: Time taken to assemble in seconds; Right: Time taken to solve in seconds; versus rank of the off-diagonal blocks

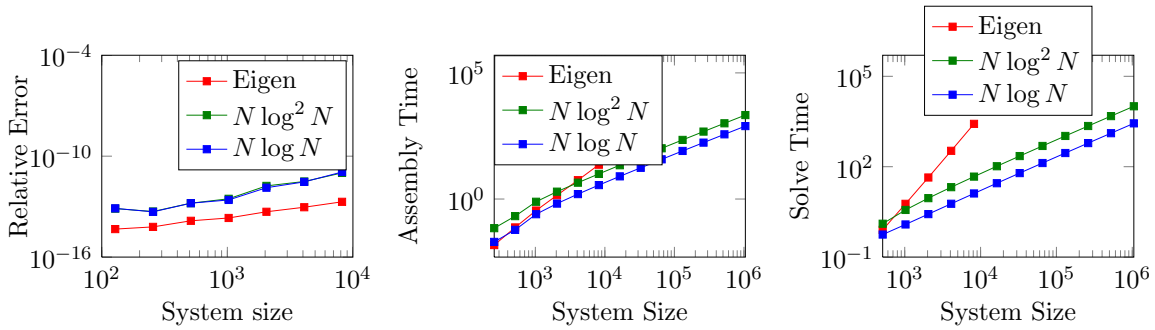

Figure 5: Keeping the rank of the off-diagonal block fixed at 30; Left: Relative error; Middle: Time taken to assemble in seconds; Right: Time taken to solve in seconds; versus system size N.

Table 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	–	–	–	$2.1 \cdot 10^{+3}$	$1.0 \cdot 10^{+4}$	$1.2 \cdot 10^{+4}$	$7.8 \cdot 10^{+2}$	$2.7 \cdot 10^{+3}$	$3.5 \cdot 10^{+3}$

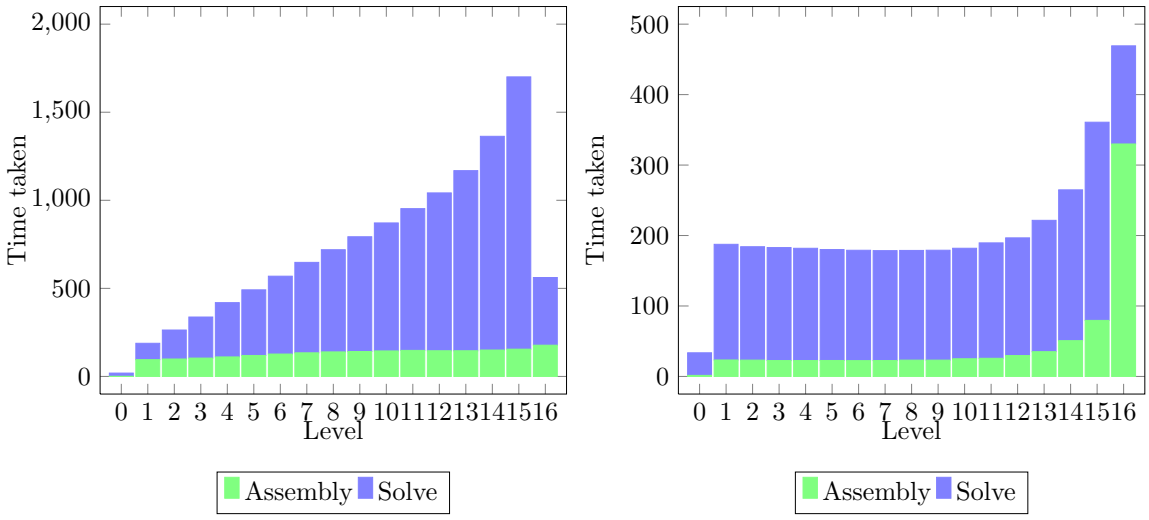

Figure 6: Split up of the time taken by the algorithms at each level for a $1,048,576 \times 1,048,576$ system. Left: $N \log^2 N$ algorithm; Right: $N \log N$ algorithm

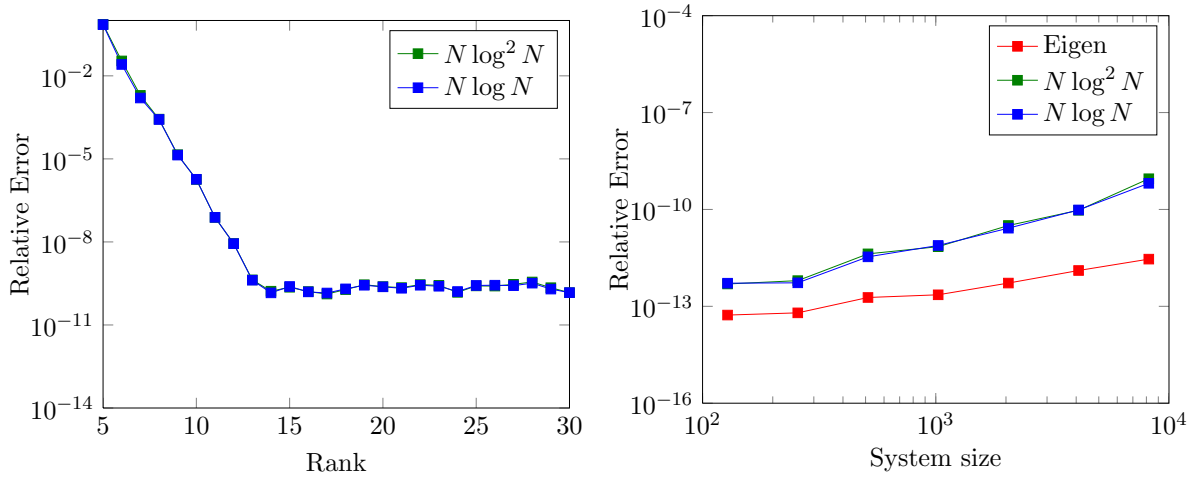

Figure 7: Relative error for $1 + (r/a)^2$. Left: versus rank for a system size of 8192 (the green and blue curves overlap); Right: versus system size keeping the off-diagonal rank as 30.

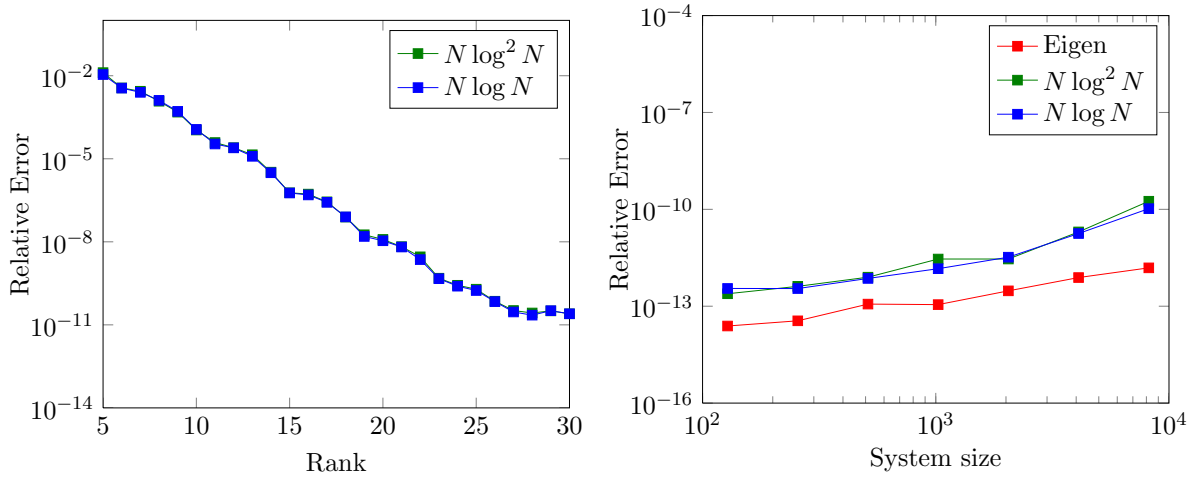

Figure 8: Relative error for $\sqrt{1 + (r/a)^2}$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30.

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} singular-value is close to machine precision as seen in figure (3). The relative error as a function of rank and system size are plotted in figure (9).

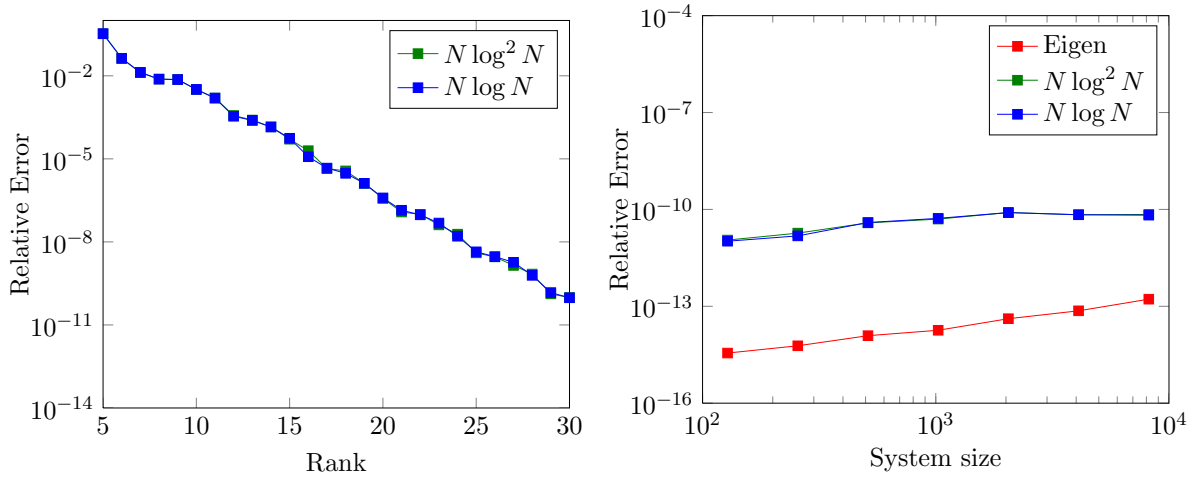

Figure 9: Relative error for $\frac{1}{1+(r/a)^2}$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30.

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 (3). The relative error as a function of rank and system size are plotted in figure (10).

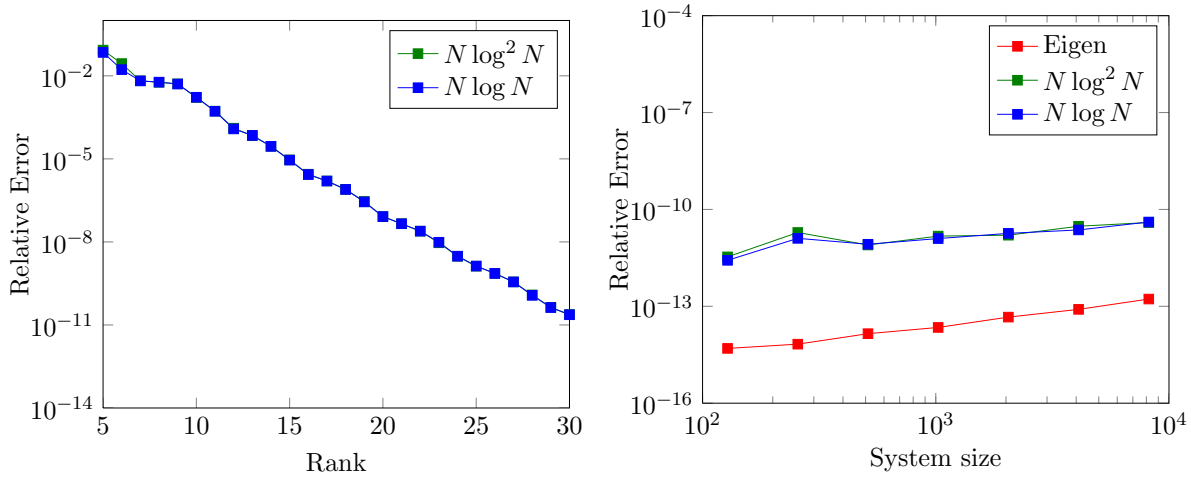

Figure 10: Relative error for $\frac{1}{\sqrt{1+(r/a)^2}}$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30.

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 (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 (11).

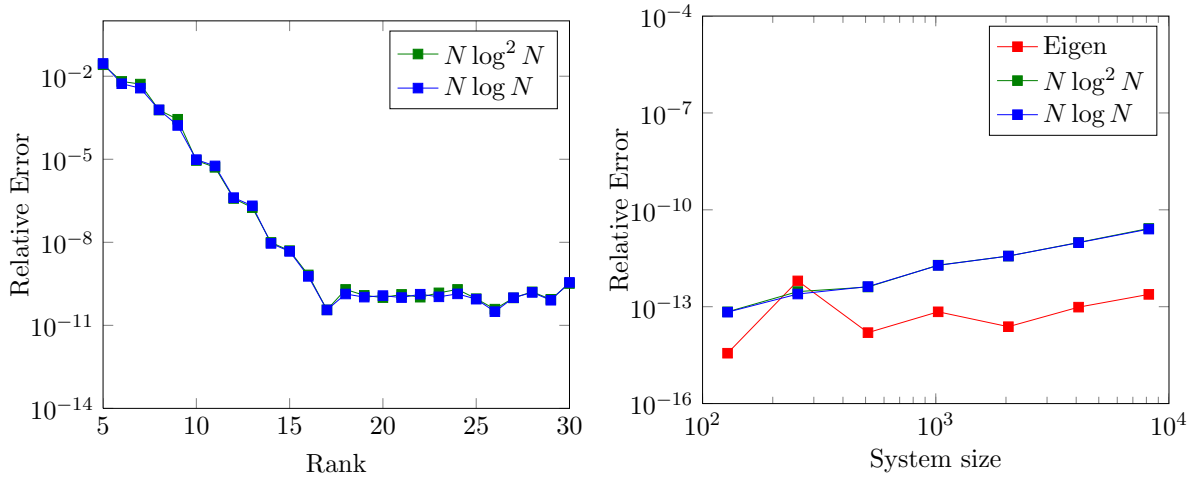

Figure 11: Relative error for $\exp(-r/a)$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30.

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 precision as seen in figure (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 (12).

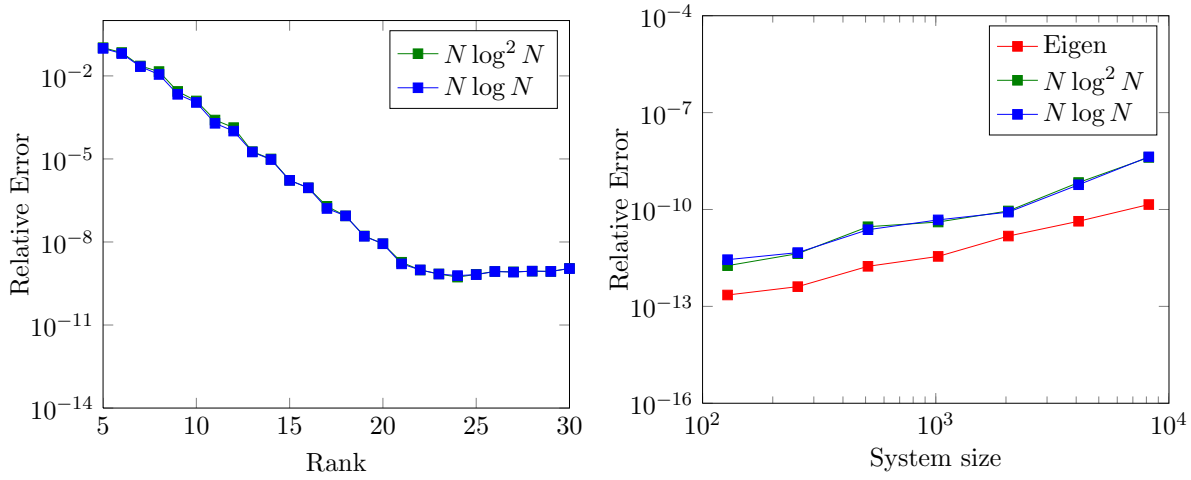

Figure 12: Relative error for $\log(1 + r/a)$. Left: versus rank for a system size of 8192; Right: versus system size keeping the off-diagonal rank as 30.

6 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. The C++ implementation can be found at `http://www.stanford.edu/~sivaambi/Fast_Direct_Solver_PACKage.html`.

acknowledgements

Sivaram Ambikasaran would like to thank Krithika Narayanaswamy for proof reading the paper and helping in generating the figures.

References

[1] Andrianakis I, Challenor P (2012) The effect of the nugget on gaussian process emulators of computer models. Computational Statistics & Data Analysis

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

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

[4] Baxter B (2010) The interpolation theory of radial basis functions. arXiv preprint arXiv:10062443

[5] Beatson R, Greengard L (1997) A short course on fast multipole methods. Wavelets, multilevel methods and elliptic PDEs pp 1–37

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

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

[8] Billings S, Beatson R, Newsam G (2002) Interpolation of geophysical data using continuous global surfaces. Geophysics 67(6):1810

[9] Börm S, Grasedyck L, Hackbusch W (2003) Hierarchical matrices. Lecture notes 21

[10] Buhmann M (2003) Radial basis functions: theory and implementations, vol 12. Cambridge University Press

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

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

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

[14] Cheng H, Greengard L, Rokhlin V (1999) A fast adaptive multipole algorithm in three dimensions. Journal of Computational Physics 155(2):468–498

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

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

[17] Darve E (2000) The fast multipole method i: Error analysis and asymptotic complexity. SIAM Journal on Numerical Analysis 38(1):98–128

[18] Darve E (2000) The fast multipole method: numerical implementation. Journal of Computational Physics 160(1):195–240

[19] Davis G, Morris M (1997) Six factors which affect the condition number of matrices associated with kriging. Mathematical geology 29(5):669–683

[20] De Boer A, Van der Schoot M, Bijl H (2007) Mesh deformation based on radial basis function interpolation. Computers & Structures 85(11-14):784–795

[21] Dietrich C, Newsam G (1995) 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

[22] Fong W, Darve E (2009) The black-box fast multipole method. Journal of Computational Physics 228(23):8712–8725

[23] Freund R (1993) A transpose-free quasi-minimal residual algorithm for non-hermitian linear systems. SIAM Journal on Scientific Computing 14:470

[24] Freund R, Nachtigal N (1991) QMR: a quasi-minimal residual method for non-hermitian linear systems. Numerische Mathematik 60(1):315–339

[25] Frieze A, Kannan R, Vempala S (2004) Fast Monte-Carlo algorithms for finding low-rank approximations. Journal of the ACM (JACM) 51(6):1025–1041

[26] Gillman A, Young P, Martinsson P (2011) A direct solver with $\mathcal{O}(N)$ complexity for integral equations on one-dimensional domains. arXiv preprint arXiv:11055372

[27] Golub G, Van Loan C (1996) Matrix computations, vol 3. Johns Hopkins Univ Press

[28] Goreinov S, Tyrtyshnikov E, Zamarashkin N (1997) A theory of pseudoskeleton approximations. Linear Algebra and its Applications 261(1-3):1–21

[29] Grasedyck L, Hackbusch W (2003) Construction and arithmetics of h-matrices. Computing 70(4):295–334

[30] Greengard L, Rokhlin V (1987) A fast algorithm for particle simulations. Journal of Computational Physics 73(2):325–348

[31] Greengard L, Rokhlin V (1997) A new version of the fast multipole method for the Laplace equation in three dimensions. Acta Numerica 6(1):229–269

[32] Greengard L, Gueyffier D, Martinsson P, Rokhlin V (2009) Fast direct solvers for integral equations in complex three-dimensional domains. Acta Numerica 18(1):243–275

[33] Gu M, Eisenstat S (1996) Efficient algorithms for computing a strong rank-revealing QR factorization. Society 17(4):848–869

[34] Guennebaud G, Jacob B, et al (2010) Eigen v3. http://eigen.tuxfamily.org

[35] Gumerov N, Duraiswami R (2007) Fast radial basis function interpolation via preconditioned Krylov iteration. SIAM Journal on Scientific Computing 29(5):1876–1899

[36] Hackbusch W (1999) A sparse matrix arithmetic based on\ cal h-matrices. part i: Introduction to {\ Cal H}-matrices. Computing 62(2):89–108

[37] Hackbusch W, Börm S (2002) Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. Computing 69(1):1–35

[38] Hackbusch W, Khoromskij BN (2000) A sparse $\mathcal{H}$-matrix arithmetic. Computing 64(1):21–47

[39] Hackbusch W, Nowak Z (1989) On the fast matrix multiplication in the boundary element method by panel clustering. Numerische Mathematik 54(4):463–491

[40] Hager W (1989) Updating the inverse of a matrix. SIAM review pp 221–239

[41] Hestenes M, Stiefel E (1952) Methods of conjugate gradients for solving linear systems. Journal of Research of the National Bureau of Standards 49(6):409–436

[42] Kong WY, Bremer J, Rokhlin V (2011) An adaptive fast direct solver for boundary integral equations in two dimensions. Applied and Computational Harmonic Analysis 31(3):346–369

[43] Liberty E, Woolfe F, Martinsson P, Rokhlin V, Tygert M (2007) Randomized algorithms for the low-rank approximation of matrices. Proceedings of the National Academy of Sciences 104(51):20,167

[44] Martinsson P (2009) A fast direct solver for a class of elliptic partial differential equations. Journal of Scientific Computing 38(3):316–330

[45] Martinsson P, Rokhlin V (2005) A fast direct solver for boundary integral equations in two dimensions. Journal of Computational Physics 205(1):1–23

[46] Messner M, Schanz M, Darve E (2011) Fast directional multilevel summation for oscillatory kernels based on Chebyshev interpolation. Journal of Computational Physics

[47] Miranian L, Gu M (2003) Strong rank revealing LU factorizations. Linear Algebra and its Applications 367:1–16

[48] Nishimura N (2002) Fast multipole accelerated boundary integral equation methods. Applied Mechanics Reviews 55(4):299–324

[49] O'Dowd RJ (1991) Conditioning of coefficient matrices of ordinary kriging. Mathematical Geology 23(5):721–739

[50] Paige CC, Saunders MA (1975) Solution of sparse indefinite systems of linear equations. SIAM Journal on Numerical Analysis 12(4):617–629

[51] Pan CT (2000) On the existence and computation of rank-revealing LU factorizations. Linear Algebra and its Applications 316(1):199–222

[52] Rjasanow S (2002) Adaptive cross approximation of dense matrices. IABEM 2002, International Association for Boundary Element Methods

[53] Saad Y, Schultz M (1986) GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM Journal on Scientific and Statistical Computing 7(3):856–869

[54] Schaback R (1995) Creating surfaces from scattered data using radial basis functions. Mathematical methods for curves and surfaces pp 477–496

[55] Schmitz P, Ying L (2011) A fast direct solver for elliptic problems on general meshes in 2D. Journal of Computational Physics

[56] Schmitz P, Ying L (2012) A fast direct solver for elliptic problems on Cartesian meshes in 3D, in review

[57] Vandebril R, Barel M, Golub G, Mastronardi N (2005) A bibliography on semiseparable matrices. Calcolo 42(3):249–270

[58] Vavasis SA (1992) Preconditioning for boundary integral equations. SIAM journal on matrix analysis and applications 13(3):905–925

[59] Van der Vorst HA (1992) 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

[60] Wang J, Liu G (2002) A point interpolation meshless method based on radial basis functions. International Journal for Numerical Methods in Engineering 54(11):1623–1648

[61] Woodbury MA (1950) Inverting modified matrices, statistical Research Group, Memo. Rep. no. 42, Princeton University

[62] Woolfe F, Liberty E, Rokhlin V, Tygert M (2008) A fast randomized algorithm for the approximation of matrices. Applied and Computational Harmonic Analysis 25(3):335–366

[63] Wu Z, Schaback R (1993) Local error estimates for radial basis function interpolation of scattered data. IMA Journal of Numerical Analysis 13(1):13–27

[64] Xia J, Chandrasekaran S, Gu M, Li X (2010) Fast algorithms for hierarchically semiseparable matrices. Numerical Linear Algebra with Applications 17(6):953–976

[65] Ying L, Biros G, Zorin D (2004) A kernel-independent adaptive fast multipole algorithm in two and three dimensions. Journal of Computational Physics 196(2):591–626