Master Thesis Literature Review: Efficiency Improvement for Panel Codes

TU Delft
MARIN

Elisa Ang Yun Mei
Student Number: 4420888

1-27-2015

CONTENTS

1 INTRODUCTION

1.1 PROBLEM STATEMENT

At MARIN, Boundary Element Methods (BEM) are used to compute the flow around ships and propellers. There are many applications, with a few examples stated below [1]:

1. FATIMA: Used to compute ship motions and added resistance from incoming waves
2. PROCAL: Used for the analysis of propellers
3. EXCALIBUR: Computes the hull pressure fluctuations induced by the propeller
4. RAPID: computes the wave system generated by the ship

A characteristic of BEM is that the linear system of equation that has to be solved is dense. This is unlike methods like FEM, where the system of equations formed is sparse. A different strategy is hence required to solve the linear system of equations from BEM efficiently. This is important, as BEM is used in applications like ship simulators, where the system of equation has to be solved in "real-time". Therefore, the aim of this project is improve the performance of the dense linear solver.

1.2 CURRENT STRATEGIES

Currently, GMRES combined with incomplete LU-decomposition preconditioner, is used to solve the systems of equations formed. M. de Jong had proposed the use of GMRES with Block-Jacobi preconditioner [1] to solve the dense system of equations more efficiently. Parallelization techniques using OpenMP and Graphics Processing Units (GPUs) were also studied to improve the performance of the Block-Jacobi preconditioner. The best solve times are listed in [1, Table 40].

1.3 REPORT PURPOSE AND OVERVIEW

The purpose of this report is to document the literature study on the available strategies to address the problem. Using test matrices provided by MARIN, the different possible strategies are tested and the results detailed in this report. The report will conclude with the plan for the thesis project following this literature study.

The report is structured in the following way. To gain a deeper understanding of the matrix characteristic, the concepts of BEM are presented in Section 2. Section 3 discusses the two solver methods considered, GMRES and IDR(s). Section 4 will address the different methods of preconditioning that could be employed to speed up computation. Section 5 is dedicated to the discussion of Fast Multipole Methods. To obtain a feel about the feasibility of the different methods discussed, MATLAB experiments were conducted to compare the performance. This is presented in Section 6. The report concludes with a discussion on the plan for the subsequent thesis work.

2 BOUNDARY ELEMENT METHOD (BEM)

BEM is a numerical method used to solve boundary value problems. Its development is relatively recent, as compared to its counterpart, the Finite Element Method (FEM). One major advantage of BEM over FEM is that it only requires discretization of the boundary, while FEM requires discretization of the entire domain. This effectively reduce the dimension of the problem by 1, thus constructing a significantly smaller system of equation. However, the system of equation that arise out of BEM is dense.

In this section, we will seek to outline the mathematical concepts of BEM, leading to the boundary integral equations. Then, the discretization of the integral equation and the construction of the linear system of equation to be solved are presented. The Laplace problem will be used as an example in this section to illustrate the concept. The Laplace problem is relevant in this case, since it described flows that are assumed to be irrotational, inviscid and incompressible. This section is written with reference to [2].

2.1 MATHEMATICAL CONCEPTS BEHIND BEM

Consider the Laplace problem, which is stated below, with a boundary $S = S_1 \cup S_2$.

$$-\nabla^2 u = 0$$

$$u = u_0 \; on \; S_1$$

$$q = \frac{\partial u}{\partial n} = q_0 \; on \; S_2$$

Equation 1

2.1.1 Weak formulation of the PDE

Like FEM, a weak formulation has to be employed to transform the PDE in its strong form to the weak integral form. The weak form can be obtained by first multiplying the strong form shown in Equation 1 with the trial function u*, taking the integral over the entire domain, and finally applying Green's Theorem to obtain Equation 2.

$$-\iiint\limits_V u^*\nabla^2 u \, dV = \iiint\limits_V \nabla u^* \nabla u \, dV - \iint\limits_S u^* \nabla \mathrm{u} \cdot \hat{\boldsymbol{n}} \, ds = 0$$

Equation 2

2.1.2 Fundamental Solution as Trial function

BEM uses the fundamental solution of the governing equation as the trial function u^*. A fundamental solution of our Laplace problem satisfies:

$$-\nabla^2 u^* = \delta(x_i)$$

Equation 3

In [2], Kythe, K.P. (1995) provides insight into techniques that can be employed to obtain the fundamental solution of boundary value problems. For our case, the Laplace problem has the following fundamental solutions:

$$u_i^* = \frac{1}{4\pi r} \text{ in 3D}, \qquad u_i^* = \frac{1}{2\pi} \ln \frac{1}{r} \text{ in 2D}$$

Equation 4

Where r is defined as:

$$r = |\boldsymbol{x} - \boldsymbol{x_i}|$$

2.1.3 Derivation of the Boundary Integral Equation

From Equation 2, we can apply the Green's Theorem again to obtain:

$$-\iiint\limits_V u\nabla^2 u^* \, dV + \iint\limits_S u \nabla \mathrm{u}^* \cdot \hat{\boldsymbol{n}} \, ds - \iint\limits_S u^* \nabla \mathrm{u} \cdot \hat{\boldsymbol{n}} \, ds = 0$$

Since u^* is the fundamental solution, it satisfies Equation 3, which can then be substituted to give:

$$-\iint\limits_S (u_i^* \nabla \mathrm{u} - u \nabla u_i^*) \cdot \hat{\boldsymbol{n}} \, ds + \iiint\limits_V u \, \delta(x_i) \, dV = -\iint\limits_S (u_i^* \nabla \mathrm{u} - u \nabla u_i^*) \cdot \hat{\boldsymbol{n}} \, ds + u(x_i) = 0$$

Noting that $\nabla u . \boldsymbol{n} = \frac{\partial u}{\partial n} = q$, we obtain:

$$\iint\limits_S (qu_i^* - uq_i^*)\, ds - u(x_i) = 0$$

Splitting the surface integral up into S_1 and S_2, and applying the boundary condition as given in Equation 1, we obtain:

$$u(x_i) = \iint\limits_{S_1} (qu_i^* - u_0 q_i^*)\, ds + \iint\limits_{S_2} (q_0 u_i^* - uq_i^*)\, ds$$

Equation 5

However, we can see from Equation 4 that u_i^* is singular at $x = x_i$. To deal with this singularity, we will consider the solution $u(x_i)$ for difference cases:

1. Case 1: When x_i is on the boundary

   In this case, we could have a singularity in the surface integral. First assume that the singularity is on S_2, and extend the boundary S_2 with a semi-sphere S_ϵ around the singular point. Let $S_{2-\varepsilon}$ be the surface S_2 without the semi-sphere. In 2-D, when S is a curve, Figure 1 illustrates this.

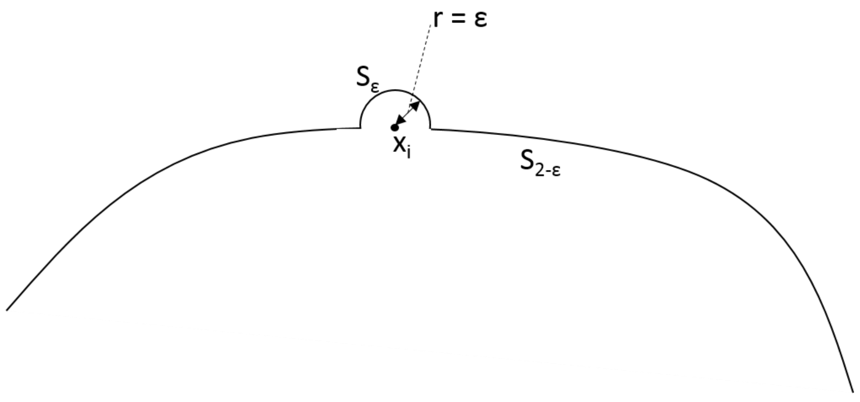


Figure 1 Extension of surface S

   Then, the integral splits into:

$$\iint\limits_{S_2} (q_0 u_i^* - uq_i^*)\, ds = \iint\limits_{S_{2-\varepsilon}} (q_0 u_i^* - uq_i^*)\, ds + \iint\limits_{S_\varepsilon} (q_0 u_i^* - uq_i^*)\, ds$$

   Focusing on the integral around the singular point, note that $q_i^* = \frac{du_i^*}{dn} = \frac{du_i^*}{dr} = -\frac{1}{4\pi r^2}$.

   Therefore, we have:

$$\iint_{S_\varepsilon} (q_0 u_i^* - u q_i^*)\, ds = \iint_{S_\varepsilon} \frac{1}{4\pi\epsilon} q_0 + \frac{u(x_i)}{4\pi\varepsilon^2} ds = \frac{q_0 \epsilon}{2} + \frac{u(x_i)}{2}$$

Taking the limit as $\varepsilon \to 0$, we obtain

$$\lim_{\varepsilon \to 0} \iint_{S_\varepsilon} (q_0 u^* - u q^*)\, ds = \frac{u(x_i)}{2}$$

Equation 6

Hence, substituting Equation 6 into Equation 5, we will arrive at:

$$\frac{u(x_i)}{2} = \iint_{S_1} (q u_i^* - u_0 q_i^*)\, ds + \iint_{S_2} (q_0 u_i^* - u q_i^*)\, ds$$

The same process can be repeated for singularity point at S_1 to obtain the same result.

2. Should the singularity point be not on the boundary, but still within the domain, the surface integral would not contain a singularity. Hence, Equation 5 can be used as it is.

With this, we arrive at the final Boundary Integral Equation:

$$c(x_i)u(x_i) + \iint_S u q_i^*\, dS = \iint_S u_i^* q\, dS\,, \qquad S = S_1 \cup S_2$$

$$c(x_i) = \begin{cases} 1 & if\ x_i\ is\ inside\ R \\ \frac{1}{2} & if\ x_i\ is\ on\ a\ smooth\ portion\ of\ S \end{cases}$$

Equation 7

2.2 DISCRETIZATION

To evaluate the boundary integral, we need to now discretize the boundary into segments. Like FEM, different boundary elements can be used.

The simplest case of the constant element would be considered in this report. Discretizing the surface into N constant elements would allow us to write Equation 7 in its discretized form:

$$c(x_i)u(x_i) + \sum_{j=1}^{N} u(x_j)\,\hat{H}_{ij} = \sum_{j=1}^{N} q(x_j)\,G_{ij}$$

$$\hat{H}_{ij} = \iint_{S_j} q_i^*\,dS$$

$$G_{ij} = \iint_{S_j} u_i^*\,dS$$

Equation 8

All $u(x_i)$ at the boundary have to be solved by constructing a linear system of equation. By writing Equation 8 for all elements from i=1 to N, we obtain the following N by N system of equation:

$$HU = GQ$$

$$H_{ij} = \begin{cases} \hat{H}_{ij} & for\ i \neq j \\ \hat{H}_{ij} + \frac{1}{2} & for\ i = j \end{cases}$$

$$U = [u_1 \quad \cdots \quad u_j \quad \cdots \quad u_N]^T$$

$$Q = [q_1 \quad \cdots \quad q_j \quad \cdots \quad q_N]^T$$

Equation 9

There will be N_1 values of u_j and N_2 values of q_j that are known from the boundary conditions. Hence, rearranging equation 9 such that all the unknowns are on the right hand side, we arrive at the linear system of equation:

$$Ax = b$$

Solving this linear system of equation will therefore provide us with all the u and q values on the boundary. With these values known, the integral $\hat{H}_{ij}$ and G_{ij} can be evaluated, and all interior points can be computed by Equation 7.

2.3 TEST MATRICES

MARIN provided us with a few test matrices generated from their existing systems. The matrices are all completely dense, and their characteristics are summarized below:

Name	Size	Real/Complex	System
Steadycav1	4620	Real	PROCAL
Steadycav2	4620	Real	PROCAL
Steadycav3	4620	Real	PROCAL
Steadycav4	4649	Real	PROCAL
DIFFRAC_1828	1828	Complex	DIFFRAC
FATIMA_7894	7894	Complex	FATIMA
FATIMA_20493	20493	Complex	FATIMA

Table 1 Test Matrices

3 SOLVER METHODS CONSIDERED

3.1 DIRECT METHOD –GAUSSIAN ELIMINATION

The Gaussian Elimination method refers to first performing an LU decomposition of a matrix A into an upper and lower triangular form.

$$A = LU = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ x & \cdots & 1 \end{bmatrix} \begin{bmatrix} x & \cdots & x \\ \vdots & \ddots & \vdots \\ 0 & \cdots & x \end{bmatrix}$$

Where x indicates non-zero entries.

Next, the decomposed system of equation is solved using forward and backward substitution. This is also known as Gaussian elimination without pivoting.

In terms of operation count, the LU decomposition of a matrix of dimension N x N requires O(N^3) flops[1]. Due to the O(N^3) flops required, When N is large, Gaussian Elimination becomes infeasible. Hence, there is a need to turn to other methods.

For further details on Gaussian Elimination, the reader is referred to [3].

3.2 ITERATIVE METHOD

Direct methods require O(N^3) flops to solve a linear system of equation. For certain matrices that are regular, iterative methods are known to bring the complexity down to O(N^2) or lower. Hence, most practical solvers utilizes iterative method.

[1] Refers to Floating Point Operation per Second

An iterative algorithm is characterize by giving an initial guess for x, and iteratively updating x to reduce the residual. The Krylov subspace method is a subclass of iterative methods available that is very successful and widely applied. In this report, the focus would be on two Krylov methods: Generalized Minimized Residual (GMRES) and Induced Dimension Reduction (IDR).

Before diving into the two methods, the concept of a Krylov Subspace for non-hermitian systems of linear equations will be briefly introduced. For more information, the reader is referred to the many open source literature available on Krylov subspace, for example [3].

A Krylov subspace $\mathcal{K}_{i+1}$ is basically the space $span\{b, Ab, A^2b, \dots, A^ib\}$. Let A be $\mathbb{C}^{NxN}$ [2]. As i approaches $N-1$, $\mathcal{K}_N$ approaches $\mathbb{C}^N$, and therefore, converges to the solution space of $Ax = b$, since $x \in \mathbb{C}^N$. The idea of a Krylov subspace method is to find a reasonably good approximation for x within the space $\mathcal{K}_m$, where $m \ll N$.

3.2.1 GMRES

GMRES is the most common iterative method employed to solve $Ax = b$, when A is not hermitian. The current solver uses GMRES. This section aims to provide an overview of the GMRES method.

3.2.1.1 Mathematical Concept

Let K_m be the N x m Krylov matrix given by

$$K_m = [b|Ab|\cdots|A^{m-1}b]$$

K_m can be decomposed into its orthonormal base Q_m. Working with the orthonormal base instead of the Krylov matrix is important for stability.

$$Q_m = [q_1|q_2|\cdots|q_m], \text{ where}$$

$$span\{q_1, q_2, \dots, q_m\} = span\{b, Ab, \dots, A^{m-1}b\}$$

The idea of GMRES is to approximate the exact solution of x_* with a vector $x_m \in \mathrm{Q}_m$ such that the residual, $||r_m|| = ||b - Ax_m||$ is minimized at every m. Since $x_m \in \mathrm{Q}_m$, we can therefore express x_m as:

$$x_m = Q_m y, \qquad y \in \mathbb{C}^m$$

Equation 10

[2] $\mathbb{C}$ is the space of complex number

Therefore, Ax_m can be written as

$$Ax_m = AQ_m y = [Aq_1|Aq_2|\cdots|Aq_m]\, y$$

The Arnoldi iteration provides a way to decompose AQ_m into $Q_{m+1}\widetilde{H}_{m+1}$, where Q_m is a orthonormal matrix, and $\widetilde{H}_{m+1}$ is a Hessenburg matrix of the form.

$$\widetilde{H}_{m+1} = \begin{bmatrix} h_{11} & \cdots & h_{1m} \\ h_{21} & & h_{2m} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & h_{m,m} \\ 0 & & h_{m+1,m} \end{bmatrix}$$

The rationale for the Arnoldi iteration can be explained by first considering $AQ_m = Q_{m+1}\widetilde{H}_{m+1}$.

$$[A]\,[q_1|q_2|\cdots|q_m] = [q_1|\cdots|q_m|q_{m+1}] \begin{bmatrix} h_{11} & \cdots & h_{1m} \\ h_{21} & & h_{2m} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & h_{m,m} \\ 0 & & h_{m+1,m} \end{bmatrix}$$

This expands out to give at column m, the following equation

$$Aq_m = h_{1,m}q_1 + h_{2,m}q_2 + \cdots + h_{m+1,m}q_{m+1}$$

Since $q_{m+1}$is orthogonal to $< q_1, \cdots, q_m >$ by definition, and $< q_1, \cdots, q_{m+1} >$ should span the same space as $\mathcal{K}_{m+1} =< b, Ab, A^2b, \ldots, A^m b >$, we could employ a Gram-Schmidt style iteration by choosing

(a) $q_1 = \frac{b}{\|b\|}$

(b) $h_{j,m} = q_j * (Aq_m)$

(c) $h_{m+1,m}$ such that q_{m+1} is normalized

The Gram-Schmidt style iteration will thus form an orthonormal matrix Q_m, which will span the same space as $\mathcal{K}_m$, and a Hessenberg matrix $\widetilde{H}_{m+1}$ such that $AQ_m = Q_{m+1}\widetilde{H}_{m+1}$. This process is known as the Arnoldi iteration.

The Arnoldi iteration allows $\|b - AQ_m y\|$ to written as $\|b - Q_{m+1}\widetilde{H}_{m+1}y\| = \|{Q^*}_{m+1}b - \widetilde{H}_{m+1}y\|$.

Noting that the construction of the orthogonal base Q_m by Arnoldi iteration necessitates that $q_1 = \frac{b}{\|b\|}$, and the subsequent q_i are all orthogonal to q_1, we can then conclude that ${Q^*}_{m+1} b = \|b\| e_1$, where

$$e_1{}^T = (1| \quad 0| \quad \ldots | \quad 0)$$

The least square method is employed to solve the least square problem of minimizing $\|r_m\| = \left\| \|b\| e_1 - \widetilde{H}_{m+1} y \right\|$ with solution vector y. The approximate solution x_m can then be found by equation 10.

3.2.1.2 GMRES complexity analysis

The complexity of GMRES depends on the number of iterations required for convergence. Mathematical analysis by approximating the GMRES with a polynomial problem arrives at the following convergence result:

$$\frac{\|r_m\|}{\|b\|} \le \inf_{p_n \in P_n} \|p_n(A)\| \le \inf_{p_n \in P_n} \kappa(V) \max_{\lambda \in \sigma(A)} \|p_n(\lambda)\|$$

Where $p_n(A) \in P_n$, and P_n is a polynomial of degree ≤ n and with p(0)=1, V is a non-singular matrix obtained from the diagonalization of A into $A = V\Delta V^{-1}$.

The result suggest that for fast convergence of GMRES, we require

(a) The matrix A is not far from normality[3]
(b) The eigenvalues of A is clustered away from the origin

3.2.2 IDR method

3.2.2.1 The IDR Method and other Krylov Subspace Methods

Krylov Subspace Methods that solve large non-symmetric systems of equation can be divided into two broad categories: Hessenberg orthogonalization methods, which refers basically to GMRES, and short recurrence methods [3].

The two categories arise mainly from the fact that non-symmetric matrices, unlike symmetric ones, cannot be reduced by unitary reduction to a hermitian matrix of tridiagonal form. Hessernberg orthogonalization methods enforces unitary reductions, however, unitary reductions of non-hermitian matrix results in hessenberg form, which means that long

[3] A matrix A is normal if $A^*A = AA^*$

recurrence relation have to be formed. We can observe from the GMRES algorithm that at the nth iteration, all q_m from previous iterations, $1 \leq m < n$ are required to form q_{n+1}. We say therefore, that the depth of recursion for GMRES is n.

Short recurrence methods, on the other hand sacrifice the unitary reductions. Such methods include tridiagonal biorthogonalization biconjugate gradients (Bi-CG) based methods, and the IDR method discussed here. However, to understand the benefits of the IDR method, the benefits and drawbacks of Bi-CG based methods will be discussed briefly.

Bi-CG based method operates on the concept of building and maintaining two mutually orthogonal bases for the Krylov Spaces $\mathcal{K}(A, r_0)$and $\mathcal{K}(A^*, r_0)$, and from there, reduces the matrix to tridiagonal form. Each basis is not required to be unitary. The main advantage of Bi-CG method is that due to the tridiagonal form, a 3-term recurrence relation can be formed. Short term recurrence means work per step and the storage requirements remain constant with increasing iteration. This is attractive especially for large matrices. However, Bi-CG based methods are known to have two major drawbacks. Firstly, GMRES looks for x_m that minimizes the residual norm at every iteration m. In the case of short recurrence methods, it is in general not possible to have an optimal minimization of some error norm in the Krylov space [4]. Thus, the convergence behaviour of methods like Bi-CG is usually slower and often erratic [1]. The second drawback is that Bi-CG requires 2 matrix–vector products in each iteration, while GMRES requires only one. Since such matrix-vector product is of O(N^2), it significantly increase the cost of every iteration.

The IDR method was introduced by Sonneveld, P. & van Gijzen, M.B. in 2008. It approaches the problem in a different direction then the Bi-CG based method. IDR uses instead nested subspaces $\mathcal{G}_j$, where $\mathcal{G}_j \subset \mathcal{G}_{j-1}$ and $\mathcal{G}_0 = \mathcal{K}(A, r_0)$. The depth of recursion is determined by s. Thus, IDR method is a short recurrence method that has the benefit of requiring at most $N + \frac{N}{s}$ matrix-vector product to arrive at the exact solution, where N is the problem size, and s is the codimension of a fixed subspace [4]. This is a significant improvement over Bi-CG, which could require as much as 2N matrix-vector product.

3.2.2.2 Mathematical Concept

The fundamental concept behind the IDR method is based on the IDR theorem. Please refer to [4] for the IDR Theorem and its proof. In summary, the IDR Theorem states:

Given matrix A in $\mathbb{C}^{N\times N}$, let v_0 be any nonzero vector in $\mathbb{C}^N$, and let $\mathcal{G}_0$ be the full Krylov space $\mathcal{K}_N(A, v_0)$. $\mathcal{S}$ is a subspace of $\mathbb{C}^N$, and define the sequence $\mathcal{G}_j, j = 1, 2, \ldots$ as

$$\mathcal{G}_j = (\boldsymbol{I} - \omega_j \boldsymbol{A})(\mathcal{G}_{j-1} \cap \mathcal{S})$$

Equation 11

Where the ω_j's are nonzero scalars. Then the following hold:

(i) $\mathcal{G}_j \subset \mathcal{G}_{j-1}\ \forall j > 0$

(ii) $\mathcal{G}_j = \{\mathbf{0}\}, for\ some\ j \leq N$

The IDR theorem provides a way for us to approximate the solution of linear system of equations. This is done by looking for the residual $r_n = b - Ax_n$ within $\mathcal{G}_j$, that is $r_n \in \mathcal{G}_j$. The theorem implies that a sequence of nested subspace $\mathcal{G}_j \subset \mathcal{G}_{j-1}$ can be generated based on Equation 11, with the dimension of the subspace $\mathcal{G}_j$ decreasing. Point (ii) allows us to conclude that there will be some $j \leq N$, where the subspace $\mathcal{G}_j$ will reduce to just the $\{\mathbf{0}\}$ vector. This means that at some point, the residual will be $\{\mathbf{0}\}$, and this is when the exact solution is found.

3.2.2.3 Mathematical formulas for the algorithm

Since we look for residual r_n in subspace $\mathcal{G}_j$, we can say that $r_{n+1} \in \mathcal{G}_{j+1}$. Then, by Equation 11, this implies

$$r_{n+1} = (\mathbf{I} - \omega_{j+1}\boldsymbol{A})v_n\,, \qquad where\ v_n \in \mathcal{G}_j \cap \mathcal{S}$$

Equation 12

Define v_n as follows

$$v_n = r_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}\,, \qquad where$$

$$\Delta r_{n-l} = r_{n-l+1} - r_{n-l}$$

Equation 13

Note that v_n must be $\in \mathcal{G}_j$. Thus, for $i = n - \hat{l}, \ldots, n-1, n,\ \ r_i \in \mathcal{G}_j$. Therefore, $\hat{l} + 1$ vectors in $\mathcal{G}_j$ are required to build v_n.

Also, v_n must also be $\in \mathcal{S}$. Recall from the IDR theorem that $\mathcal{S}$ is any proper subspace of $\mathbb{C}^N$. Therefore, we can assume $\mathcal{S}$ to be the left null-space of some N x s matrix P:

$$P = [p_1| \quad p_2| \quad \cdots | \quad p_s] , \quad P^T s = 0 \; \forall s \in \mathcal{S}$$

Since $v_n \in \mathcal{S}$, v_n satisfies $P^T v_n = 0$. Substituting Equation 13 into the above, we obtain

$$P^T \left(r_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l} \right) = 0$$

Which can be rewritten as

$$(P^T \Delta R_n) c = P^T r_n, \quad with$$

$$\Delta R_n = [\Delta r_{n-1}| \quad \Delta r_{n-2}| \quad \cdots | \quad \Delta r_{n-\hat{l}}]$$
$$c^T = [\gamma_1| \quad \gamma_2| \quad \cdots | \quad \gamma_{\hat{l}}]$$

Equation 14

This system of equation has a unique solution for c if $\hat{l} = s$. This is the reason why s defines the recurrence depth.

Next, we need to set up the recurrence relation to compute r_{n+1} from Δr_n and the solution vector x_n from Δx_n. Since $r_n = b - Ax_n$,

$$r_{n+1} - r_n = \Delta r_n = -A\Delta x_n$$

Substituting Equation 13 into Equation 12 gives

$$\Delta r_n = -\omega_{j+1} A v_n - \Delta R_n c$$

Equation 15

Thus we have:

$$\begin{aligned} \Delta x_n &= \omega_{j+1} A^{-1} A v_n + A^{-1} \Delta R_n c \\ &= \omega_{j+1} v_n - \Delta X_n c \end{aligned}$$

Equation 16

Putting the mathematical formulas into an algorithm, we obtained:

If we assume $\{r_n, r_{n-1}, \dots, r_{n-s}\} \in \mathcal{G}_j$, then we can compute $r_{n+1} \in \mathcal{G}_{j+1}$ as follows:

a. Using equation 14, Solve $(P^T \Delta R_n) c = P^T r_n$ for c
b. Using equation 13, solve $v_n = r_n - \Delta R_n c$
c. Using equation 15, solve $\Delta r_n = -\omega_{j+1} A v_n - \Delta R_n c$
d. Using equation 16, solve $\Delta x_n = \omega_{j+1} v_n - \Delta X_n c$

e. Subsequently, we need to generate s+1 vectors in $\mathcal{G}_{j+1}$. This can be done by repeating step (a), (b), (c) and (d). Note that since $\{\Delta r_n, \Delta r_{n-1}, \dots, \Delta r_{n-s}\}$ *and* $\{v_{n+1}, v_{n+2}, \dots, v_{n+s}\} \in \mathcal{G}_j$, $\{r_{n+1}, r_{n+2}, \dots, r_{n+s+1}\} \in \mathcal{G}_{j+1}$.

f. Once s+1 vectors in $\mathcal{G}_{j+1}$ is generated, we can then expect the next r_{n+s+2} residual to be in $\mathcal{G}_{j+2}$. Thus, we repeat from step (a).

3.2.2.4 Performance of the IDR Algorithm

By the IDR theorem (5.2.1), a maximum of N iteration $(j = 1, \dots, N)$ is required for the $\mathcal{G}_j$space to reduce from $\mathcal{G}_0$ to $\{0\}$, and within each j, recursion of depth s+1 needs to be performed. Thus, at the worst case, a total of $N \times (s+1)$ steps are required, with each step requiring one matrix-vector multiplication (Av_n). In practical applications, convergence occurs much earlier.

The rate at which the dimension of the $\mathcal{G}_j$ space is reduced can be proven to be "almost always" equal to s. This can be concluded from the proof of the extended IDR theorem, for which we refer to [4]. Assuming that the dimension of the $\mathcal{G}_j$ space is reduced by s for every outer iteration, we would then have a total of $\frac{N}{s}(s+1) = N + \frac{N}{s}$ steps required for convergence. Each step requires 1 matrix-vector multiplication involving Av. This means that a better performance can be achieved as compared to bi-CG, if the number of iterations required for convergence is comparable. The work per step as compared with GMRES could also be lower, especially when the problem size is huge, since IDR uses short term recurrence.

Numerical experiments using MATLAB have been conducted to compare the performance of these iterative solvers for our problem. This is presented in Section 6.

4 PRECONDITIONING

Krylov subspace methods are likely to suffer from slow convergence without the use of preconditioners for general matrices. Preconditioning involves the introduction of a preconditioning matrix M such that either of the following occur [5]:

(a) the linear system of equation Ax=b transforms into:

$$M^{-1}Ax = M^{-1}b$$

M is known as the left preconditioner in this case

(b) the linear system of equation Ax=b transforms into:

$$AM^{-1}u = b, x = M^{-1}\mathrm{u}$$

In either case, the preconditioned equation provides the same solution x as the original equation $Ax = b$.

This formulation highlights some important characteristics that the preconditioned matrix M should possess:

a. M should clearly be non-singular
b. Operations with M^{-1} should not be expensive
c. The matrix $M^{-1}A$ should have a better convergence characteristic when compared to A.

We will explore some of the preconditioning techniques that are thought to be applicable for our dense system of equations arising from BEM.

4.1 Block-Jacobi

The Block-Jacobi preconditioner is a domain decomposition method that is well suited for parallelization. Consider a matrix A that can be divided into the following domains:

$$A = \begin{bmatrix} A_{11} & A_{2p} & \cdots & A_{1p} \\ A_{21} & A_{22} & \cdots & A_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & A_{pp} \end{bmatrix}$$

We define the preconditioner M as

$$M = \begin{bmatrix} A_{11} & 0 & \cdots & 0 \\ 0 & A_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{pp} \end{bmatrix} = \begin{bmatrix} L_1 & 0 & \cdots & 0 \\ 0 & L_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & L_p \end{bmatrix} \begin{bmatrix} U_1 & 0 & \cdots & 0 \\ 0 & U_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & U_p \end{bmatrix}$$

We can then preconditioned the system in parallel for k from 1 to p in this way:

a. Let $Z_k = M_k^{-1}A_k$, where $Z_k, A_k \in ₵^{\mathrm{kxN}}$. Then, we solve the small system $M_k Z_k = A_k$ in parallel.
b. Next, let $w_k = M_k^{-1}b_k$, where $w_k, b_k \in ₵^{\mathrm{k}}$. Then, we solve the small system $M_k w_k = b_k$ in parallel.

Finally, we solve $Zx = w$.

This technique was applied and extensively tested in [1]. A similar code was implemented in MATLAB for experimental purposes (Appendix A.1). The result is presented in Section 6.

4.2 DEFLATION

The main idea behind the deflation preconditioner is to split the solution space of Ax=b into two complementary subspaces using a projection operator Q_d, $x = Q_d x + (I - Q_d)x$ [6]. Through the proper selection of Q_d, we can reduce the original system into a deflated system$P_D Ax = P_D b$, which should be easier to solve. The mathematical concepts behind this is presented below:

With reference to [7], we shall define the projectors:

$$P_D = I - AZE^{-1}Y^T$$

$$Q_D = I - ZE^{-1}Y^T A$$

$$where\ E = Y^T AZ \text{ and } Z, Y \in \mathbb{C}^{Nxd} \text{ are deflation subspaces}$$

Let the solution x be split into two complementary subspaces using Q_D. Thus,

$$x = (I - Q_D)x + Q_D x = ZE^{-1}Y^T b + Q_D x$$

Consider Ax:

$$\begin{aligned} Ax &= A(ZE^{-1}Y^T A)x + (A - AZE^{-1}Y^T A)x = (I - P_D)Ax + (I - AZE^{-1}Y^T)Ax \\ &= (I - P_D)Ax + P_D Ax \end{aligned}$$

Substituting Ax=b, we obtain

$$P_D Ax = P_D b$$

Let the solution of this system of equation be $\tilde{x}$, then the approximated solution to Ax=b is given by

$$x = ZE^{-1}Y^T b + Q_D \tilde{x}$$

We need d << N and E such that systems like Eu=f is easy to solve, since this will ensure that $ZE^{-1}Y^T b$ can be evaluated efficiently.

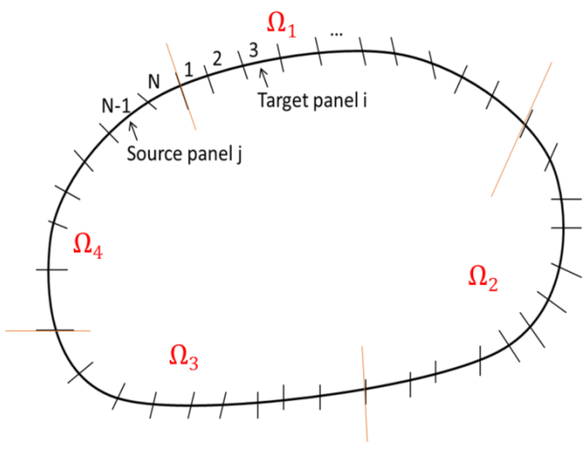
There are different ways to choose the deflation subspace. For instance, defining Z and Y as the space spanned by the d eigenvectors of A corresponding to the smallest eigenvalues. This has been shown to shift the small eigenvalues towards 0. This helps in cases where there are isolated eigenvalues close to 0 that are causing slow convergence. This approach was discussed in detail in [8]. However, computation of eigenvectors are expensive. Instead, subdomain decomposition would be employed instead.

In subdomain decomposition, Z=Y is chosen based on decomposition of the domain. With reference to [9], we define Z as follows. Let the domain be decomposed into d non-overlapping subdomains, Ω_i. Then each column z_i of Z, $1 \leq i \leq d$ is defined as follows:

$$z_{i_j} = \begin{cases} 0 & if\ x_i \notin \Omega_i \\ 1 & if\ x_i \in \Omega_i \end{cases}$$

x_i is the location of the panel corresponding to index i.

We can see that the subdomain deflation subspace is trivial to construct. Assuming that the 2D domain in our case is similar to that shown below, then Z will just be of the form:



$$Z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & \vdots & \vdots \\ 0 & 1 & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & 1 & 0 & \vdots \\ \vdots & 0 & 1 & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & 1 & \vdots \\ \vdots & \vdots & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2 Domain decomposition with r=4

The construction of the subdomain deflation space and its use of IDR(s) is shown in Appendix A.2.

5 Fast Multipole Method (FMM)

While the previous two sections are applicable to general matrix, this section takes a different track and focus on using special properties of matrices arising from BEM. BEM matrices have a hierarchical structure that can be exploited to speed up matrix-vector multiplication. This is important, as the limiting step in iterative solvers is the dense matrix-vector multiplication, which is O(N^2) in complexity.

Rokhlin and Greengard first introduced the Fast Multipole Method (FMM) as a way to accelerate the solutions of the Boundary Integral Equations arising from BEM in 1980s [10].

FMM allows the matrix-vector multiplication to be achieved in O(NlogN) or O(N) steps, which would significantly speed up the solver. It also greatly reduces the storage requirement, reducing it from O(N^2) to O(NlogN) [11]

This section aims to first state all the necessary ingredients involved in FMM and finally show how it could reduce the complexity of the matrix-vector multiplication from O(N^2) to O(N). A large part of this section is written with reference to [12]. The final part of this chapter provides an alternative implementation of FMM without domain or kernel information.

5.1 KERNELS

Kernels are functions of the form $K(x_i, x_j)$ that specify the coefficients of the matrix A, $a(i,j)$ [12]. As seen in Section 2, for constant element discretization, the elements of our dense Matrix A is either:

$$K(x_i, x_j) = \int_{S_j} q^*(x_i,\ x_j)dS\ \ or\ \int_{S_j} u^*(x_i,\ x_j)dS$$

For linear and higher order elements, we need to add shape functions N_α and the Jacobian J arising from coordinate transformation. The kernels are then of the form:

$$K(x_i, x_j) = \int_{S_j} q^*(x_i,\ x_j)N_\alpha|J|dS\ \ or\ \int_{S_j} u^*(x_i,\ x_j)N_\alpha|J|dS$$

Kernels that arise from BEM typically satisfies the following condition [12]:

$$\left|\partial_x^\beta \partial_y^\gamma K(x,y)\right| \le Cp!\,|x-y|^{-p}, \qquad \beta+\gamma = p, p \in \mathbb{N}$$

Equation 17

Consider the bivariate Taylor expansion of $K(x,y)$ about a point (c_σ, c_τ) [12].

$$K(x,y) = \sum_{l=0}^{p-1} \frac{1}{l!}\left[(x-c_\sigma)\partial_x + (y-c_\tau)\partial_y\right]^l K(c_\sigma - c_\tau) + R_p(x,y)$$

Equation 18

Using binomial expansion, we obtain

$$[(x-c_\sigma)\partial_x+(y-c_\tau)\partial_y]^l K(c_\sigma-c_\tau)$$
$$=\sum_{m=0}^{l}\frac{l!}{m!\,(l-m)!}\left[\partial_x^{l-m}\partial_y^{m}K(c_\sigma-c_\tau)\right]\ (x-c_\sigma)^{l-m}(y-c_\tau)^m$$

Substituting this back to Equation 18 and replacing l with $-m$, we obtained

$$K(x,y)=\sum_{m=0}^{l+m}\sum_{l=-m}^{p-1-m}\frac{1}{m!\,l!}\partial_x^l\partial_y^m K(c_\sigma-c_\tau)\ (x-c_\sigma)^l(y-c_\tau)^m+R_p(x,y)$$

$$R_p(x,y)=\sum_{l=p}^{\infty}\sum_{m=0}^{l}\frac{1}{m!\,(l-m)!}\partial_x^{l-m}\partial_y^{m}K(c_\sigma-c_\tau)(x-c_\sigma)^{l-m}(y-c_\tau)^m$$

Since $K(x,y)$ satisfy Equation 17, we therefore arrive at:

$$R_p(x,y)\le\sum_{l=p}^{\infty}\sum_{m=0}^{l}\frac{C\,l!}{m!\,(l-m)!}\frac{(x-c_\sigma)^{l-m}(y-c_\tau)^m}{|x-y|^l}=\sum_{l=p}^{\infty}C\left[\frac{(x-c_\sigma)+(y-c_\tau)}{|x-y|}\right]^l$$

Equation 19

Thus for the residual to approach zero in the limit, we require:

$$\left|\frac{(x-c_\sigma)+(y-c_\tau)}{|x-y|}\right|<1$$

Condition 20

Assuming that Condition 20 is fulfilled, we can approximate $K(x,y)$ with the following:

$$K(x,y)=\sum_{m=0}^{l+m}\sum_{l=-m}^{p-1-m}\frac{1}{m!\,l!}\partial_x^l\partial_y^m K(c_\sigma-c_\tau)\ (x-c_\sigma)^l(y-c_\tau)^m$$

Equation 21

Other expansions of $K(x,y)$ are possible. For instance, [10] uses the multipole expansion theorem.

5.2 HIERARCHICAL SPLITTING OF MATRIX A

The hierarchical splitting of Matrix A is basically the process of hierarchically partitioning A into blocks, and determining whether the blocks are admissible. To better illustrate this, consider Figure 3.

At level $l = 0$, A is not partitioned. At level $l = 1$, A is partition into 4, $l = 2$, A is partition into 16 blocks and so on until $l = n$, when each block is deemed to be small enough. We denote each block obtained from level $M_{\sigma,\tau}(l)$, where $\sigma, \tau = 0,1,\dots,2^l$.

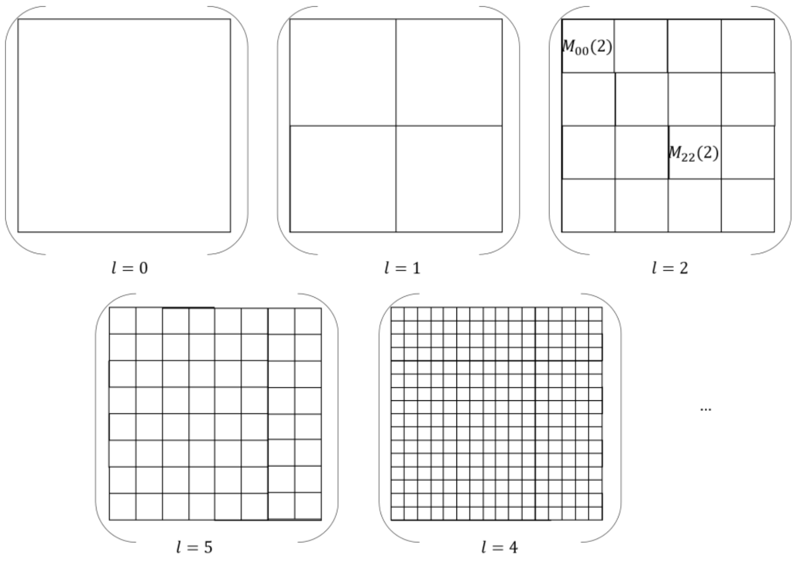


Figure 3 Hierarchical Partitioning of Matrix A

We note that all matrices derived from BEM with constant element discretization has the following structure

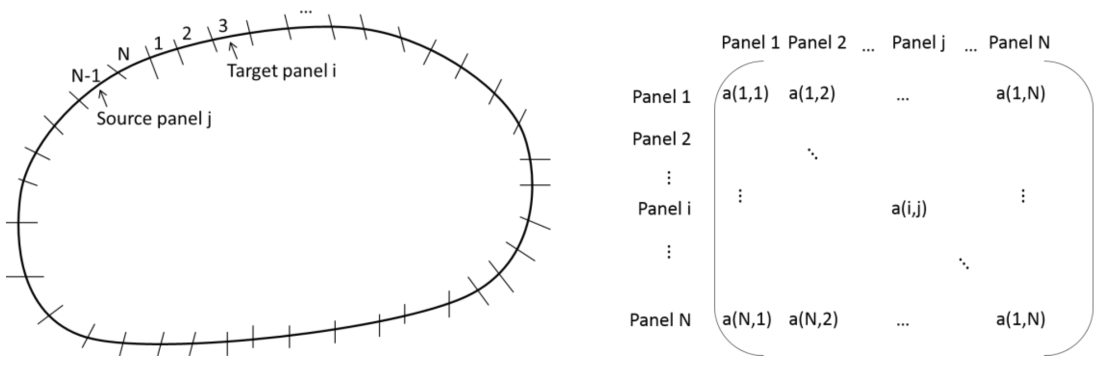


Figure 4 Discretization of Boundary with constant element and the resulting BEM matrix

As can be seen from Figure 4, the element $a(i,j) = K(x_i, x_j)$ represents the interaction between target panel i and source panel j.

Let X be the vector that contains the location of the centres of each panel. For each block $M_{\sigma,\tau}(l)$, we define the following:

(i) The centres c_σ and c_τ are defined as:

$$c_\sigma = \underset{\frac{\sigma N}{2^l} \leq i \leq \frac{(\sigma+1)N}{2^l}}{\text{average}} X(i) \,, \qquad c_\tau = \underset{\frac{\tau N}{2^l} \leq i \leq \frac{(\tau+1)N}{2^l}}{\text{average}} X(i)$$

(ii) The radius r_σ and r_τ are defined as:

$$r_\sigma = \max_{\frac{\sigma N}{2^l} \le i \le \frac{(\sigma+1)N}{2^l}} |X(i) - c_\sigma| \;, \qquad r_\tau = \max_{\frac{\tau N}{2^l} \le i \le \frac{(\tau+1)N}{2^l}} |X(i) - c_\sigma|$$

Then, we say that the block $M_{\sigma,\tau}(l)$ is admissible if:

$$\frac{r_\sigma + r_\tau}{c_\sigma + c_\tau} \le \eta, \quad \eta \in (0,1)$$

Equation 22

We note that if a block is admissible, Equation 17 is fulfilled, and the Taylor expansion for $K(x_i, x_j)$ in Equation 21 thus can be used.

Splitting A up this way results in the following, where the shaded black blocks represents admissible blocks at that level:

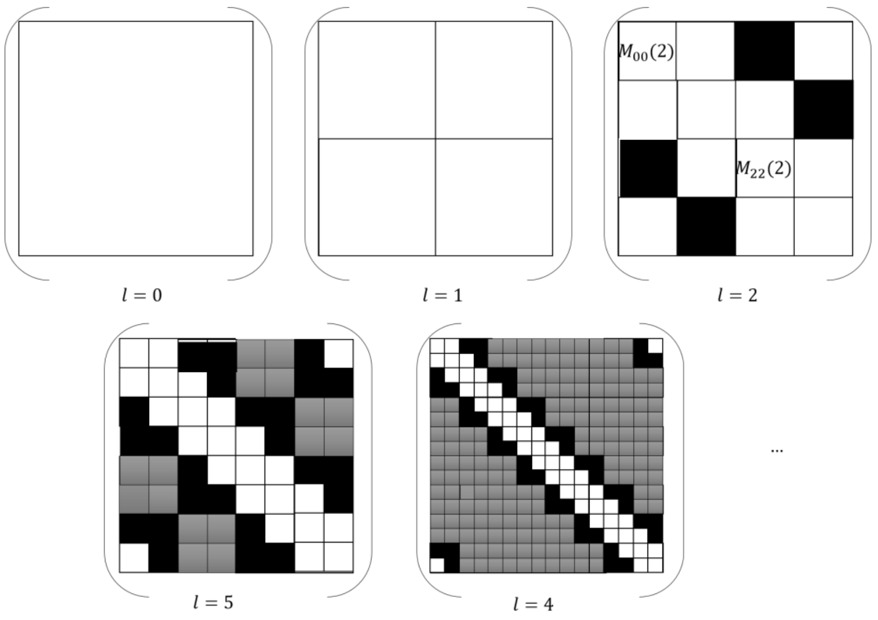


Figure 5 Hierarchical splitting of A

The remaining non-admissible (white blocks) are then referred to as the matrix N_n. Thus, for all levels up to n, we have:

$$A = \sum_{l=2}^{n} M_l + N_n$$

To have a better understanding of the hierarchical structure of the matrix, MATLAB experiment were carried out to hierarchically split the test matrices and to determine if each block is admissible. The experimental setup and results are discussed in Section 6.

5.3 Low Rank Approximation of Admissible Blocks

Consider an admissible block $M_{\sigma,\tau}(l)$. Elements within the admissible block satisfy Condition 20. We define the following:

Let the Taylor expansion of $K(x, y)$ be expanded till the $p - 1$ term. Then we define the matrix $S^{\sigma,\tau} \in \mathbb{C}^{pxp}$, with elements $(s_{l,m})$ defined as follows:

$$s_{l,m} = \begin{cases} \frac{1}{l!\,m!}\partial_x^l \partial_y^m K(c_\sigma,\ c_\tau) & if \qquad if\ 0 \le l + m \le p - 1 \\ 0 & else \end{cases}$$

Matrix $S^{\tau,\sigma}$ is an upper triangular matrix.

We also define the matrix $\Psi^\tau \in \mathbb{C}^{b_y x p}, \Psi^\sigma \in \mathbb{C}^{b_x x p}$, $b_x = block\ size\ in\ terms\ of\ rows, b_y = block\ size\ in\ terms\ of\ columns$, with elements $\left(\psi_l^{\sigma,i}\right)$ and $(\psi_m^{\tau,j})$ as follows:

$$\psi_{i\times l}^\sigma = (x - c_\sigma)^l, \quad x = X(i), \quad where\ \frac{\sigma N}{2^l} \le\ i\ \le \frac{\sigma N}{2^l} + b$$

$$\psi_{j\times m}^\tau = (y - c_\tau)^m, \quad y = X(j), \quad where\ \frac{\sigma N}{2^l} \le\ j\ \le \frac{\sigma N}{2^l} + b$$

Thus

$$\Psi^\sigma \in \mathbb{C}^{b_x \times p} = \begin{pmatrix} \psi_{1\times 1}^\sigma & \cdots & \psi_{1\times p}^\sigma \\ \vdots & \ddots & \vdots \\ \psi_{b_x\times 1}^\sigma & \cdots & \psi_{b_x\times p}^\sigma \end{pmatrix}, \qquad \Psi^\tau \in \mathbb{C}^{b_y \times p} = \begin{pmatrix} \psi_{1\times 1}^\tau & \cdots & \psi_{1\times p}^\tau \\ \vdots & \ddots & \vdots \\ \psi_{b_y\times 1}^\tau & \cdots & \psi_{b_y\times p}^\tau \end{pmatrix}$$

Equation 23

Then according to Equation 21, the block $M_{\sigma,\tau}(l)$ can be approximated as:

$$M_{\sigma,\tau}(l) \approx \widetilde{M}_{\sigma,\tau}(l) = (\Psi^\sigma) S^{\sigma,\tau} (\Psi^\tau)^T$$

Since each block can be written in this decomposed form, and all blocks with the same row shares the same Ψ^σ, while all blocks with the same column share the same Ψ^τ, we therefore have

$$M_l \approx \sum \widetilde{M}_{\sigma,\tau}(l) = blockdiag\left(\Psi^\sigma\right)_{\sigma=0,1,\ldots,2^l} S(l) blockdiag\left(\Psi^\tau\right)^T_{\tau=0,1,\ldots,2^l}$$

We define:

$$\Psi^\tau(l) = blockdiag\left(\Psi^\tau\right)_{\tau=0,1,\ldots,2^l} \in \mathbb{C}^{N\times p * 2^l}$$

$$\Psi^{\sigma}(l) = blockdiag(\Psi^{\sigma})_{\sigma=0,1,\dots,2^{l}} \in \mathbb{C}^{N\times p*2^{l}}$$

$$S(l) = matrix\ where\ (S^{\sigma,\tau})is\ inserted\ at\ the\ position\ of\ the$$

$$admissible\ block\ in\ the\ original\ matrix \in \mathbb{C}^{p*2^{l}\times p*2^{l}}$$

Thus,

$$M_l \approx \sum \widetilde{M}_{\sigma,\tau}(l) = \Psi^{\tau}(l)S(l)\left(\Psi^{\sigma}(l)\right)^{T}$$

Equation 24

5.4 CONSISTENCY CONDITION

We say that a matrix Ψ satisfy the consistency condition if [12]:

$$\Psi^{'} and\ \Psi^{''}\ are\ the\ child\ of\ \Psi$$

$$\Psi = \mathrm{C}'\Psi' + \mathrm{C}''\Psi''$$

Equation 25

$\Psi^{\sigma}, \Psi^{\tau}$ defined in Equation 23 satisfy the consistency condition. This can be shown as follows:

Consider Ψ^{σ} with elements:

$$\psi^{\sigma}_{i\times l} = (x - c_{\sigma})^{l} = \begin{cases} \left[(x - c_{\sigma'}) + (c_{\sigma'} - c_{\sigma})\right]^{l}, & x = X(i),\ \ \dfrac{\sigma N}{2^{l}} \leq\ i\ \leq \dfrac{\sigma N}{2^{l}} + \dfrac{b}{2} \\ \left[(x - c_{\sigma''}) + (c_{\sigma''} - c_{\sigma})\right]^{l}, & x = X(i),\ \ \dfrac{\sigma N}{2^{l}} + \dfrac{b}{2} \leq\ i\ \leq \dfrac{\sigma N}{2^{l}} \end{cases}$$

Using binomial expansion, we obtain

$$\psi^{\sigma}_{i\times l} = \begin{cases} \displaystyle\sum_{m=0}^{l} \frac{l!}{m!\,(l-m)!}\left(c_{\sigma'} - c_{\sigma}\right)^{l-m} \psi^{\sigma',i}_{m}, & x = X(i),\ \ \dfrac{\sigma N}{2^{l}} \leq\ i\ \leq \dfrac{\sigma N}{2^{l}} + \dfrac{b}{2} \\ \displaystyle\sum_{m=0}^{l} \frac{l!}{m!\,(l-m)!}\left(c_{\sigma''} - c_{\sigma}\right)^{l-m} \psi^{\sigma'',i}_{m}, & x = X(i),\ \ \dfrac{\sigma N}{2^{l}} + \dfrac{b}{2} \leq\ i\ \leq \dfrac{\sigma N}{2^{l}} \end{cases}$$

With $l = 0, \dots, p-1$

Therefore, we can define

$$C^{'} \in \mathbb{C}^{pxp} = \begin{cases} 0 & for\ m > l \\ \dfrac{l!}{m!\,(l-m)!}\left(c_{\sigma'} - c_{\sigma}\right)^{l-m} & for\ m \leq l \end{cases}$$

$$C^{''} \in \mathbb{C}^{pxp} = \begin{cases} 0 & for\, m > l \\ \frac{l!}{m!\,(l-m)!}\left(c_{\sigma^{''}} - c_{\sigma}\right)^{l-m} & for\, m \leq l \end{cases}$$

Where $C^{'}$ and $C^{''}$ are upper triangular. Thus, Equation 25 is obtained. The exact same process can be repeated for Ψ^{τ}.

Define

$$D_{l,l+1} \in \mathbb{C}^{2p*2^{l},p*2^{l}} = blockdiag(\begin{bmatrix} C^{'} \\ C^{''} \end{bmatrix})$$

Thus, we can construct the block diagonal matrices $\Psi^{\sigma}(l)$, $\Psi^{\tau}(l)$ from their child $\Psi^{\sigma}(l+1)$, $\Psi^{\tau}(l+1)$ using the following formula:

$$\Psi^{\sigma}(l) = (\Psi^{\sigma}(l+1))D^{\sigma}_{l,l+1}$$

$$\Psi^{\tau}(l) = (\Psi^{\tau}(l+1))D^{\tau}_{l,l+1}$$

Thus, we can rewrite $\widetilde{M}_l$ as:

$$\widetilde{M}_l = \Psi^{\tau}(l)S(l)\left(\Psi^{\sigma}(l)\right)^{T} = \Psi^{\tau}(l+1)D^{\tau}_{l,l+1}S(l)\left(\Psi^{\sigma}(l+1)D^{\sigma}_{l,l+1}\right)^{T} = \cdots$$
$$= \Psi^{\tau}(n)D^{\tau}_{n-1,n} \ldots D^{\tau}_{l+1,l+2}D^{\tau}_{l,l+1}\, S(l)\left(D^{\sigma}_{l,l+1}\right)^{T}\left(D^{\sigma}_{l+1,l+2}\right)^{T} \ldots \left(D^{\sigma}_{n-1,n}\right)^{T}\left(\Psi^{\sigma}(n)\right)^{T}$$

Equation 26

5.5 MATRIX VECTOR MULTIPLICATION

Any operation of the form Ax can now be approximated as

$$Ax \approx \sum_{l=2}^{n} \widetilde{M}_l x + N_n x$$

Where $\widetilde{M}_l$ can be decomposed using Equation 26.

The complexity of the each matrix vector multiplication is now significantly reduced. It can be carried out as follows [12]:

1. First we compute $x_n = \left(\Psi^{\sigma}(n)\right)^{T} x$

   Since $\Psi^{\sigma}(n)$ has at most pN non zero elements, the complexity of this multiplication is hence O(pN)

2. Next, for $l = n-1, \ldots, 2$, we compute $x_l = \left(D^{\sigma}_{l,l+1}\right)^{T} x_{l+1}$

Each $D^{\sigma}_{l,l+1}$ has 2^l non-zero diagonal blocks of dimension$2p \times p$. Thus, the matrix vector multiplication in this step requires $2^l \times 2p \times p$ muliplications at each step l. Adding from $l = n-1, \dots, 2$, by geometric series, we have

$$\sum_{l=2}^{n-1} 2p^2 2^l \leq 2p^2 \sum_{l=0}^{n-1} 2^l = 2p^2 \frac{2^n - 1}{1} \leq 2p^2 2^n \leq 2p^2 N$$

Thus, we see that the arithmetic complexity in this case is O(2p^2N).

3. Next for each $l = 2, \dots, n$, we compute $x_l = S(l)x_l$

   $S(l)$ contains of blocks $(S^{\sigma,\tau})$ where the corresponding block at $A^{\sigma,\tau}$ was deemed to be admissible. In total, there will be less then $2^l \times c$ such blocks at level l, where c is a constant. Each block is of size $p \times p$. Taking geometric series again, we obtain:

$$\sum_{l=2}^{n} cp^2 2^l \leq cp^2 \sum_{l=0}^{n} 2^l = cp^2 \frac{2^{n+1} - 1}{3} \leq cp^2 2^n \leq cp^2 N$$

   This, the arithmetic complexity of this step is again O(2p^2N).

4. In this step, we would try to compute $\sum_{l=2}^{n} \Psi^{\tau}(n) D^{\tau}_{n-1,n} \dots D^{\tau}_{l+1,l+2} D^{\tau}_{l,l+1} x_l$

   Note that

$$\sum_{l=2}^{n} D^{\tau}_{n-1,n} \dots D^{\tau}_{l+1,l+2} D^{\tau}_{l,l+1} x_l = D^{\tau}_{n-1,n} \dots D^{\tau}_{3,4} D^{\tau}_{2,3} x_2 + D^{\tau}_{n-1,n} \dots D^{\tau}_{4,5} D^{\tau}_{3,4} x_3 + \dots + x_n$$
$$= D^{\tau}_{n-1,n}\left(\dots D^{\tau}_{4,5}\left(D^{\tau}_{3,4}\left(D^{\tau}_{2,3} x_2 + x_3\right) + x_4\right) + \dots + x_{n-1}\right) + x_n$$

   This allows us to compute the summation by the recurrence relation:

$$x_l = D^{\tau}_{l-1,l} x_{l-1} + x_l$$

   Finally, $\sum_{l=2}^{n} \widetilde{M}_l x = \Psi^{\tau}(n) \times x_n$

   The complexity of this step can be analysed as follows. Each $D^{\tau}_{l,l+1}$ contains 2^l $2pxp$ non-zero blocks. Hence, the matrix vector multiplication requires O(2$p^2 2^l$) arithmetic steps at each level l. Using geometric series again, this requires O(2p^2N) steps. The final block-diagonal matrix multiplication is dependent on the non-zero elements of the matrix $\Psi^{\tau}(n)$. Since there is pN non zero elements in the matrix $\Psi^{\tau}(n)$, this is of O(pN),

5. Finally, we need to add $N_n x$ to the existing solution $\sum_{l=2}^{n} \widetilde{M}_l x$. N_n is a sparse matrix with at most $c \times N$ non-zero elements, c is again a constant here. This operation has arithmetic complexity O(N).

Therefore, we see that following the steps 1-5, we have reduced the O(N^2) matrix vector multiplication to O(N). We can also observe that there is no need to store the full matrix anymore. For non-admissible blocks, the kernels still have to be evaluated and stored in their respective positions in N_n. For blocks which are admissible, we can use the domain and kernel information to store $\Psi^{\tau}(l)$, $S(l)$, and $\Psi^{\sigma}(l)$ for each level l. There would be no need for the direct evaluation of the integral. Since there are less then $\log_2 N$ levels, and the dimension of each matrix is $\Psi^{\tau}(l) \in \mathbb{C}^{N \times p*2^l}$, $S(l) \in \mathbb{C}^{p*2^l \times p*2^l}$ and $\Psi^{\sigma}(l) \in \mathbb{C}^{N \times p*2^l}$, we therefore only need O(Nlog2N) of storage. Building up such matrices is also of complexity O(N) [10].

5.6 LITERATURE RESULTS

Many research papers and books have set up the BEM matrices using the above construction and had documented the results for some test cases. This section aims to highlight some of these studies done.

In [10], Liu, Y.J. presented a case study on the solving of potential problem for an annular region. In his example, conventional BEM refers to computing the integrals, forming the matrices and solving the resulting system of equation using direct solvers. Hence complexity is of order O(N^3). The FMM method he described is coupled with GMRES, by replacing the matrix-vector multiplication with the hierarchical representation. The CPU time taken to solve the potential problem in an annular region using the conventional BEM and FMM BEM is shown in Figure 6. We can observe that the CPU time is drastically reduced, especially for large matrix sizes, due to the O(N) complexity of the FMM method. In terms of accuracy, he concluded that the FMM-BEM was found to be equally accurate as the conventional BEM.

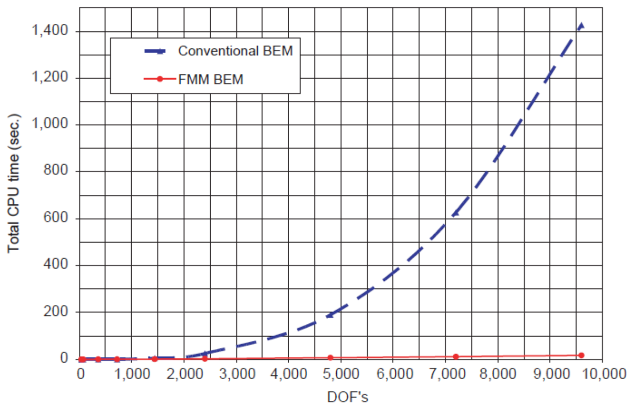
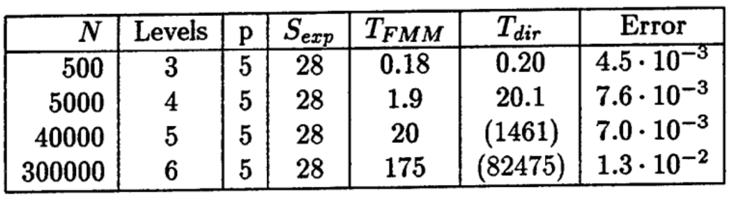


Figure 6 Comparison of the CPU time used by conventional BEM and FMM-BEM [10]

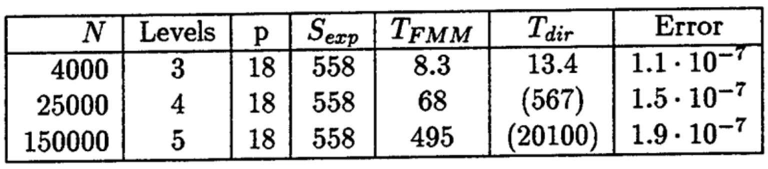
In 1996, Greengard L. & Rokhlin V. also published results for using the FMM to solve the laplace equation in three dimension [13]. The application was to evaluate the pairwise interactions in N-body simulations. An example of a numerical results obtained is extracted and shown in Figure 7.



N	Levels	p	S_{exp}	T_{FMM}	T_{dir}	Error
500	3	5	28	0.18	0.20	$4.5 \cdot 10^{-3}$
5000	4	5	28	1.9	20.1	$7.6 \cdot 10^{-3}$
40000	5	5	28	20	(1461)	$7.0 \cdot 10^{-3}$
300000	6	5	28	175	(82475)	$1.3 \cdot 10^{-2}$

Figure 7 Timing results for the FMM using fifth order expansions and 28 exponential basis functions [13]

T_{FMM} represents the time taken using the FMM method, and T_{dir} represents the time taken using the direct method. We can see that as the number of element grows, the FMM method results in significant cost savings, with reasonable error. The error can be reduced by increasing p, the order of expansions of the approximation, an the exponential basis functions.



N	Levels	p	S_{exp}	T_{FMM}	T_{dir}	Error
4000	3	18	558	8.3	13.4	$1.1 \cdot 10^{-7}$
25000	4	18	558	68	(567)	$1.5 \cdot 10^{-7}$
150000	5	18	558	495	(20100)	$1.9 \cdot 10^{-7}$

Figure 8 Timing results for the FMM using 18th order expansions and 558 exponential basis functions

With p increased to 18, we see that the error reduces to 1e-8, with the time taken still within reasonable limits.

5.7 Alternative without kernel or domain information

The discussion so far has been based on the assumption that domain and kernel information is readily available, and the kernel satisfies the constraints as given in Section 5.1. In the event that such information is not available, it is still possible to reduce the matrix vector multiplication to a complexity of O(NlogN), through low rank matrix approximation using the Lanczos Bidiagonalization process.

From Section 5.1 to 5.5, we have built an approximation for admissible block of the form given in Equation 24, where $S(l)$ is upper triangular. It turns out that each admissible block can also be represented using low rank matrix approximation. Performing the Singular Value Decomposition (SVD) of the admissible block reveals that only a small number of singular values are significantly large. Many singular values are closed to zero, and therefore can be dropped. We arrive at the following low rank approximation of the admissible block:

$$\widetilde{M}_{\sigma,\tau}(l) \approx U\Sigma V^H$$

However, the use of SVD is not practical, since it is computational expensive. The Lanczos bi-diagonalization algorithm provides an alternative. It performs partial bi-diagonalization decomposition of a matrix, such that when terminated at step p, it returns the first p columns of the bidiagonal matrix B, and the orthonormal matrices U and V. The following sub-sections would describe the Lanczos bi-diagonalization algorithm and the use of this algorithm to hierarchically divide and obtain the low rank approximations of the test matrices.

5.7.1 Lanczos bi-diagonalization

Consider the reduction of $A \in \mathbb{C}^{mxn}$ into bidiagonal form:

$$A\,[v_1 \quad |\ldots| \quad v_n] = [u_1 \quad |\ldots| \quad u_m] \begin{bmatrix} \alpha_1 & \beta_1 & & \\ & \alpha_2 & \ddots & \\ & & \ddots & \beta_{n-1} \\ & & & \alpha_n \end{bmatrix}$$

Let $U = [u_1 \quad |\ldots| \quad u_m]$ and $V = [v_1 \quad |\ldots| \quad v_n]$. U and V are required to be orthogonal matrices. Also, let B be the bidiagonal matrix $\begin{bmatrix} \alpha_1 & \beta_1 & & \\ & \alpha_2 & \ddots & \\ & & \ddots & \beta_{n-1} \\ & & & \alpha_n \end{bmatrix}$.

We can then write, for the kth column:

$$Av_k = \beta_{k-1}u_{k-1} + \alpha_k u_k, \text{ or}$$

$$\alpha_k u_k = Av_k - \beta_{k-1} u_{k-1}$$

Equation 27

Since $AV = UB$, then $A^H U = VB^H$ must be also true. Hence, we obtained:

$$A^H \begin{bmatrix} u_1 & |\dots| & u_m \end{bmatrix} = \begin{bmatrix} v_1 & |\dots| & v_n \end{bmatrix} \begin{bmatrix} \alpha_1 & & & \\ \beta_1 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_{n-1} & \alpha_n \end{bmatrix}$$

Which provides us with the formula for the kth column:

$$A^H u_k = \alpha_k v_k + \beta_k v_{k+1}, \quad \text{or}$$

$$\beta_k v_{k+1} = A^H u_k - \alpha_k v_k$$

Equation 28

Thus, if we specify any unit vector v_0 and assume β_0 is 0, then Equation 27 and 28 form the recurrence to obtain u_k and v_{k+1} at every step k. The two scalers α_k and β_k are chosen to normalize u_k and β_k. This algorithm is described in many literature, for example [14].

The algorithm is modified such that the decomposition is terminated after p steps. Hence, the dimensions of the decomposed matrix are:

$$U \in \mathbb{C}^{mxp}, \qquad V \in \mathbb{C}^{nxp}, \qquad B \in \mathbb{C}^{pxp}$$

5.7.2 Checking admissibility criteria with Lanzcos Bi-diagonalization

Again, let each block in the hierarchical division of A be $M_{\sigma,\tau}(l)$. Applying the Lanzcos bi-diagonalization algorithm to $M_{\sigma,\tau}(l)$ gives us an approximation $\widetilde{M}_{\sigma,\tau}(l) = UBV^H$. To check if $M_{\sigma,\tau}(l)$ is admissible, we can consider the singular values of $M_{\sigma,\tau}(l)$, which can be obtained by a SVD of B.

$$\widetilde{M}_{\sigma,\tau}(l) = UU_B \Sigma V_B V^H$$

By the theory of SVD, if the singular values in $\Sigma(p,p)$ has dropped to below a user specified tolerance tol, we can deemed the block as admissible [15].

However, performing SVD of B is a step which we would like to avoid. Through careful observation of the diagonal elements of B and comparing it with the singular values Σ, it was found that $diag(B)$ closely approximates Σ. For instance, a plot of the $diag(B)$ elements and

Σ is shown in Figure 9 from one of the admissible blocks of the FATIMA_7894 matrix at level 2.

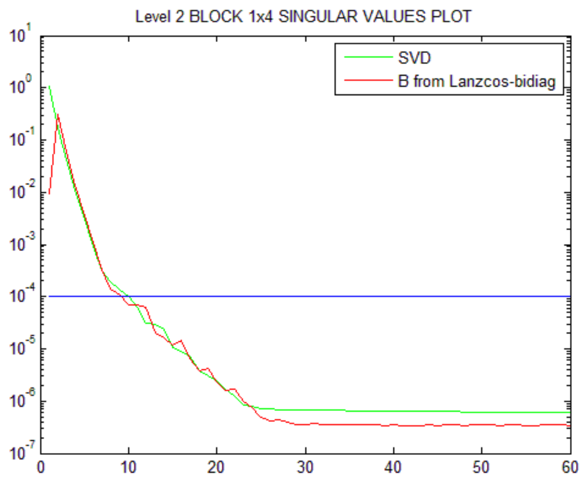


Figure 9 Comparison between singular values and diag(B) from Lanczos Bi-diagonalization of an admissible block of the FATIMA_7894 matrix

Similar plots were also generated for the rest of the admissible blocks for all test matrices, and the same trend was observed. Hence, although in general, the diagonal elements of B will not always be close to the singular values, for our matrices, this seems to hold.

Therefore, instead of computing the SVD of B additionally, admissibility criteria would be based instead on B(p,p), the pth diagonal element from B derived from Lanczos Bi-diagonalization.

5.7.3 Matrix Vector Multiplication

By decomposing the matrix A using the Lanczos Bi-diagonalization procedure this way, we are not able to obtain the block diagonal form as given in Section 5.3, and we also lose the consistency conditions that would make the matrix vector multiplication O(N). The matrix vector multiplication in implemented in the following way instead.

Consider $\widetilde{M}_l x$. At level l, A would be partitioned into $2^l \times 2^l$ blocks. Let each admissible block be represented by $\widetilde{M}_{\sigma,\tau}(l)$, where σ represents the block number row-wise, and τ represents the block number column wise, and $1 \leq \sigma \leq 2^l, 1 \leq \tau \leq 2^l$.

$$\widetilde{M}_{\sigma,\tau}(l) = U_{\sigma,\tau} B_{\sigma,\tau} V_{\sigma,\tau}^H$$

Next, partition x with respect to the column partition of $\widetilde{M}_l$, and $\widetilde{M}_l x$ with respect to the row partition of $\widetilde{M}_l$. Each partition of x is represented by x_τ, and each partition of $\widetilde{M}_l x$ by $\left(\widetilde{M}_l x\right)_\sigma$

Then the multiplication can be performed as follows:

$$\widetilde{M}_l x = \begin{bmatrix} (\widetilde{M}_l x)_1 \\ \vdots \\ (\widetilde{M}_l x)_{2^l} \end{bmatrix} = \begin{bmatrix} \sum_{\tau=1}^{2^l} \widehat{U}_{1,\tau} B_{1,\tau} \widehat{V}_{1,\tau}^H x_\tau \\ \vdots \\ \sum_{\tau=1}^{2^l} \widehat{U}_{2^l,\tau} B_{2^l,\tau} \widehat{V}_{2^l,\tau}^H x_\tau \end{bmatrix}$$

MATLAB experiments were done to hierarchically divide the test matrices using Lanzcos Bi-diagonalization and to compare the time taken to do the matrix vector multiplication using the hierarchical matrices with the normal vector multiplication. The results are presented in the next section.

6 MATLAB EXPERIMENTS

6.1 SYSTEM INFORMATION

All tests are run on the following system, with MATLAB R2013b

Brand/Type	DELL
Owner/ System no.	TU DELFT/ TUD205717
CPU	Intel® Core™ i5-4670 CPU @ 3.40GHz
No. of cores	4
Cache	256 KB x 4 L2/ 6 MB Smart L3
Memory	8 GB RAM DDR3-1333/1600
Motherboard	Dell 0PC5F7
Operating System	Windows 7
System Kernel	
GPU	Intel® HD Graphics 4600
Memory	1696 MB
No of cores	20

Table 2 System Information

6.2 COMPARISON IN PERFORMANCE BETWEEN GMRES AND IDR(S)

As discussed in Section 3, solvers under consideration are GMRES and IDR(s). We will also include the Bi-CG method here for discussion purpose.

The test matrix used in this experiment would be the Steadycav matrices (See Section 2 for the full list of test matrices). Only 1 matrix, Steadycav1 is used, since the results are not expected to differ significantly. The Fatima matrices took too long to be solved in MATLAB using pure solvers without preconditioners, and therefore, would not be considered in this particular experiment. The result are presented below:

Matrix		**GMRES**	**IDR(10)**	**IDR(60)**	**Bi-CG**
Steadycav1 N=4620	**Timing (s)**	5.4	5.9	5.1	16
	Iteration	237	379	278	490

Table 3 Results for MATLAB experiment - Comparison across solvers

We can see from this experiment that IDR(s) is comparable with GMRES. Although the number of iterations is significantly larger for IDR(10) (379) when compared with GMRES (237), the difference in timing is not as significant. This is due to the short recurrence property of IDR(s) method. When compared with Bi-CG, we see that much more iterations are required for convergence, and also due to the 2 matrix-vector multiplication requirement, Bi-CG incurs significantly more time as compared with the other two solvers.

We can also conclude from this experiment that for the IDR(s) algorithm, s could be tuned to improve performance. A larger value of s would imply that the depth of recurrence is higher, thus more work per step needs to be carried out, and more memory is required. On the other hand, it would also mean that the number of iterations required is reduced, since the maximum number of iterations is $N + \frac{N}{s}$ for IDR(s).

In conclusion, GMRES and IDR(s) would be considered for our application. Preconditioners would next be applied.

6.3 PERFORMANCE OF BLOCK JACOBI AND DEFLATION PRECONDITIONER

In this section, we will investigate the effects of applying the block Jacobi and the deflation preconditioners discussed in Section 4. The preconditioned system is solved using both GMRES and IDR(10). The results are presented in Table 4. The best results in terms of the time and number of iterations required for solving is highlighted in green.

We observe first that the application of the block jacobi preconditioner improved the time and number of iteration required for convergence significantly. For the Fatima_7894 matrix, the number of iterations falls from 3982 to 72 for GMRES after preconditioning by block Jacobi. Although no convergence was observed for IDR(10), we attained convergence at iteration number 84 by preconditioning the system with block Jacobi. In fact, without preconditioning, the system could not solve the FATIMA_20493 matrix in reasonable time.

Next, we see that a great advantage of the IDR(10) can be observed when considering the FATIMA_20493 matrix. For smaller systems, like the FATIMA_7894 and below, the performance of IDR(10) is on par when compared to GMRES (with a time difference of 1 seconds and below). However, for large matrix like the FATIMA_20493, IDR(10) achieved convergence at a timing that is half of that for GMRES, due to the short recurrence relation.

Finally, for all test cases, no improvement was observed when subdomain deflation preconditioner was applied. To understand the reason why, the convergence plots for the block Jacobi preconditioned system was studied for all test matrices except FATIMA_20493[4]. This can be seen in Figure 10. We can observe that the convergence behavior is linear, which indicates that there are no isolated eigenvalues that are causing slow convergence. The convergence behavior seems to be determined by all eigenvalues fairly equally.

Next, we study the eigenvalue plots of the test matrices, presented in Figure 11. From the plots, we do observe some small eigenvalues clustered near 0. This brings to the question of whether deflation subspaces built up by the eigenvectors corresponding to the smallest eigenvalues will be more suitable in this case. The experiment was thus repeated with deflation using eigenvectors. This is shown in the last column of Table 4. We see that there is indeed an improvement in the number of iterations required for convergence. For the Steadycav matrices, the number of iteration is reduced by half, while the reduction for the FATIMA_7894 matrix is less significant. This agrees with the eigenvalue plots, since there are less small eigenvalues clustered near zero for the FATIMA_7894 matrix. However, the time taken to build up the eigenvectors need to be improved. This will be part of the plan for subsequent work.

[4] Results for FATIMA_20493 was not included because it would take too long to run

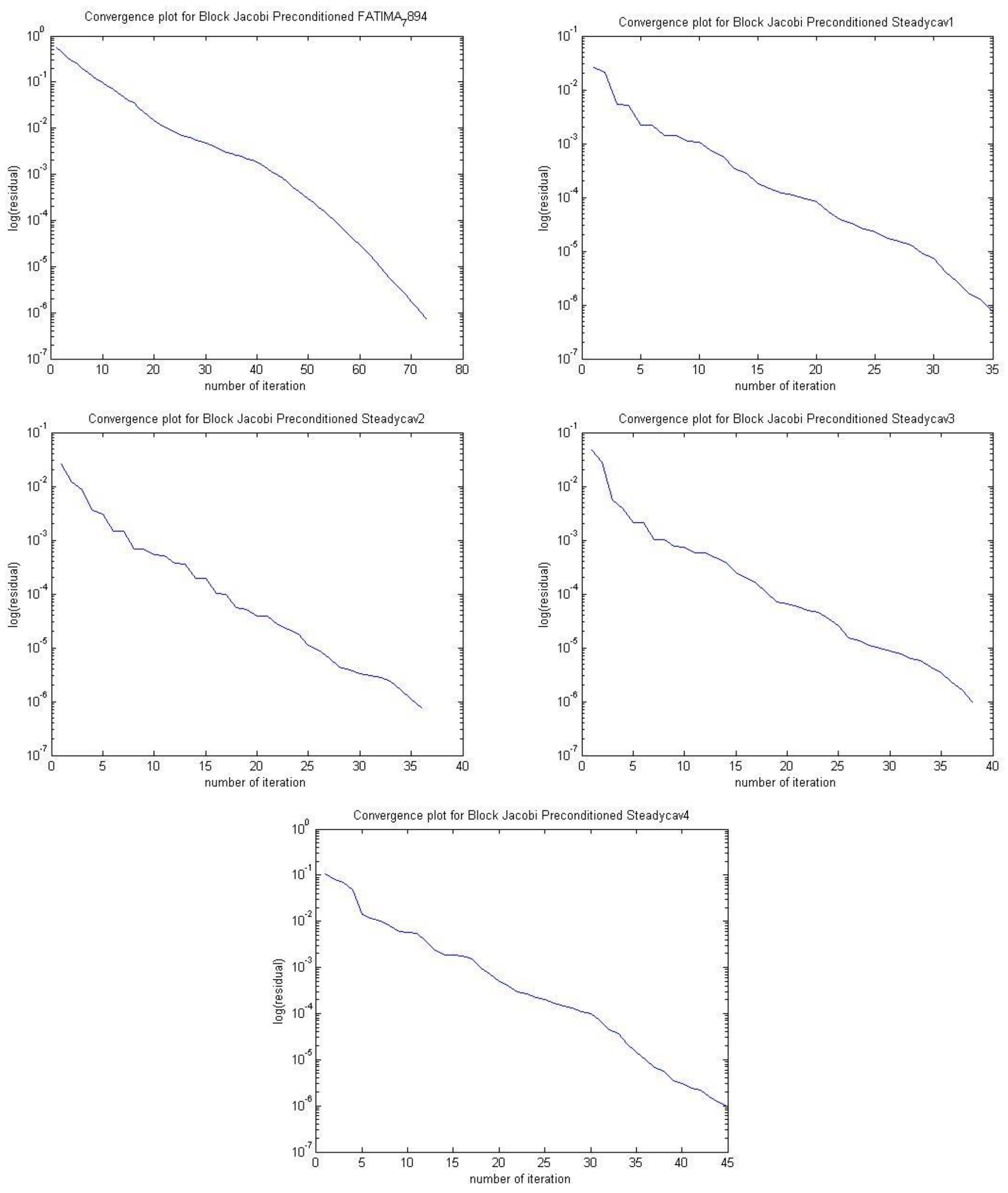


Figure 10 Convergence plots for block Jacobi preconditioned test matrices

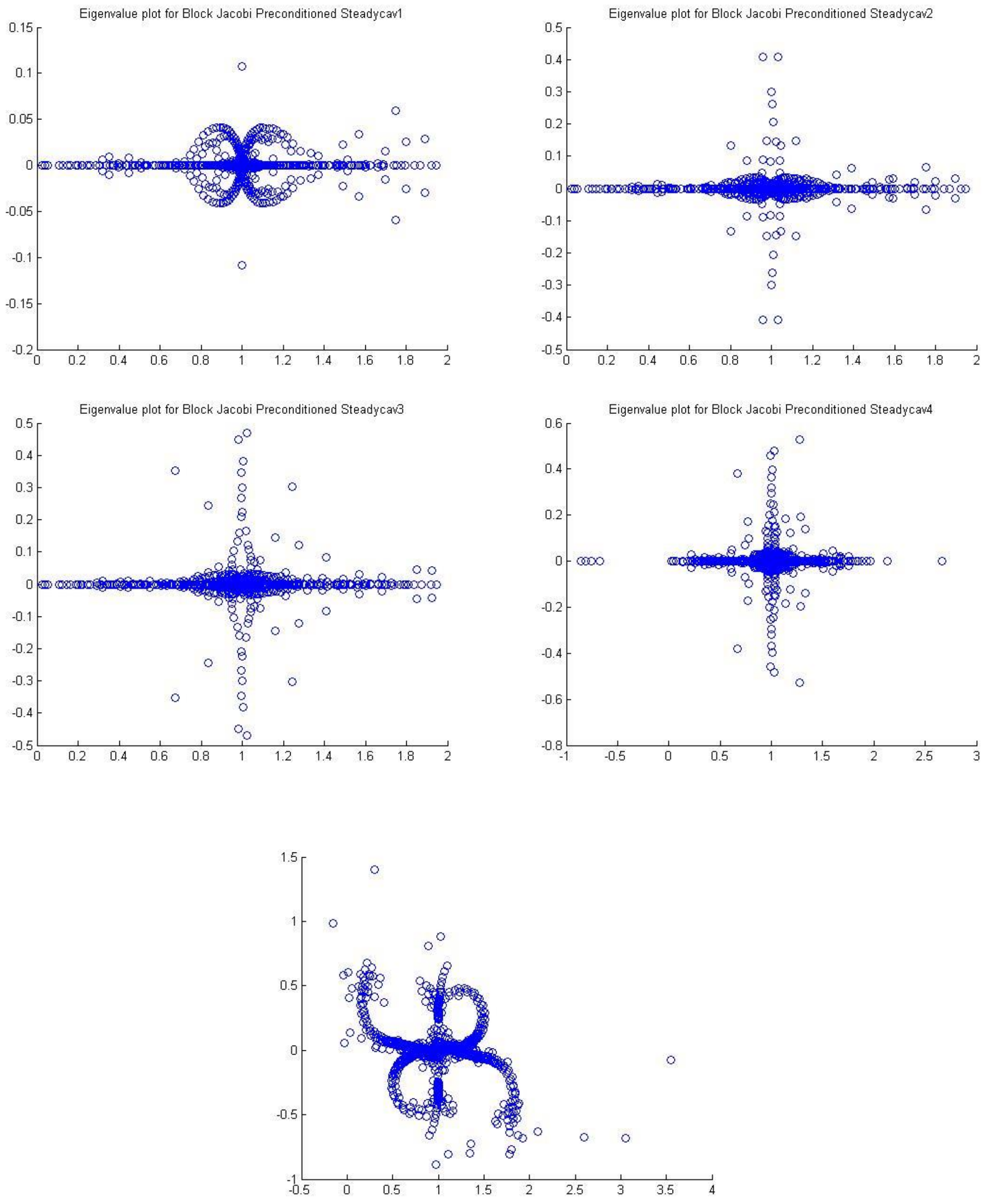


Figure 11 Eigenvalue plots for block Jacobi preconditioned test matrices

	Matrix	GMRES	GMRES with block jacobi[5]	GMRES with subdomain deflation[6]	IDR(10)	IDR(10) with block jacobi	IDR(10) with subdomain deflation	IDR(10) with eigenvector deflation[7]
Timing for preconditioning (s)	Steadycav1 N=4620	N.A	0.68	0.66 BJ[8] 0.16 D[9]	N.A	0.65	0.69 BJ 0.13 D	0.69 BJ 2.25 D
Timing for solving (s)		5.9	0.61	0.47	5.5	0.49	0.44	0.29
Total time (s)		5.9	1.29	1.29	5.5	1.13	1.26	3.23
Iteration		323	35	36	515	45	41	24
Timing for preconditioning (s)	Steadycav2 N=4620	N.A	0.66	0.67 - BJ 0.17 - D	NA	0.7	0.68 BJ 0.14 D	0.67 BJ 2.21 D
Timing for solving (s)		17.4	0.44	0.5	19.8	0.5	0.49	0.26
Total time (s)		17.4	1.12	1.34	19.8	1.20	1.31	3.14
Iteration		624	36	36	1918	45	46	23
Timing for preconditioning (s)	Steadycav3 N=4620	N.A	0.68	0.67 BJ 0.16 D	N.A.	0.68	0.67 BJ 0.14 D	0.67 BJ 2.29 D
Timing for solving (s)		19.6	0.48	0.48	18.5	0.51	0.51	0.30
Total time (s)		19.6	1.16	1.31	18.5	1.19	1.32	3.26
Iteration		656	38	37	1817	47	47	25
Timing for preconditioning (s)	Steadycav4 N=4649	N.A.	0.70	0.49 BJ 0.17 D	N.A	0.70	0.70 BJ 0.14 D	0.71 BJ 2.72 D
Timing for solving (s)		18.0	0.55	0.51	24.4	0.58	0.55	0.41

[5] No parallelization. 5 blocks
[6] Number of domain r = 5
[7] 5 eigenvectors corresponding to the 5 smallest eigenvalues are used (If eigenvectors are complex, only the real parts are used)
[8] Indicates block Jacobi precondition time
[9] Indicates subdomain deflation precondition time

Total time (s)		18.0	1.25	1.17	24.4	1.28	1.39	3.84
Iteration		622	45	40	2341	54	52	36
Timing for preconditioning (s)	FATIMA 7894 N=7894	N.A	7.9	8.0 BJ 1.18 D	N.A	7.7	8.0 BJ 1.1 D	8.0BJ 52.5D
Timing for solving (s)		26904	9.3	8.96	N.A	10.1	10.2	7.8
Total time (s)		26904	17.2	18.14	1798.5 (no converge nce)	17.8	19.3	68.3
Iteration		3982	73	72	15788	84	86	66
Timing for preconditioning (s)	FATIMA 20493 N=20493[10]		5000+			5000+		
Timing for solving (s)			1948			894.3		
Total time (s)			7000+			6000+		
Iteration			96			115		

Table 4 Results for Comparison across preconditioners

[10] Only two sets of results were collected for the FATIMA_20493 as the time required to operate with a matrix of this size is too long on our system.

6.4 MATLAB EXPERIMENT FOR THE HIERARCHICAL SPLITTING OF EXISTING TEST MATRICES

This experiment aims to highlight the hierarchical structure inherent in matrices generated through BEM. Each test matrix will be hierarchically divided into admissible low rank blocked matrix M_l and the final sparse non-admissible matrix N_n.

For this experiment, a total of 4 levels would be considered (n=4). SVD would be used to determine the rank of the blocks. It is noted that SVD is not feasible as a way to construct the low rank approximation.

Using SVD, the matrix A would be recursively divided into blocks as follows:

- For each Matrix M, divide matrix M into 4 blocks, M_1, M_2, M_3, M_4
- For each block M_i, use SVD to decompose the block and count the number of singular values (p) that are more than a threshold value (1e^{-4}). Below this threshold value, we can assume that the singular value is approximately 0.
- Blocks which have p<60 (which is just another arbitrary threshold value) is admissible.
- Blocks which have p≥60 are inadmissible. If $l \neq 4$, recursively split M_i. If $l = 4$, the block is stored in its position in non-admissible matrix N_n

We have the following results for our test matrices: Steadycav1, Steadycav2, Steadycav3, Steadycav4 and Fatima_7894.

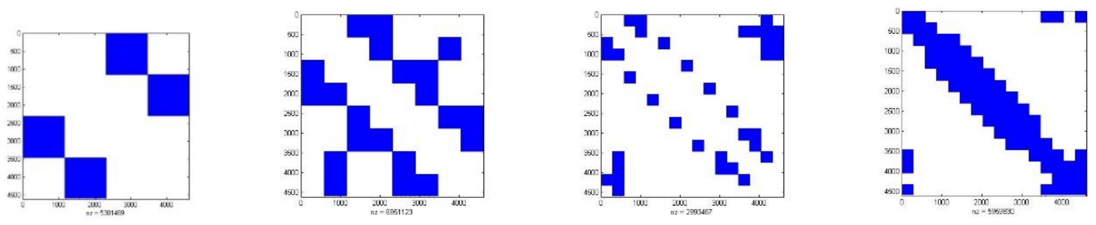


Figure 12 M2, M3, M4 and N4 of Steadycav1 matrix

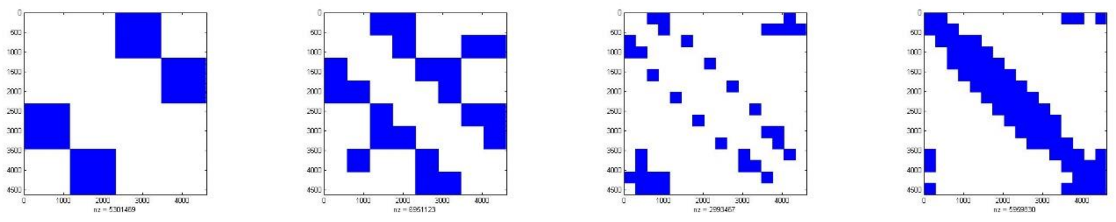


Figure 13 M2, M3, M4, N4 for Steadycav2 matrix

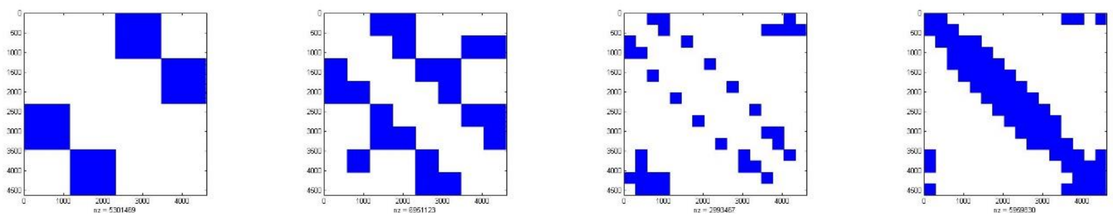


Figure 14 M2, M3, M4, N4 of Steadycav3 matrix

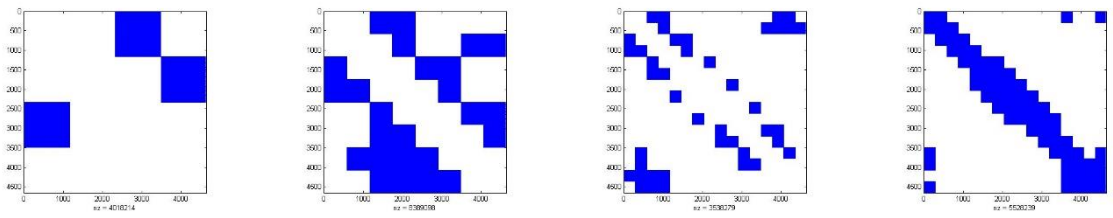


Figure 15 M2, M3, M4, N4 of Steadycav4 matrix

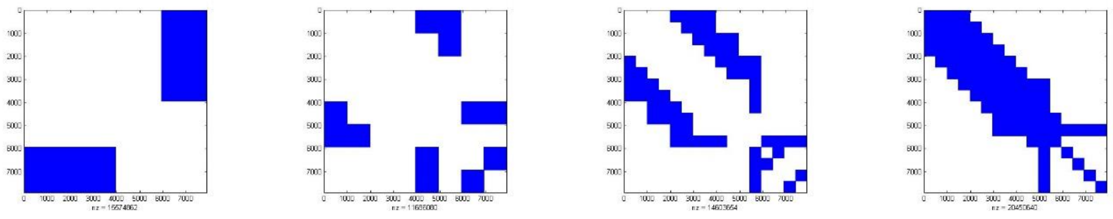


Figure 16 M2, M3, M4, N4 of FATIMA_7894 matrix

The hierarchical structure of the test matrices can be seen clearly in all cases. A significant number of blocks are low rank, with the full rank blocks located near the main diagonal. After removing all the low rank blocks, the non-admissible matrix N_4 turns out to be sparse. From this, we can also expect that the final non-admissible matrix can also be used as a preconditioner to speed up convergence.

6.5 MATLAB EXPERIMENT FOR FMM USING LANCZOS BI-DIAGONALIZATION

The hierarchical splitting algorithm with lanczos bi-diagonalization was applied to the test matrices. The parameters recorded are the time taken to perform hierarchical splitting, t_{split}, and the time taken for matrix vector multiplication in the hierarchical form, t_{m-hie}. t_{m-hie} is compared with the time taken for matrix vector multiplication in the full matrix form t_{m-full}.

For the measurement of t_{m-full}, instead of using the matrix vector multiplication in MATLAB (A*x), for a fair comparison, for loops would be used instead to compute the matrix vector multiplication at this stage. This is to prevent the optimization in MATLAB from clouding the results. For implementation details, please refer to Appendix A.3.

The experiment was repeated for different values of block size b and rank of admissible block p (as described in section 5.9.2). Block size b determines the level of recursion for the hierarchical division. In view of the sizes of our test matrices, two b values would be tested out here, b=100 and b=200. When b gets lower than about 50, t_{split} increased without a significant reduction in t_{m-hie}. If b is too big (above 200), there is no significant improvement of t_{m-hie}, since the blocks considered are too big, hence, there are few admissible blocks. Four values of p are chosen, p=10, 20, 35 and 50.

The accuracy is measured by the difference between the matrix vector multiplication Av obtained from hierarchical form, and the full matrix form, in the 2-norm. With an admissibility requirement of 10^{-5} (see section 5.9.2), all difference is in the order of 10^{-4}.

The storage requirement for the full matrix, and that of the matrix in hierarchical form is summarized in Table 6.

The MATLAB code for the recursive hierarchical division of a matrix using lanczos bidiagonalization is provided in Appendix A.4. The code to perform hierarchical matrix multiply is given in Appendix A.5.

The results are shown below.

		Steadycav1				Steadycav2				Steadycav3			
b	p	t_{m-full}	t_{m-hie}	% reduction	t_{split}	t_{m-full}	t_{m-hie}	% reduction	t_{split}	t_{m-full}	t_{m-hie}	% reduction	t_{split}
100	10	0.043	0.034	20.93%	1.35	0.043	0.035	18.60%	1.31	0.044	0.035	20.45%	1.31
	20		0.028	34.88%	2.22		0.03	30.23%	2.24		0.029	34.09%	2.24
	35		0.024	44.19%	3.79		0.025	41.86%	3.64		0.025	43.18%	3.63
	50		0.024	44.19%	5.07		0.024	44.19%	5.01		0.024	45.45%	5.06
200	10	0.034	0.032	5.88%	0.91	0.034	0.032	5.88%	0.91	0.033	0.032	3.03%	0.92
	20		0.0286	15.88%	1.5		0.025	26.47%	1.49		0.025	24.24%	1.48
	35		0.02	41.18%	2.34		0.02	41.18%	2.41		0.02	39.39%	2.37
	50		0.018	47.06%	3.32		0.019	44.12%	3.29		0.019	42.42%	3.33

		Steadycav4				FATIMA_7894			
b	p	t_{m-full}	t_{m-hie}	% reduction	t_{split}	t_{m-full}	t_{m-hie}	% reduction	t_{split}
100	10	0.044	0.035	20.45%	1.34	0.3	0.14	53.33%	11.03
	20		0.03	31.82%	2.38		0.12	60.00%	18.7
	35		0.023	47.73%	3.87		0.13	56.67%	31.46
	50		0.023	47.73%	5.17		0.12	60.00%	48.6
200	10	0.034	0.035	-2.94%	0.93	0.25	0.15	40.00%	9.4
	20		0.027	20.59%	1.56		0.13	48.00%	15.01
	35		0.02	41.18%	2.4		0.11	56.00%	24.33
	50		0.018	47.06%	3.38		0.11	56.00%	33.25

Table 5 Results from FMM using lanczos bidiagonalizarion – matrix vector multiplication timing

	Steadycav1	Steadycav2	Steadycav3	Steadycav4	FATIMA_7894
Storage requirement for full matrix	**0.17 GB**	**0.17 GB**	**0.17 GB**	**0.17 GB**	**0.99 GB**
Storage requirement in hierarchical form (b=100, p=50) [11]	N: 0.086 GB B: 3.71e-4 GB U: 0.038 GB V: 0.037 GB **Total: 0.16**	N: 0.085 GB B: 3.74e-4 GB U: 0.038 GB V: 0.037 GB **Total:0.16**	N: 0.084 GB B: 3.81e-4 GB U: 0.038 GB V: 0.038 GB **Total: 0.16**	N: 0.08 GB B: 3.82e-4 GB U: 0.04 GB V: 0.039 GB **Total: 0.16**	N: 0.56 GB B: 5.28e-4 GB U: 0.078 GB V: 0.077 GB **Total: 0.72 GB**
Decrease in storage required	**5.88%**	**5.88%**	**5.88%**	**5.88%**	**27.3%**

Table 6 Results from FMM using lanczos bidiagonalizarion – storage requirements

[11] All matrixes are stored in the sparse form in MATLAB, otherwise, there would not be a reduction in size, since MATLAB would store zeros.

First and foremost, we see the drastic improvement in the time taken for matrix vector multiplication in the hierarchical form as opposed to the standard dense matrix form. The results with the most significant improvement is highlighted in green for each test matrix. For the smaller PROCAL matrices, a percentage reduction of ~45% was attained, and for the FATIMA_7894, a percentage reduction of 60% was achieved. An even larger percentage reduction can be expected for FATIMA_20943.

We can draw an important observation from the comparison across the various p values chosen. It is important to ensure that p closely approximate the "rank" of the admissible blocks. If p was chosen too high, work is wasted on unnecessary lanczos bi-diagonalization iterations and the matrix-vector multiplication (as well as storage). However, if p was chosen too low, the final non-admissible matrix N_n would be too dense, since most blocks would be deemed inadmissible.

With regards to this, the ranks of the admissible blocks of the test matrices were analyzed. Figure 17 shows a histogram plot depicting rank versus the number of admissible blocks at level 5 for the Steadycav3 matrix. The ranks of the admissible block can be approximated by the number of singular values that are above a certain threshold (for illustration, 1e-4 was chosen as the threshold value in the figures). Figure 18 shows the same plot, but for the FATIMA_7894 matrix.

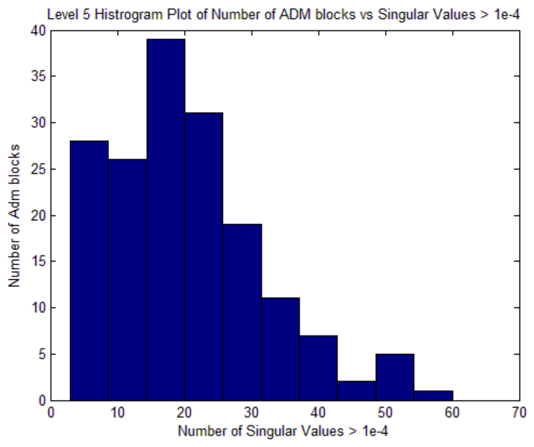


Figure 17 Histograms plot of admissible blocks vs rank (reflected by the number of singular values not closed to 0) for Steadycav3 at level 5

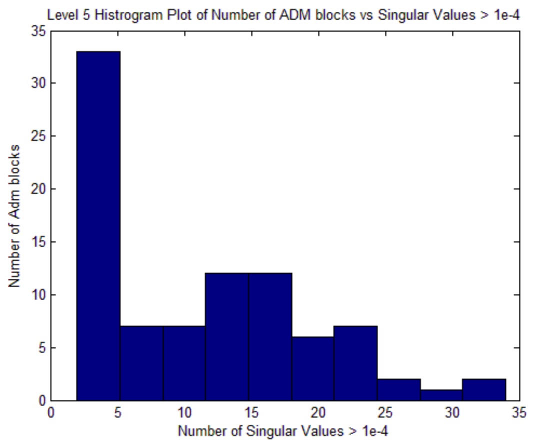


Figure 18 Histograms plot of admissible blocks vs rank (reflected by the number of singular values not closed to 0) for FATIMA_7894

It was observed that for the matrices arising from the PROCAL system (Steadycav1, Steadycav2, Steadycav3 and Steadycav4), majority of the admissible blocks have rank approximately 35. This can be deduced by the histogram plot for Steadycav3. The other test matrices from the PROCAL system yield similar plots at all levels. Very few admissible blocks have ranks lower than 30. Hence, should p be chosen lower then 30, the resulting N_n would be dense, and there would be no improvement in time for the matrix vector multiplication. This explains why for the PROCAL test matrices, the timing is drastically reduced when p changes from 20 to 35. Increasing p to 50 does reduce the time taken for matrix vector multiplication. However, the increase in time needed to do the hierarchical splitting may not justify the reduction.

On the other hand, the FATIMA_7894 matrix displays a much lower rank as compared to the PROCAL test matrices. A good proportion of the admissible blocks have rank less than 10. Hence, the improvement in the percentage reduction when p is increased to 20, 35 or 50 is minor. For example, when b=100, the percentage reduction is already 53% when p=10. Increasing p to 20 brings the reduction to 60%, but increases t_{split} by 7s. Subsequent increased in p does not improve t_{m-hie}, but increases t_{split} significantly.

In general, b=200 seems to be a better choice than b=100. The percentage reduction attained is approximately the same, but t_{split} takes less time since the level of recusion is lower.

Turning our attention to the storage requirements, the comparison between the storage required for the full matrix, and the matrix in hierarchical form is shown in Table 6. We note that for the smaller Steadycav matrix, putting the matrix in hierarchical form gives a storage savings of 6%, or around 0.01 GB. For the FATIMA_7894, a storage saving of 27.3% is attained, which amounts to 0.27 GB. It can be expected that the storage savings attainable for the FATIMA_20493 matrix will be even higher.

In conclusion, we can that FMM can indeed speed up the matrix multiplication, and reduce storage requirements significantly. If kernel and domain information can be successfully exploited, t_{split} and t_{m-hie} can be expected to be much lower, since the complexity order is O(n) as highlighted in Section 5, while using the lanczos bidiagonalization brings the complexity to O(n^2) for t_{split} and O(nlogn) for t_{m-hie}. However, it is to be noted that full matrix vector multiplication can be parallelized and optimized to reduce the time required significantly. For example, when we use MATLAB inbuilt matrix vector multiplication to compute Av, where A is the FATIMA_7894 matrix, the time taken is only 0.064s. This is so much lower than the 0.25s obtained in our experiment. As there is inherently many small matrix vector multiplications in the hierarchical method, there could be a need to further optimized or parallelized these to achieve significant time savings from optimized full matrix vector multiplication implementations.

7 CONCLUSION AND PLAN FOR SUBSEQUENT THESIS WORK

From the literature review and the MATLAB experiments conducted, the following conclusions can be drawn:

1. IDR(s) shows the potential to significantly reduce the solving time especially for large matrix size. This is evident from the results of the FATIMA_20493 of Section 6.3. We see that IDR(10) took half the time as GMRES to solve the preconditioned problem.
2. Deflation with eigenvectors corresponding to the smallest eigenvalues has the potential to further speedup convergence. Work should be carried out to reduce the time required to construct the eigenvectors.
3. Hierarchical methods exhibit great potential to reduce the time and storage required to solve the BEM problem. However, much work has to be done to incorporate this method with the BEM codes already established in MARIN, and optimize the code for practical uses.
4. The use of lanczos bidiagonalization algorithm provides an alternate way to exploit the hierarchical structure without domain or kernel information. It can be implemented as a standalone, and then integrated with the existing BEM codes. Work

has to be done to more efficiently implement the admissibility checking algorithm, and the matrix vector multiplication, to make it competitive with existing highly optimized code for matrix vector multiplications.

With these findings, there are a few tracks recommended for the subsequent thesis work

1. To implement IDR(s) with block Jacobi. Project will focus on implementation and optimization in FORTRAN.
2. To further optimize the use of deflation with eigenvectors. Primary focus would be on reducing the time required to construct the eigenvectors required.
3. To explore the use of Graphic Processing Unit (GPU) to further reduce computational time.
4. To work on the Hierarchical method implementation using lanzcos bidiagonalization. This would concentrate on coding in FORTRAN, and optimizing so that it provides an improvement over the current matrix vector multiplication implementation available in FORTRAN linear algebra libraries.
5. To work on the Hierarchical method implementation using domain and kernel information. This would require access to the BEM codes currently used by MARIN and the integration is expected to be of higher risk.

8 REFERENCES

1. de Jong, M. & van der Ploeg, A. 2012. *Efficient Solvers for Panel Codes.* MARIN Report No. 21750-2-RD.
2. Kythe, K.P. 1995. *An Introduction to Boundary Element Methods.* Boca Raton, CRC Press.
3. Trefethen, L,N. & Bau, D. 1997. *Numerical Linear Algebra.* Philadelphia, SIAM.
4. Sonneveld, P. & van Gijzen, M.B. 2008. *IDR(s): A Family of Simple and Fast Algorithms for Solving Large Nonsymmetric Stsrems of Linear Equations.* SIAM Journal for Scientific Computing Vol. 31, No. 2, pp. 1035-1062.
5. Saad, Y. 2003. *Iterative Methods for Sparse Linear Systems: 2nd Edition*.
6. Coulaud, O., Giraud, L., Ramet, P. & Vasseur, X. 2013. *Deflation and Augmentation Techniques in Krylov Subspace Methods for the Solution of Linear Systems.* INRIA, Research Report N° 7003.
7. Erlangga, Y.A. & Nabben, R. 2008. *Deflation and Balancing Preconditioners for Krylov Subspace Methods Applied to Nonsymmetric Matrices.* SIAM J. Matrix Analysis Applications, Vol. 30, No. 2, pp. 684-699.
8. Chapman, A. & Saad, Y. 1996. *Deflated and Augmented Krylov Subspace Techniques.* Numerical Linear Algebra Applications, Vol. 4, Issue. 1, pp. 43-66.
9. Frank, K. & Vuik, C. 2001. *On the Construction of Deflation-Based Preconditioners.* SIAM Journal of Scientific Computing, Vol. 23, Issue. 2, pp. 442-462.
10. Liu, Y. 2009. *Fast Multipole Boundary Element Method: Theory and Applications in Engineering.* Cambridge, Cambridge University Press.
11. Hackbusch, W. & Nowak, Z.P. 1989. *On the Fast Matrix Multiplication in the Boundary Element Method by Panel Clustering.* Numerische Mathematik, Vol. 54, Issue 4, pp. 463-491.
12. Fenn, M. & Steidl, G. 2002. *FMM and H-Matrices: a Short Introduction to the Basic Idea.* Preprint Universität Mannheim (2002).
13. Greengard, L. & Rokhlin, V. 1996. *A New Version of the Fast Multipole Method for the Laplace Equation in Three Dimensions.* Research Report Yale University Department of Computer Science/ DCS/ RR-1115

14. Demmel, J. 2000. *Iterative Algorithms: Golub-Kahan-Lanczos Method*. In Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., & van der Vorst, H., editors. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide.* Philadelphia, SIAM.

15. Horst D.S., & Zha, H.Y. 2000. *Low Rank Approximation using the Lanczos Bidiagonalization Process with Applications.* SIAM Journal of Scientific Computing, Vol. 21, No. 6, pp. 2257-2274.