

Research Internship (PRe)

Field of Study: Applied Maths
Scholar Year : 2023-2024

Hierarchical Matrices and Inexact GMRES

Confidentiality Notice
Non-confidential and publishable report

Author: GUIMARAES Eduardo **Promotion: 2025**

ENSTA Paris Tutor: FARIAS Luiz
MARCHAND Pierre
Host Organism Tutor: CHAPOUTOT Alexandre

Internship from 13/05/2024 to 15/08/2024

POEMS
Address: 828, Boulevard des Maréchaux, 91762 Palaiseau France

This page was intentionally left blank.

Confidentiality Notice

This present document is not confidential. It can be communicated outside in paper format or distributed in electronic format.

Abstract

In this project we will focus on combining a recently develop inexact GMRES algorithm, which differs from the classic GMRES algorithm in that the underlying linear system is allowed to change at each iteration by using a hierarchical matrix approximation to the boundary integral discretisation. The main goal is to (i) understand the inner workings of GMRES and how changing the linear system at each iteration affects the convergence properties, and (ii) explore whether inexact GMRES has practical interest when combined with a variable precision $\mathcal{H}$-matrix compression. This project was carried out in the context of two existing libraries: HMatrices.jl and Inti.jl. The internship took place at the POEMS laboratory, and was supervised by Luiz M. Faria (chercheur INRIA) and Pierre Marchand (chercheur INRIA).

Keywords— GMRES, BEM, applied maths, acceleration, Julia

Acknowledgments

I am deeply grateful to all those who have contributed to the successful completion of this research project during my internship at UMA and the POEMS group.

First and foremost, I would like to express my sincere gratitude to Mr. Farias and Mr. Marchand for providing me with this opportunity. The guidance provided during the project allowed me to work in a field that interests me very much and that I intend to pursue after my stay at ENSTA Paris. I also extend this gratitude for ENSTA Paris and UMA, who granted the funding of the project and made everything possible.

I would also like to extend my heartfelt thanks all students and staff of UMA that I have had the opportunity of meeting and/or working with. Even though I'm far away from home, the environment they created always made me feel welcomed and thankful for the support I received. A special mention goes to the students I had the pleasure of sharing a room with, and whose company helped shape this amazing experience.

Finally, I wish to thank my family and friends, both at Brazil but also the ones I got to meet here in France and are already considered as a part of my family.

Thank you all for your invaluable contributions to this research.

Contents

List of Figures

Chapter 1

Iterative Methods and Krylov's Subspace

1.1 Iterative Methods and motivation

We consider the solution of a linear system, with $A \in \mathbb{C}^{n \times n}$ an invertible matrix, $b \in \mathbb{C}^n$ our right-hand side and $x \in \mathbb{C}^n$ a vector of unknowns:

$$Ax = b \tag{1.1}$$

A system that can be solved by direct methods, like *Gaussian elimination* and *LU decomposition*. The problem with such approaches comes with their complexity: direct methods usually have $\mathcal{O}(m^3)$ complexity [13], where m is the dimension of the input matrix. Since we apply larger matrices in practice, to achieve high resolution in the solution of PDEs, the direct algorithms become too slow, and we require a more efficient approach.

These iterable methods find, after a certain number of iterations, a sequence x_k that converges to x, the exact solution of the problem 1.1. This should be done while making the large-scale computations faster, i.e., obtaining a complexity smaller than $\mathcal{O}(m^3)$, and keeping a maximum tolerance between the iterable solution and the exact one.

$$x = \lim_{k \to \infty} x_k \tag{1.2}$$

The method stops after k iterations, where x_k is the fist element of the sequence to satisfy the condition 1.3.

$$\frac{||Ax_k - b||}{||b||} \leq \epsilon \tag{1.3}$$

We define ϵ as the tolerance given to the algorithm.

To achieve a smaller complexity than $\mathcal{O}(m^3)$, Iterative Methods employ matrix vector products, complexity $\mathcal{O}(m^2)$, instead of the product between matrices found in direct methods. So, considering an Iterative Method finds a solution in k steps, its complexity would be $\mathcal{O}(km^2)$.

Therefore, if $k << m$, i.e., the number of iterations is sufficiently smaller than m, the iterative methods become more efficient than the direct ones.

The method we employ in our problems is the Generalized Minimal Residuals(GMRES), explained later in the report. Its main idea involves the projection of a high dimensional problem, as large as A in 1.3, in a lower dimensional *Krylov Space*:

$$x_k \in \text{span}(b, Ab, A^2b, ..., A^{k-1}b) \tag{1.4}$$

Explained in more detail below.

1.2 Krylov subspace

Let $A \in \mathbb{C}^{n\times n}$ a matrix and $b \in \mathbb{C}^n$. For each $k \leq n$ the Krylov subspace $\mathcal{K}_k = \mathcal{K}_k(A, b)$ associated to A, b is defined as 1.5.

$$\mathcal{K}_k(A, b) = \text{span}(b, Ab, A^2b, \ldots, A^{k-1}b) \tag{1.5}$$

These subspaces also have the following property: $k < l \rightarrow \mathcal{K}^k \subset \mathcal{K}^l$ [3].

The subspace $\mathcal{K}_k(A, b)$ is also the subspace of all the vectors from $\mathbb{R}^m$ which could be written as $x = p(A)b$, where $p(A)$ is a polynomial of degree less than $k - 1$ which $p(0) = 1$.

The problem with using $A^kb, k \in 0, 1, 2, \ldots$ as a base comes from the fact that successive products of A make vectors that are *approximately collinear*, since those are really close of the eigenvector with the largest eigenvalue of A.

Such problem then requires finding a new base for this space, one that does not suffer from the power iteration of A, and it is the subject of the next section.

1.3 Arnoldi's Method

Arnoldi's Method is an orthogonal projection method used to find an orthonormal basis $q_1, \ldots, q_k$ to $\mathcal{K}_k(A, b)$.

An algorithm for the method can be found in 1.

Algorithm 1 Arnoldi's iteration

1: $A \in \mathbb{K}^{n\times n}$ et $b \in \mathbb{K}^n$
2: $x = 0, \beta = \|b\|, q_1 = \frac{b}{\beta}$
3: **for** $j = 1, 2, \ldots k$ **do**
4: $\quad q_{j+1} = Aq_j$
5: $\quad$ **for** $i = 1, 2, \ldots j$ **do**
6: $\quad\quad h_{ij} = q_{j+1}^t q_i$
7: $\quad\quad q_{j+1} = q_{j+1} - h_{ij}q_i$
8: $\quad$ **end for**
9: $\quad h_{j+1,j} = \|q_{j+1}\|$
10: $\quad q_{j+1} = \frac{q_{j+1}}{h_{j+1,j}}$
11: **end for**

As we can see, at each step in 1, the previous vector q_j is multiplied by A and then orthonormalized in relation to all previous q_i's with a Gram-Schmidt procedure. If q_{j+1} ever vanishes during the inner loop between lines 5 and 8, the algorithm stops.

What is left is to show the q_i generated by 1 form an orthonormal basis for $\mathcal{K}_k(A, b)$.

Proposition 1. *Assume that 1 does not stop before the k-th step. Then the vectors $q_1, q_2, \ldots, q_k$ form an orthonormal basis of the Krylov subspace $\mathcal{K}_k(A, b)$*

$$\mathcal{K}_k(A, b) = \text{span}\{b, Ab, A^2b, \ldots, A^{k-1}b\}$$

Proof. By construction q_j, $j = 1, 2, \ldots, k$ are orthonormal. To show they span $\mathcal{K}_k(A, b)$ we prove q_j has the form $p_{j-1}(A)b$, where $p_j(A)$ is a polynomial of degree $j - 1$ in A. Using induction the result is true for $j = 1$ since $q_1 = b$. We assume the result is true for all integers $\leq j$ and consider q_{j+1}. Using the definition of q_{j+1} in 1 we have:

$$h_{j+1,j}q_{j+1} = Aq_j - \sum_{i=1}^{j} h_{ij}q_i = Ap_{j-1}(A)b - \sum_{i=1}^{j} h_{ij}p_{i-1}(A)b \tag{1.6}$$

Since, by the induction step above, $q_i = p_{i-1}(A)b$.
This shows q_{j+1} can be written as $p_j(A)b$ and completes the proof. □

We also make note of the fact $q_1 = \frac{b}{||b||}$.
Next we prove the relation between A and the Hessenberg's matrix defined by 1.

Proposition 2. *Denote by Q the $n \times k$ matrix with column vectors $q_1, \ldots, q_k$ found in 1 and H_k the $(k+1) \times k$ Hessenberg matrix whose nonzero entries h_{ij} are given just as in 1. Then the following relation holds 1.7.*

$$AQ_k = Q_{k+1}H_k \tag{1.7}$$

Proof. For each column-vector of Q, q_i, 1.7 could be written as 1.8, where the representation of $\mathcal{K}_k(A, b)$ with an orthonormal basis becomes more evident.

$$Aq_m = h_{1m}q_1 + h_{2m}q_2 + \ldots h_{m+1,m}q_{m+1} \tag{1.8}$$

This relation can be directly seen in 1 by using line 10 and the inner loop between lines 5 and 8:

$$\begin{aligned} q_{m+1}h_{m+1,m} &= Aq_m - \sum_{i=1}^{m} h_{im}q_i \\ Aq_m &= \sum_{i=1}^{m+1} h_{im}q_i \end{aligned} \tag{1.9}$$

□

This method can also be used to find one(or some) of the eigenvalues of A, through the *Rayleigh-Ritz method* [13].

By 1.7, H_k is a Hessenberg Matrix of dimensions $k+1 \times k$, usually a modest size, with its eigenvalues can be computed more efficiently. These, known as *Ritz eigenvalues*, typically converge to the largest eigenvalues of A.

Chapter 2

GMRES

As mentioned above, GMRES is the iterative method chosen to solve the linear system in 1.1. This will mainly be done by approximating, at each iteration k, the solution x to an element in the lower dimensional Krylov subspace $\mathcal{K}_k(A, b)$, A being the matrix associated to the linear system and b its right-hand side.

Defining the residual of iteration k as:

$$r_k = \|b - Ax_k\| \tag{2.1}$$

Our x_k, the $k-th$ approximation to x, if chosen as the solution of the minimization problem of r_k. Since we make the projections in $\mathcal{K}_k(A, b)$, $x_k = K_k y_k$, our solution is given by the problem:

$$\min_{x \in \mathcal{K}_k(A,b)} \|b - Ax\| = \min_{y \in \mathbb{C}^n} \|b - AK_k y\| \tag{2.2}$$

Where K_k is the Krylov matrix with columns equal to the vectors that span the Krylov subspace of the current iteration:

$$K_k = \left[b, Ab, A^2b, \ldots, A^kb\right] \tag{2.3}$$

But as said in the previous chapters,$b, Ab, \ldots, A^kb$ has approximately collinear vectors, and we choose the base from the Arnoldi's Method instead.

Then, we take a projection in $\mathcal{K}_k(A, b)$, where we take the different approximations as in 2.4, where Q_m is the vector in 1.7.

Using the projections Q_k from 1.7:

$$x_k = x_0 + Q_k y_k \tag{2.4}$$

With 2.4 and 1.7 the residual becomes 2.5, where $x_0 = 0$, $\beta = \|b\|$ and $Q_{k+1}^t b = (\|b\| \, 0 \, 0 \ldots)^t$ since the columns of Q_{k+1} are orthonormal vectors and $q_1 = \frac{b}{\|b\|}$.

$$\begin{aligned} r_k &= \|b - Ax_k\| \\ &= \|b - A(Q_k y_k)\| \\ &= \|b - Q_{k+1} H_k y_k\| \\ &= \left\|Q_{k+1}(Q_{k+1}^t b - H_k y_k)\right\| \\ &= \|\beta e_1 - H_k y_k\| \end{aligned} \tag{2.5}$$

Thus, y_k which appears in 2.4, is found as the solution of the residual's minimization problem in 2.5:

$$y_k = \arg\min_{y \in \mathbb{C}^k} \|\beta e_1 - H_k y\| \tag{2.6}$$

An initial version of the GMRES is in 2. The lines 4 to 12 contain the Arnoldi's Method presented in 1.

Algorithm 2 Initial GMRES

1: $A \in \mathbb{K}^{n\times n}$ and $b \in \mathbb{K}^n$
2: $x = 0, \beta = \|b\|, q_1 = \frac{b}{\beta}$
3: **for** $k = 1, 2, \ldots$ **do**
4: **for** $j = 1, 2, \ldots k$ **do**
5: $q_{j+1} = Aq_j$
6: **for** $i = 1, 2, \ldots j$ **do**
7: $h_{ij} = q_{j+1}^t q_i$
8: $q_{j+1} = q_{j+1} - h_{ij} q_i$
9: **end for**
10: $h_{j+1,j} = \|q_{j+1}\|$
11: $q_{j+1} = \frac{q_{j+1}}{h_{j+1,j}}$
12: **end for**
13: Find $y = \min_y \|\beta e_1 - H_m y\|$
14: $x = Q_k y$
15: **Stop** if the residual is smaller than the tolerance
16: **end for**

However, 2 doesn't present an efficient way of finding the residual in each iteration. To solve this problem and also to find a more efficient way of solving the least squares in 2.6, we apply a transformation to H_k, turning it into a triangular matrix.

2.1 Givens's Rotation

Givens's operator, $G(i, i+1)$, is a unitary matrix such that the column vector $a = Gb$ has the elements $a(i) = r \in \mathbb{R}$ and $a(i+1) = 0$. It has a structure as in 2.7. The coefficients c_i, s_i only appear in the rows i et $i+1$.

$$G(i,i+1) = \begin{bmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & c_i & s_i & & & \\ & & & -s_i & c_i & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{bmatrix} \tag{2.7}$$

This operator offers a way to transform the columns in H_m, *zeroing* the elements outside the main diagonal. Since a product of unitary operators is still unitary, 2.6 can be written as 2.8, where R_m and g_m are the results from the application of multiple Givens's operators to H_m and βe_1.

$$y = \arg\min_y \|\beta e_1 - H_m y\| = \arg\min_y \|g_m - R_m y\| \tag{2.8}$$

Thus, the new problem 2.8 can be solved with a simple backwards substitution. If $g_m = [\gamma_1 \dots \gamma_{m+1}]^t$, an $m+1$ column vector, and $\{R_m\}_{ij} = r_{ij}$ an $m+1$ by m upper triangular matrix with $r_{ii} \neq 0$ and its last row filed with zeros, each element of $y_m = [y_1 \dots y_m]$ is given by 2.9.

$$\begin{aligned} \gamma_k &= \sum_{i=k}^{m} r_{ki} y_i \\ y_m &= \frac{\gamma_m}{r_{mm}} \\ y_i &= \frac{1}{r_{ii}} \left(\gamma_i - \sum_{j=i+1}^{m} r_{ij} y_j \right) \end{aligned} \tag{2.9}$$

A simple algorithm to this end can be written as 3.

Algorithm 3 Backwards substitution

1: $A \in \mathbb{K}^{n \times n}, \{A\}_{ij} = a_{ij}$ and $b \in \mathbb{K}^n$
2: **for** $k = n, n-1, \dots$ **do**
3: $\quad y_k = b_k$
4: $\quad$ **for** $j = n, n-1, \dots k+1$ **do**
5: $\quad\quad y_k = y_k - a_{kj} y_j$
6: $\quad$ **end for**
7: $\quad y_k = \frac{y_k}{a_{kk}}$
8: **end for**

It can be shown that g_m also contains the residual of each iteration [11].

Proof. Since it's an $m+1$ column vector, we have, with Ξ_m being the necessary Givens's Rotations to make H_m upper triangular 2.10.

$$\|b - Ax_m\| = \left\|Q_{m+1}^t(\beta e_1 - H_m y_m)\right\| = \|\beta e_1 - H_m y_m\| = \left\|\Xi_m^t(g_m - R_m y_m)\right\| \tag{2.10}$$

And since Ξ_m^t is a rotation matrix, its unitary and $\left\|\Xi_m^t(g_m - R_m y_m)\right\| = \|g_m - R_m y_m\|$. For any vector y, as the last line in 2.10 is field with zeros:

$$\|g_m - R_m y_m\|_2^2 = |\gamma_{m+1}|^2 + \|(g_m)_{1:m} - (R_m)_{1:m,1:m}(y_m)_{1:m}\|_2^2 \tag{2.11}$$

Since in the minimization problem, the second term of the right side becomes zero(a triangular system), the residual has its value as $|\gamma_{m+1}|$, which gives a more efficient way to obtain the residuals during each iteration. □

2.2 Inexact GMRES

The most expensive part in the code is in the matrix-vector product 1, line 4. Therefore, one approach to accelerate the iterations involves an approximation of Aq, instead of using the exact product, as shown in 2.12.

$$\mathcal{A}q = (A + E)q \tag{2.12}$$

Where E in 2.12 is a *perturbation matrix* that changes with each iteration and will be written as E_k for iteration k.

When we execute the inexact matrix-vector product, instead of the regular one, the left side of 1.7 must be changed by 2.13.

$$\begin{aligned}[(A+E_1)q_1,(A+E_2)q_2,\ldots,(A+E_k)q_k] &= Q_{k+1}H_k \\ (A+\mathcal{E}_k)Q_k &= Q_{k+1}H_k, \quad \mathcal{E}_k = \sum_{i=1}^{k} E_i q_i q_i^t \\ \mathcal{A}_k Q_k &= W_k\end{aligned} \tag{2.13}$$

Where $W_m = Q_{m+1}H_m$ from this point forward.

Now the subspace spawn by the vectors of Q_k is not the Krylov's subspace $\mathcal{K}_k(A,b)$, but these are still orthonormal. To see what kind of subspace our new Q spans, 2.13 is looked into in 2.14.

$$(A+E_k)q_k = \mathcal{A}_k q_k = h_{1,k}q_1 + h_{2,k}q_2 + \cdots + h_{k+1,k}q_{k+1} \tag{2.14}$$

For $k=1$, we have that q_2 is a combination of the vectors $\mathcal{A}_1 b$ and b (since $q_1 = \mathrm{b}$). For $k=2$ we see that q_3 is a combination that involves $\mathcal{A}_2\mathcal{A}_1 b$ and so forth.

Expression 2.13 then shows that Q_k becomes a basis for a new Krylov's subspace, $\mathcal{K}_k(A+\mathcal{E}_k,b) = \mathrm{span}\{b,\mathcal{A}_1 b,\ldots,\mathcal{A}_k\ldots\mathcal{A}_1 b\}$, made by a large perturbation in A, that gets updated in each iteration.

A new distinction should also be made between the two types of residuals appearing in the process. We denote by r_k the exact residual of an iteration, defined as $r_k = \|b - Ax_k\|$, but too expensive to calculate every time. And $\tilde{r}_k = \|b - AQ_k y_k\|$, the one that will really be calculated throughout the algorithm. This is analogous to what was done in the conventional GMRES, where the residual being computed each time was $\|b - AQ_k y_k\| = \|\beta e_1 - H_k y_k\|$, the difference being that now these two different residuals are **not** the same, as we will see below in 2.15.

$$\begin{aligned} r_k &= r_0 - AQ_k y_k \\ &= r_0 - (Q_{k+1}H_k - [E_1 q_1,\ldots,E_k q_k])y_k \\ &= \tilde{r}_k + [E_1 q_1,\ldots,E_k q_k]y_k \\ \rightarrow \delta_k &= \|r_k - \tilde{r}_k\| = \|[E_1 q_1,\ldots,E_k q_k]y_k\| \end{aligned} \tag{2.15}$$

Considering $y_k = [\eta_1^{(k)}\ldots\eta_k^{(k)}]$, upper index to clarify the iteration, an upper bound for δ_k can be found, but before we go through 2.16.

$$\begin{aligned} \|[E_1 q_1,\ldots,E_k q_k]y_k\| &= \left\|\sum_{i=1}^{k} E_i q_i \eta_i^{(k)}\right\| \\ \left\|\sum_{i=1}^{k} E_i q_i \eta_i^{(k)}\right\| &\leq \sum_{i=1}^{k} \|E_i\| \left\|q_i \eta_i^{(k)}\right\| \\ \left\|\sum_{i=1}^{k} E_i q_i \eta_i^{(k)}\right\| &\leq \sum_{i=1}^{k} \|E_i\| \left|\eta_i^{(k)}\right| \end{aligned} \tag{2.16}$$

We use the fact that q_i are orthonormal between the last two lines. The bound on δ_k is then found in 2.17.

$$\delta_k = \|r_k - \tilde{r}_k\| \leq \sum_{i=1}^{k} \|E_i\| \left\|\eta_i^{(k)}\right\| \tag{2.17}$$

2.17 tells us that in order to keep both residuals close, either the perturbation of A, somewhat measured by $\|E_i\|$, or the elements of y_i should be kept small. Since we expect to use more *relaxed* approximations of A as the iterations go on, a greater tolerance in E_k could be compensated with a sufficiently small y_k.

The problem is y_k is only found after the construction of E_k, so an upper bound must be also found for its value.

We consider now $\Xi_k = G(k, k+1)G(k-1, k)\dots G(1, 2)$, where each G represents a Givens rotation as shown in 2.7. So Ξ_k is the unitary matrix(since all Givens rotations are themselves unitary) that transforms H_k into an upper triangular matrix.

Since $\|H_k y - \beta e_1\| = \|\Xi_k(H_k y - \beta e_1)\|$, the application of Ξ_k in the norm $\|H_k y - e_1\beta\|$ transforms the problem into a minimization of a triangular linear system with non-zero diagonal elements in its coefficient matrix:

$$y_k = \arg\min_{y\in\mathbb{C}^k} \|\Xi_k H_k y_k - \Xi_k e_1\beta\| = \arg\min_{y\in\mathbb{C}^k} \|R_k y_k - g_k\| \tag{2.18}$$

Knowing y_k is the solution of this triangular system, that can be solved by backwards substitution, we have 2.19:

$$R_k y_k = g_k \tag{2.19}$$

Both sides of the equation above are column vectors of $k+1$ elements, since we are still considering R_k with its last line filled with zeros. If we denote $\tilde{R_k} = (R_k)_{1:k,1:k}$ and $\tilde{g_k} = (g_k)_{1:k}$, 2.19 can also be written as:

$$y_k = {\tilde{R_k}}^{-1}\tilde{g_k} \tag{2.20}$$

Since $\tilde{R_k}$, the transformation of a Hessenberg matrix by a series of Givens rotations, is upper triangular and has no line full of zeros, then its inverse also is upper triangular. Being an upper triangular matrix, the first $i-1$ elements of its i-th line are zeros, so using Matlab index notation in 2.21.

$$({\tilde{R_k}}^{-1})_{i,1:k}\tilde{g_k} = ({\tilde{R_k}}^{-1})_{i,i:k}\tilde{g_k} \tag{2.21}$$

Using this last result in 2.20 gives 2.22.

$$\begin{aligned} |\eta_i^{(k)}| &= \left\|({\tilde{R_k}}^{-1})_{i,i:k}(g_k)_{i:k}\right\| \\ |\eta_i^{(k)}| &\leq \left\|e_k{\tilde{R_k}}^{-1}\right\| \|(g_k)_{i:k}\| \\ |\eta_i^{(k)}| &\leq \left\|e_k{\tilde{R_k}}^{-1}\right\| \|(g_k)_{i:k}\| \end{aligned} \tag{2.22}$$

Since $\left\|e_k{\tilde{R_k}}^{-1}\right\| \leq \left\|{\tilde{R_k}}^{-1}\right\| = \sigma_k(H_k)^{-1}$ and $\|(g_k)_{i:k}\| \leq \|\tilde{r}_{i-1}\|$ [12], the bound is given by 2.23.

$$\left\|\eta_i^{(k)}\right\| \leq \frac{1}{\sigma_k(H_k)} \|\tilde{r}_{i-1}\| \tag{2.23}$$

Using both results of 2.23 and 2.17 gives the results in 2.24. Thus, a sufficient condition for $\delta_k \leq \epsilon$ is given by 2.25.

$$\delta_k \leq \sum_{i=1}^{k} \frac{\|E_i\|}{\sigma_k(H_k)} \|\tilde{r}_{i-1}\| \tag{2.24}$$

$$\|E_i\| \leq \frac{\sigma_k(H_k)\epsilon}{k\,\|\tilde{r}_{i-1}\|} \tag{2.25}$$

Since H_k is also one of the matrices constructed throughout the method, a workaround is necessary to apply these bounds in a practical situation. Either using an estimation of $\sigma_k(H_k)$ with the singular values of A or grouping all uncalculated terms in an ℓ_k that will be estimated empirically [12], obtaining 2.26.

$$\|E_i\| \leq \ell_k \frac{1}{\|\tilde{r}_{i-1}\|}\epsilon \tag{2.26}$$

It has been noted in [12] that among the initial bounds, some aren't really sharp, mainly 2.16 and 2.23, and further empirical analysis of these bounds could show a better theoretical bound can be found for both.

In [12](section 5), it also has been shown that $\sigma_k(H_k)$ in Equation 2.25 can be approximated by $\sigma_n(A)$, the smallest singular value of A, through the bound:

$$\sigma_k(H_k) \leq \sigma_n(A) - \|E_1 q_1, E_2 q_2, \ldots, E_k q_k\| \tag{2.27}$$

Giving another way to evalue ℓ_k without the need of calculating $\sigma_k(H_k)$ at each iteration. A plot of these bounds for the cavity problem will be studied later.

It should also be noted that $\tilde{r}_k$, after the transformation of H_k into an upper triangular matrix, is also found in the $i+1$-th element of g_m in 2.19. The demonstration follow the same proof as for g_m in 2.8, given that y_m is a solution to a linear system that involves an upper triangular matrix.

The remaining theory in this report also explains the basics of Hierarchical Matrices, the structure that will be used to compress the matrices used in the algorithm, since A appears in the discretization of integral operators and uses large dimensions. It's also though these structures each $\mathcal{A}_k$ will be made. As it will be explained later, at each iteration an E_k will be indirectly constructed during the inexact product $\mathcal{A}_k q$, mainly using the residues that appear during this structure's construction, in the iterations of the *ACA Method*.

Chapter 3

Hierarchical Matrices and ACA Method

3.1 Low-rank Matrices

In practice, most matrices are large, so storing each element is not efficient, or even possible for some physical setups. If $A \in \mathbb{C}^{n\times m}$ has a rank k such that $k \leq m$ and $k(n+m) < n*m$ (A is low-rank), A can be written in outer product form, as a product between the matrices $U \in \mathbb{C}^{n\times k}$ and $V \in \mathbb{C}^{m\times k}$, which can be seen in 3.1, where u_i, v_i are the column vectors of U and V.

$$A = UV^H = \sum_{i=1}^{k} u_i v_i^* \tag{3.1}$$

Therefore, storing $k(n+m)$ elements to write A, and not $n \times m$. A matrix A that can be represented as 3.1 is an element of $\mathbb{C}_k^{n\times m}$.

The representation in 3.1 also facilitates other operations with A, like matrix-vector products Ab that are always present in methods like GMRES [2] and different kinds of norms, like $\|A\|_F, \|A\|_2$ [2].

However, even full rank matrices can be approximated by matrices with lower rank. A theorem [2] establishes that the closest matrix from $\mathbb{C}_k^{n\times m}$ of a matrix from $\mathbb{C}^{n\times m}$ can be obtained from the SVD $A = U\Sigma V^H$, where Σ contains the singular valuers $\sigma_1 \geq \sigma_2 \ldots \sigma_m \geq 0$ and U, V are unitary.

If A_k is the approximation obtained after taking the first k elements of Σ (creating the matrix Σ_k), the error between A and A_k is 3.2.

$$\|A - A_k\| = \left\| U\Sigma V^H - U'\Sigma_k V'^H \right\| = \|\Sigma - \Sigma_k\| \tag{3.2}$$

If the spectral norm, $\|.\|_2$ is used instead, the error in 3.2 is given by σ_{k+1}. For Frobenius's norm, $\|.\|_F$, the error becomes $\sqrt{\sum_{l=k+1}^{n} \sigma_l^2}$.

If a problem involves a large matrix that is not low-rank, it can have sub-blocks that can be approximated by matrices of this kind. Blocks that appear after the discretization of elliptic operators also have the possibility of being approximated by matrices that decay exponentially with k, S_k, as in 3.3.

$$\|A - S_k\|_2 < q^k \|A\|_2 \tag{3.3}$$

3.2 ACA Method(Adaptative Cross Approximation)

As shown in the last section, the SVD method gives us an approximation of A given a certain ϵ, through the relation in 3.2. Nevertheless, this is an expensive method. Not only having a large complexity, but also requiring knowledge of every element of the block going to be compressed.

The algorithm for the method is in 4, where a_{ij} are the elements of a matrix $A \in \mathbb{R}^{n \times m}$ and κ at the stopping criterion is the cluster parameter for a certain admissibility condition(discussed in the next section). The main objective is to approximate A as $A = S_k + R_k$, $S_k = \sum_{l=1}^{k} u_l v_l^t$ and R_k is the residual.

Algorithm 4 ACA Method

$k = 1$ et $\mathbf{Z} = \emptyset$
repeat
  Find i_k
  $\hat{v}_k = a_{i_k,1:m}$
  for $l = 1, \ldots, k-1$ **do**
    $\hat{v}_k = \hat{v}_k - (u_l)_{i_k} v_l$
  end for
  $Z = Z \bigcup \{i_k\}$
  if $\hat{v}_k$ does not vanish **then**
    $j_k = argmax_{j=1,\ldots,m} |(\hat{v}_k)_j|$
    $v_k = (\hat{v}_k)_{j_k}^{-1} \hat{v}_k$
    $u_k = a_{1:n,j_k}$
    for $l = 1, \ldots, k-1$ **do**
      $u_k = u_k - (v_l)_{j_k} u_l$
    end for
    $k = k + 1$
  end if
until $\|u_k\|_2 \|v_k\|_2 \leq \frac{\epsilon(1-\kappa)}{1+\epsilon} \|S_k\|_F$

And then, for the Frobenius Norm, it can be show [13](Section 3.4.1) that:

$$\frac{\|A - S_k\|_F}{\|A\|_F} = \epsilon \tag{3.4}$$

3.3 Hierarchical Matrices

Going a little more in depth about the use of low-rank approximations in larger matrices that are not really low-rank themselves, we present the concepts of *partition* and *admissibility*.

As mentioned in the first section of this chapter, even if $A \in \mathbb{C}^{m \times n}$ cannot be entirely approximated by a low-rank matrix, in the context we work with it can still be approximated in its sub-blocks. More specifically, in the sub-blocks of a proper partition P of the matrix indices.

Using $I = 1, 2, \ldots, m$ and $J = 1, 2, \ldots, n$, a subset P from the set of subsets of $I \times J$ is called a partition if:

$$I \times J = \bigcup_{b \in P} b \tag{3.5}$$

Where if $b_1 \cap b_2 \neq \varnothing$ then $b_1 = b_2$.

From here on out we also use the notation A_b or A_{ts} for the restriction of a matrix A to the indices in $b = t \times s$ where $b \in P$, $t \in I$ and $s \in J$.

The many algorithms that compress a given matrix have to make sure A_b will either be approximated by a low-rank block or is very small(making sure storing its elements will not be expensive). The number of blocks compressed in a matrix also should not be large.

The problem of knowing whether a block can be compressed comes from the abstract concept of *admissibility*, usually relying on the specific problem we solve. Nevertheless, *admissibility conditions* can be established that each one of the compressed blocks need to satisfy:

- If b is admissible, then its singular values decay exponentially;
- the admissibility condition for a block $t \times s$ can be verified with $\mathcal{O}(|t| + |s|)$;
- if b is admissible, then $b^{'} \in b$ also is.

A partition P is said admissible if every one of its blocks are either admissible or small, i.e., $|t|, |s|$, with $t \in I, s \in J$, satisfy $\min\{|t|, |s|\} \leq n_{min}$ for a given $n_{min} \in \mathbb{N}$.

Although the partition P has been mentioned multiple times throughout the text, we have not disclosed how such partition could be made.

It's obvious searching the entire set of partitions P of $I \times J$ is unfeasible, so the partitions are usually recursive subdivisions of both I and J. And it's shown such partitions can lead to linear complexity [2].

The result of the recursive division of a set results in a hierarchy of partitions that can be represented as a *cluster tree*.

A tree $T_I = (V, E)$ with vertices V, edges E, with $S(t) = t^{'} \in V : (t, t^{'}) \in E$ the set of sons of t and $\mathfrak{L}(T_I)$ the set of leaves of T_I is called a cluster tree of a set $I \subset \mathbb{N}$ if it follows the conditions:

1. I is the root of T_I;
2. $\varnothing \neq t = \bigcup_{t' \in S(t)} t^{'}$ for all $t \in V \setminus \mathfrak{L}(T_I)$;
3. the degree $\deg t := |S(t)| \geq 2$ of each vertex $t \in V \setminus \mathfrak{L}(T_I)$ is bounded from below.

To make a hierarchical structure for the *blocks* of a matrix, we extend the notion of this partition tree to $I \times J$, which contains the indices of said matrix. Thus, we have in each of its leaves, the set $\mathfrak{L}(T_{I\times J})$, one admissible block of the partition.

This cluster tree $T_{I\times J}$ for $I \times J$ is called a *block cluster tree*.

The set of hierarchical matrices in $\mathbf{T}_{I\times J}$ with rank k for each block A_b is defined in 3.6.

$$\mathfrak{H}(\mathbf{T}_{I\times J}, k) = \left\{A \in \mathbb{C}^{I\times J} : rank A_b \leq k, \forall b \in P\right\} \tag{3.6}$$

In practice, 4 works with ϵ given by the user during the assembling of the Hierarchical Matrix, and the compression acts in each of the admissible blocks that will then be represented as low rank matrices, shown in 3.1. We also store each one of the residuals obtained in the outer loop of 4.

After giving a tolerance $\nu \leq \epsilon$ for an inexact product, the algorithm, for each admissible block of the cluster tree, uses the right amount of columns $k^{'}$ of the outer product representation of the block as to reach the desired tolerance. The number of columns can be inferred by using the list of residuals of the ACA Method. A brief explanation can be found in 5.

Where, if no value smaller than ν is found, we use all k columns of each block to execute the product.

So, the different perturbations E_k used in 2.13 are the matrices that, when added to A, leave each admissible block with the right amount $k^{'}$ of columns in 3.1, so the product can be

Algorithm 5 Inexact Product algorithm.

1: Be $\nu \in \mathbb{R}$, $b \in P$ the tolerance for the product, $A_b = \sum_{i=1}^{k} u_i v_i^*$ a restriction of A to one of its admissible blocks and $r \in \mathbb{R}^{\urcorner}$ a vector with the residuals obtained in each step of the ACA.
2: **for** $k^{'} \in \{1, \ldots, k\}$ **do**
3: $\quad \alpha \leftarrow r_{k'}$
4: $\quad$ **if** $\alpha \leq \nu$ **then**
5: $\quad\quad \tilde{A}_b = \sum_{i=1}^{k^{'}} u_i v_i^*$
6: $\quad$ **end if**
7: **end for**

approximated by the tolerance ν. Calling the approximation of each block A_b as $\tilde{A}_b$, we have 3.7.

$$\frac{\left\| A_b q - \tilde{A}_b q \right\|}{\|A_b q\|} \leq \nu, \qquad \tilde{A}_b = \sum_{i=1}^{k^{'}} u_i v_i^* \tag{3.7}$$

And then, since the Frobenius norm of the hierarchical matrix is [10](Section 3.5.1):

$$\|E_k\|_F = \|\mathcal{A}_k - A\| = \sqrt{\sum_{b \in P} \left\| A_b - \tilde{A}_b \right\|_F^2} \tag{3.8}$$

But from the ACA Method 3.4, we know the method stops as to make each block's successive approximation inferior to previous one by a factor of ϵ, then:

$$\|E_k\|_F = \sqrt{\sum_{b \in P} \left\| A_b - \tilde{A}_b \right\|_F^2} \leq \sqrt{\sum_{b \in P} \epsilon^2 \|A_b\|_F^2} = \epsilon \sqrt{\sum_{b \in P} \|A_b\|_F^2} \leq \epsilon \|A\|_F \tag{3.9}$$

With this result, we can then substitute $\|E_k\|$ appearing in the upper bounds of chapter 2, using the fact that $E_k \propto \epsilon_k \|A\|$, where ϵ_k is the tolerance used in the k-th inexact matrix-vector product.

Chapter 4

Boundary Element Methods

4.1 Boundary Element Methods

Until now, we've disclosed most of the tools used in the project. From the basic problem of solving a linear system in 1.1, the algorithm in use and some of its details in chapter 2 and also the data structure permitting to make the inexact approximations in chapter 3, a great deal of ground was covered, but we haven't talked on **how** obtaining A and b in 1.1.

Therefore, we dedicate a small chapter to elaborate how the Boundary Element Methods work, more specifically, how to generate the components of our linear system.

4.2 Mathematical model and Integral Equations

Boundary Element Methods(BEM) are numerical approximations of Boundary Integral equations, which themselves are tools for analyzing of boundary value problems for partial differential equations(PDEs). BEM do present some advantages, like the fact that only the boundary gets discretized and how boundary values of the solution can be directly obtained, but also some shortcomings. The necessity of knowing a fundamental solution to the differential equation and the problems arising from non-smooth boundaries may require that application of alternative methods, like FEM, or maybe a mixture of both [5].

Starting out with the PDE, we use Laplace's equation as an illustration. For the boundary problem, we'll start out with Dirichlet conditions:

$$\begin{aligned} \nabla u &= 0 \quad \text{in a domain } \Omega \in \mathbb{R}^n \\ u &= g \quad \text{on the boundary } \Gamma := \partial\Omega \end{aligned} \tag{4.1}$$

As said before, we need to know the *fundamental solution* to the differential equation. Considering $\mathbb{R}^2$ the fundamental solution of the Laplace equation is:

$$\gamma(x,y) = \frac{-1}{2\pi} \log|x-y| \quad (x,y \in \mathbb{R}^2) \tag{4.2}$$

We then represent the solution of the differential equation by using a representation formula. Since Laplace's equation's is known, we write Green's third identity for the solution:

$$u(x) = \int_\Gamma \partial_{n_y}\gamma(x,y)[u(y)]_\Gamma \, d\Gamma y - \int_\Gamma \gamma(x,y)[\partial_{n_y}u(y)]_\Gamma \, d\Gamma y \quad , x \in \mathbb{R}^2 \backslash \Gamma \tag{4.3}$$

Where ∂_{n_y} denotes the derivative taken with respect to the exterior normal of Γ(pointing outwards), and u is harmonic, regular in the interior and exterior domains.

The term $[]_\Gamma$ represents the jump value of a function across Γ:

$$[v(x)]_\Gamma = v_{|R^2\setminus\Omega}(x) - v_{|\overline{\Omega}}(x) \tag{4.4}$$

This step also brings one of the many choices we have to make to get different forms of the BEM. All of these depend on the choice on the assumptions we make about u in $\mathbb{R}^2\setminus\Omega$.

We use here the *direct method*, choosing $u_{|\mathbb{R}^2\setminus\Omega} = 0$, obtaining the new expression:

$$u(x) = -\int_\Gamma \partial_{n_y}\gamma(x,y)u(y)\,d\Gamma y + \int_\Gamma \gamma(x,y)\partial_{n_y}u(y)\,d\Gamma y \quad ,x \in \Omega \tag{4.5}$$

Where each one of the terms on the right side of this equation are called the double and single layer, respectively. Their densities, the function appearing in the integrand other than the fundamental solution $\gamma(x,y)$, are the solution and its normal derivative, both on the boundary.

But 4.5 is written for values in the domain Ω, we need a relation for values in Γ. The single layer potential $\int_\Gamma \gamma(x,y)v(y)\,d\Gamma y$ is continuous across Γ, then its extension becomes [1](Chapter 3):

$$S[v](x) = \int_\Gamma \gamma(x,y)v(y)\,d\Gamma y \quad ,x \in \Gamma \tag{4.6}$$

If in 4.3 we had chosen $[u]_F$ instead, the single layer would be the only term remaining, and we would have $S[u](x) = g$ in the boundary, a Fredholm integral equation of the first kind.

The double layer potential, $\int_\Gamma \partial_{n_y}\gamma(x,y)v(y)\,d\Gamma y$, has the following extension [1](Chapter 3) to the boundary:

$$\int_\Gamma \partial_{n_y}\gamma(x,y)v(y)\,d\Gamma y = \frac{-v(x)}{2} + \int_\Gamma \partial_{n_y}\gamma(x,y)v(y)\,d\Gamma y = (\frac{-1}{2} + D)v(x) \quad ,x \in \Gamma \tag{4.7}$$

Where:

$$D[v](x) = \int_\Gamma \partial_{n_y}\gamma(x,y)v(y)\,d\Gamma \quad ,x \in \Gamma \tag{4.8}$$

If in 4.3 we had chosen $[\partial_n u]_\Gamma = 0$, the double layer expression would result in $(\frac{-1}{2} + D)u = g$ in the boundary, a Fredholm integral equation of the second kind.

Using 4.6, 4.7 and 4.8 in 4.5, we have:

$$\begin{aligned} u(x) &= \frac{u(x)}{2} - D[u](x) + S[\partial_n u](x) \\ (\frac{1}{2} + D)u &= S(\partial_n u) \end{aligned} \tag{4.9}$$

Now, the last line in 4.9, coupled with the boundary conditions in 4.1 can be used to find $\partial_n u(x)$ and then the final answer.

If instead of a Dirichlet problem, we had a Neumann's one, where $\partial_n u(x)$'s values are specified at the boundary, we would then find $u(x)$ directly through 4.9.

4.3 Discretization and Linear System

The expressions found in the section above, even though they contain the answers we need, are still continuous. To obtain the linear systems we'll be working on, a discretization process is necessary.

We start, as mentioned before, by assuming Γ can be decomposed into a finite number of subsets, each one represented by a parameter in $\mathbb{R}^{n-1}$(noting that we used n=2 in the last section as a mere illustration). Then we choose a partition of this parameter domain and corresponding finite element functions.

We start with a boundary integral equation as in 4.9:

$$Au = f \quad , x \in \Gamma \tag{4.10}$$

And we search for an approximate solution:

$$u_h(x) = \sum_{j=1}^{N} \alpha_j \mu_h^j(x) \tag{4.11}$$

With the basis functions $\mu_h^j = \mu_h^j | j = 1, \ldots, N$ and $\alpha_j | j = 1, \ldots, N$ the unknown coefficients.
The linear system can be obtained by three methods: Collocation Method.

4.3.1 Collocation method

Collocation method starts out by choosing a set of points $x_j | j = 1, \ldots, N \subset \Gamma$ of collocation points and assumes 4.10 is satisfied in those.

Using 4.11 with the collocation points x_k and 4.10:

$$\sum_{j=1}^{N} (A\mu_h^j)(x_k)\alpha_j = f(x_k) \quad , k = 1, 2, \ldots, N \tag{4.12}$$

Where, in this chapter, A is the integral operator in Equation 4.10.

Chapter 5

Results

By this point, the report brought up all the tools we need for an application of the InexactGMRES algorithm, from the algorithm itself to the way we generate the coefficient matrices used in the linear equations.

Before we use the main algorithm directly in a more complex problem, we start out with simpler geometries, to first validate the speedup obtained in a simple matrix-vector product with different tolerances. We then apply the Inexact GMRES in the problem of a cavity scattering, known for taking a larger number of iterations in the standard GMRES.

5.1 Validation of the inexact matrix-vector product

For the first test, we use Laplace's equation (first line in 4.1, and that we repeat here for context) to generate our first coefficient matrix. The second matrix-vector product example will use Helmholtz's equation, the second line in the following equation:

$$\begin{aligned} \Delta u &= 0 \\ \Delta u + k^2 u &= 0 \end{aligned} \tag{5.1}$$

With k the wave number associated with the solution.

As for the geometry, we choose a unit circle around the origin as the boundary, using the Inti library [8] to go through all steps in chapter 4.

The first coefficient matrix A_L is made from the fundamental solution of Laplace's equation, the first line in Equation 5.1, after transforming it into an integral equation, through the single layer potential mentioned in Equation 4.6.

With $\gamma(x, y)$ the fundamental solution of Laplace's equation:

$$\gamma(x, y) = \frac{-1}{2\pi} \log |x - y| \quad (x, y \in \mathbb{R}^2) \tag{5.2}$$

The coefficient matrix A_H for the second example is generated from Helmholtz's equation, second line in Equation 5.1, using the single and double layer potentials in Equation 4.6 and Equation 4.8. The fundamental solution of the differential equation is:

$$\gamma(x, y) = \frac{i}{4} H_0^{(1)}(k|x - y|) \tag{5.3}$$

The Hankel function of the first kind.

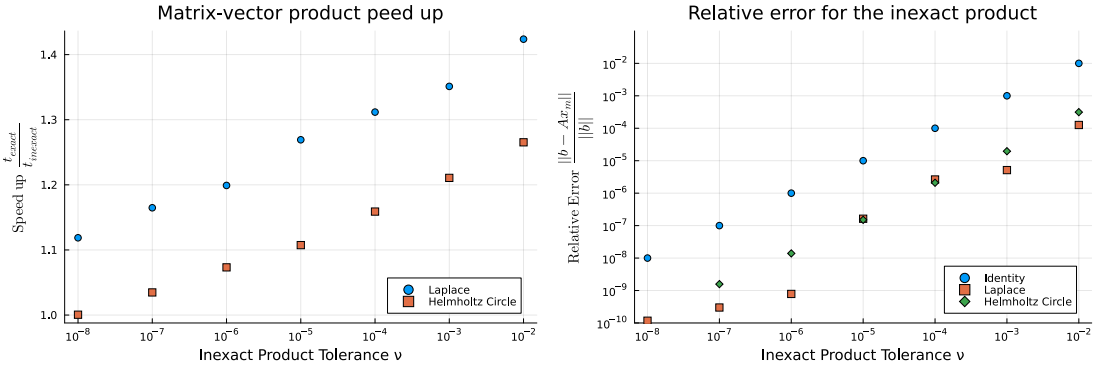
Before going into each one of the results, we talk a little about both extremes of the performance evaluation in the matrix-vector problem. How to give an upper bound to the acceleration we expect?

If making the matrix-vector product's tolerance as 0 implies the use of the "original" matrix(hierarchical matrices are already an approximation). Then, we would expect 0% speedup and t_{exact}, the execution time for the exact product, to be equal to $t_{inexact}$, the execution time for the inexact version.

A good way to infer an upper bound for the inexact products would be using only rank 1 blocks for the admissible ones and measuring its execution time. To do that, we change the product tolerance to a very large value, *inf* from Julia, and the product will be realized with only rank-1 blocks, since it's programmed to get the first approximation lower than its given tolerance.

Of course, such thing would not happen in a practical situation, but it gives an upper bound for the speedup we should expect, the maximum value $\frac{t_{exact}}{t_{inexact}}$ could attain.

To assemble both hierarchical matrices, A_L and A_H, we use an overall tolerance ϵ of 10^{-8}. The results for the speed-ups and the relative residual for each product tolerance, ν, are in Figure 5.1.



(a) Speed-ups for the inexact product of a hierarchical matrix and a vector.

(b) Relative residuals for inexact product of a hierarchical matrix and a vector.

Figure 5.1: Results for the initial example of a inexact product. The coefficient matrices have dimensions 62835 × 62835 and were assembled with an overall tolerance $\epsilon = 10^{-8}$.

We leave the identity curve in the plot on the right due to compare each inexact product's residual with the minimum accuracy that should be satisfied by its resulting vector. And since the matrix-product is accelerated, we should expect some speedup in the algorithm as well.

It should be noted that the acceleration is highly dependent on the compression of the coefficient matrix. Matrices with low maximum rank of each block would not receive a big boost, since the number of columns used for each sub-block might not vary much with each different tolerance.

5.2 Cavity scattering

5.2.1 Tolerance heuristic

We mentioned the inexact product's tolerance, ν, multiple times throughout this study, but haven't really disclosed **how** to obtain it. For the simple example of the matrix-vector product of before, we could simply define it as we please, but the situation changes for the Inexact GMRES. For the latter, everything needs to be made as to give a converging solution, specially to respect all bound found in the inexact session of chapter 2.

As told in chapter 3, we adjust the inexact matrix-vector product with the number of columns used by each one of the subblocks during the operation.

We use a heuristic to obtain, at each iteration k, a product tolerance ν_k as to keep $||E_k||$ lower than the bounds in Equation 2.25 or Equation 2.26. Then ν_k is later converted to an integer, the number of columns used by each admissible block, through algorithm 5.

The following formula, used in [14], is equivalent to setting ℓ_k in Equation 2.26 as 1.

$$\nu_k = \min\left(\frac{\epsilon}{\min(\tilde{r}_{k-1}, 1)}, 1\right) \tag{5.4}$$

Where $\tilde{r}_{k-1}$ is the calculated residual of the previous $k-1-th$ iteration and ϵ the overall tolerance given to the inexact GMRES algorithm.

Any of the other bounds showed in chapter 2 can be made by multiplying this expression by $\frac{\sigma_k(H_k)}{k}$, $\frac{\sigma_n(A)}{k}$ or other values of ℓ_k. A comparison between these different heuristics in the problem of a cavity scattering can be found below 5.3.

5.2.2 Problem and results

Since the solution we search for this larger problem is the scattered field, u, produced by the incident wave $u_i = \exp^{ik(\vec{d}.\vec{x})}$, we still need to find the correct boundary condition in Γ to obtain a unique solution.

Assuming the total field, $u_t = u_i + u$ satisfies a homogenous Neumann condition and that u satisfies the Sommerfeld radiation condition, we have the following equations to the problem:

$$\begin{aligned} \Delta u + k^2 u = 0 \qquad & , x \in \mathbb{R}^2 \backslash \overline{\Omega} \\ \partial_n u = -\partial_n u_i \qquad & , x \in \Gamma \end{aligned} \tag{5.5}$$

Which gives us, with 4.9, all the mathematical formulation we need to continue with the application. The final linear system is:

$$\left(\frac{I}{2} + D - ikS\right) x = Ax = g \tag{5.6}$$

With g the application of $\partial_n u_i$ in each collocation point made for the discretization of the cavity.

The mesh uses a *.geo* file available in [9] and a figure of the whole geometry can be seen in 5.2. We choose the incident wave's angle as $\frac{\pi}{4}$ rad.

We begin by running smaller examples and study the bounds found of chapter 2, mainly 2.25 and 2.26. Satisfying these bounds is a guarantee that the residual measured at each k-th iteration, $\tilde{r_k}$, is close to the exact one $r_k = b - Ax_k$.

This test run was executed with a smaller A, of dimensions 1650×1650, and we use $\epsilon = 10^{-8}$ to assemble the hierarchical matrix as well as the algorithm's overall tolerance, i.e., the iterations stop as soon as $\frac{\|b-Ax_k\|}{\|b\|} \leq \epsilon$. The plots are in Figure 5.3.

The bigger picture in Figure 5.3 shows how the different bounds of the perturbation's behave through iterations. Recalling that a greater tolerance means a bigger room to work with larger perturbations of A, what can make the inexact matrix-vector product faster.

Even though choosing $\ell_k = 1$ makes the inexact product's tolerance bigger, i.e., "allows" for a more perturbed A, it stills follows the restriction in the bound 2.17, as we can see on the left smaller plot. This shows bigger tolerances can still be used to achieve higher acceleration, while still maintaining the residual in a proper value.

The different product tolerances at each iteration did not result in big variations in the overall amount of iterations for the Inexact GMRES.

Passing on to the speed-up evaluation, we repeat a similar numeral setup: A is assembled using the overall tolerance $\epsilon = 10^{-8}$. All product tolerances for each iteration, ν_k, are obtained through the heuristic in 5.4.

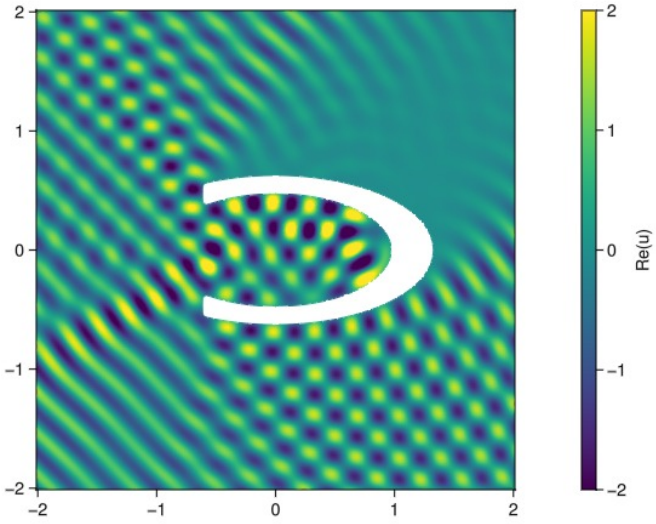
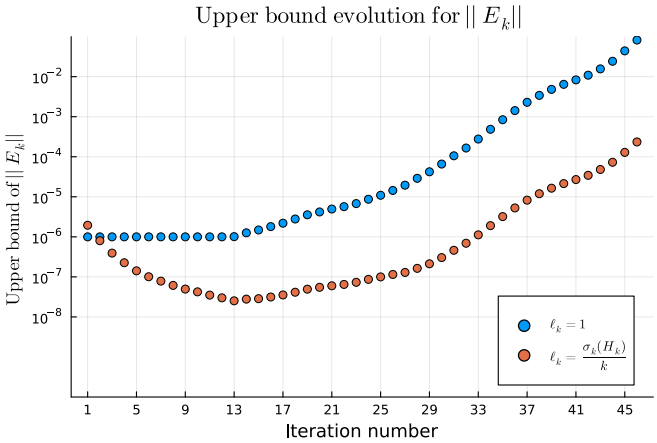
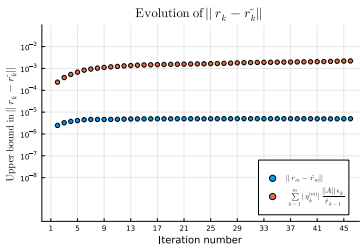


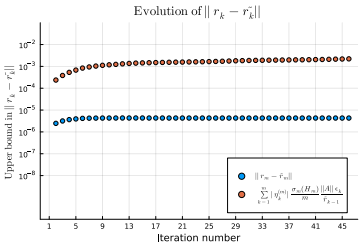
Figure 5.2: Geometry used in the test.



(a) Evolution for the upper bound of $||E_k||$, using $\ell_k = 1$ and $\ell_k = \frac{\sigma_k(H_k)}{k}$.



(b) Evolution of the upper bound in $\|r_k - \tilde{r_k}\|$ for the case $\ell_k = 1$.



(c) Evolution of the upper bound in $\|r_k - \tilde{r_k}\|$ for the case $\ell_k = \frac{\sigma_k(H_k)}{k}$.

Figure 5.3: Multiple Images

For the first batch of tests, we fix A's dimensions and change the overall tolerance passed to the algorithm. The speed-up results for this fixed size test are in Figure 5.4.

An evolution of the number of iterations in face of the different tolerances passed to the algorithm is in 5.5.

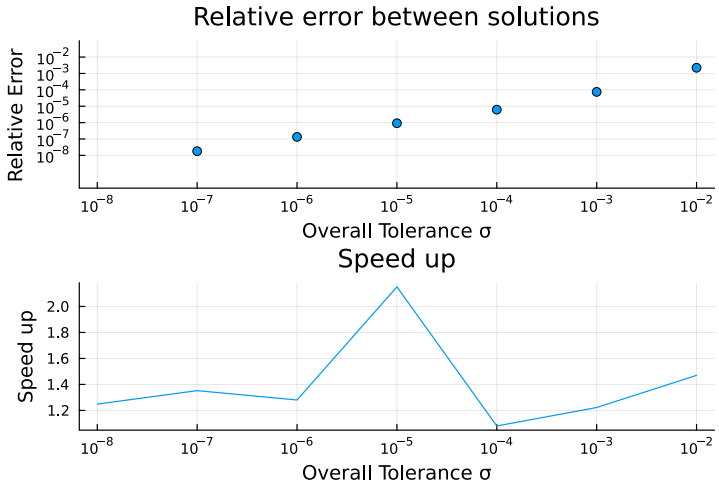


Figure 5.4: Speedup witnessed in the application of the Inexact GMRES in a 50000×50000 matrix.

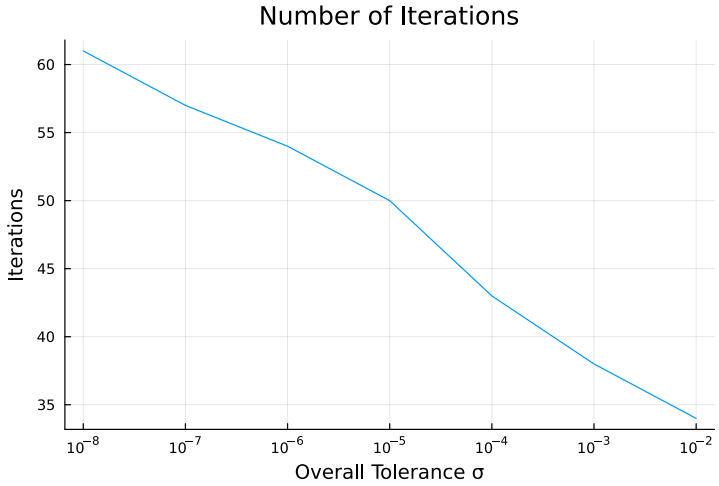


Figure 5.5: Evolution of the quantity of iterations needed for convergence and overall tolerance passed as an argument.

Chapter 6

Bibliography

[1] Stefan A. Sauter and Christoph Schwab. *Boundary Element Methods.* Springer, 2011.

[2] Mario Bebendorf. *Hierarchical matrices.* Springer, 2008.

[3] Marc Bonnet. *Lecture notes on numerical linear algebra.* ENSTA Paris - POEMS Group, 2021.

[4] Amina Bouras and Valérie Frayssé. Inexact matrix-vector products in krylov methods for solving linear systems: a relaxation strategy. *SIAM Journal on Matrix Analysis and Applications*, 26(3):660–678, 2005.

[5] Martin Costabel. Principles of boundary element methods. *Computer Physics Reports*, 6(1-6):243–274, 1987.

[6] Eric Darve and Mary Wootters. *Numerical linear algebra with Julia.* SIAM, 2021.

[7] Luiz Farias. Hmatrices.jl. `https://github.com/IntegralEquations/HMatrices.jl`.

[8] Luiz Farias. Inti.jl. `https://github.com/IntegralEquations/Inti.jl`.

[9] Eduardo Guimaraes. Github's repository. `https://github.com/DuduGuima/InexactGMRES.git`.

[10] Wolfgang Hackbusch et al. *Hierarchical matrices: algorithms and analysis*, volume 49. Springer, 2015.

[11] Yousef Saad. *Iterative methods for sparse linear systems.* SIAM, 2003.

[12] Valeria Simoncini and Daniel B Szyld. Theory of inexact krylov subspace methods and applications to scientific computing. *SIAM Journal on Scientific Computing*, 25(2):454–477, 2003.

[13] Lloyd N Trefethen, David Bau III, and Ricardo D Fierro. Numerical linear algebra. *SIAM Review*, 40(3):735–738, 1998.

[14] Tingyu Wang, Simon K. Layton, and Lorena A. Barba. Inexact krylov iterations and relaxation strategies with fast-multipole boundary element method. `https://arxiv.org/abs/1506.05957`, 2016.