

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, and hierarchical matrix approximation to a 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 will be carried out in the context of two existing libraries: HMatrices.jl and Inti.jl. The internship will take place at the POEMS laboratory, and will be 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 begin with the usual problem of solving a linear system, with $A \in \mathbb{C}^{m \times n}$ an invertible matrix, $b \in \mathbb{C}^m$ 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 [12], where m is the dimension of the input matrix. Since larger matrices are usually employed in practice, the direct algorithms become inefficient, and we require a more reliable approach.

The idea of the iterable methods is to 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, guaranteeing that the convergence rate of the method is sufficiently fast gives a $k << m$, and the Interative Method can be way more efficient than its counterpart.

The method we employ in our problems is the Generalized Minimal Residuals(GMRES), explained later in the report. Its main ideia 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

Be $A \in \mathbb{C}^{n\times n}$ a matrix and $b \in \mathbb{C}^n$. To 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 polynom 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 colinear*, 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, an iterative method itself, 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 othornormal basis for $\mathcal{K}_k(A, b)$.

Proof. By construction q_j, $j = 1, 2, \ldots, k$ are othornormal. 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 $q_1 = \frac{b}{||b||}$.

If we denote by Q the nxk matrix with column vectors $q_1, \ldots, q_k$ found in 1 and H_k the $(k+1)xk$ Hessenberg matrix whose nonzero entries h_{ij} are given just as in 1, we have 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* [12].

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*, tipically 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 answer x to an element in the lower dimentional 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 answer of the minisation problem of r_k. Since we make the projections in $\mathcal{K}_k(A, b)$, $x_k = K_k y_k$, our answer 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:

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

But as said in the previus chapters,$b, Ab, \ldots, A^k b$ has approximately colinear 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 minimisation problem in 2.5:

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

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.

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

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 [10].

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

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

And since Ω_m^t is a rotation matrix, its unitary and $\left\|\Omega_m^t(g_m - R_m y_m)\right\| = \|g_m - R_m y_m\|$. For any vector y, as the last line in 2.10 is fileld 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 minimisation 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 heaviest 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 answer, as shown in 2.12.

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

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

When we realise 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 foward.

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 spams, 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 $q_1 = \text{b}$). 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, $\mathcal{K}_k(A+\mathcal{E}_k,b) = \text{span}\{b,\mathcal{A}_1 b,\ldots,\mathcal{A}_k\ldots\mathcal{A}_1 b\}$, made by a large pertubation 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: r_k, the exact residual of an iteration, and $\tilde{r}_k$, the one that will really be calculated. A detailed definition for both and a measure of how distant they are is in 2.15.

$$\begin{aligned} r_k &= r_0 - AQ_k y_k \\ &= r_0 - (Q_{k+1}H_k - [E_1q_1,\ldots,E_kq_k])y_k \\ &= \tilde{r}_k + [E_1q_1,\ldots,E_kq_k]y_k \\ \rightarrow \delta_k &= \|r_k - \tilde{r}_k\| = \|[E_1q_1,\ldots,E_kq_k]y_k\| \end{aligned} \tag{2.15}$$

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

$$\begin{aligned} \|[E_1q_1,\ldots,E_kq_k]y_k\| &= \left\|\sum_{i=1}^{k} E_i q_i \eta_i^{(k)}\right\| \\ \left\|\sum_{i=1}^{k} E_i q_i \eta_i^{(k)}\right\| &\leq \sum_{i=1}^{k} \|E_i\| \left\|q_i \eta_i^{(k)}\right\| \\ \left\|\sum_{i=1}^{k} E_i q_i \eta_i^{(k)}\right\| &\leq \sum_{i=1}^{k} \|E_i\| \, |\eta_i^{(k)}| \end{aligned} \tag{2.16}$$

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 pertubation 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.

Knowing y_k is the solution of the minimisation of $\|H_k y_k - e_1\beta\|$, we consider $\Omega_k = G(k, k+1)G(k-1, k)\dots G(1, 2)$ where each G represents a Givens rotation as shown in 2.7, so Ω_k is the matrix that transforms H_k into an upper triangular matrix.

The aplication of Ω_k in either side of $H_k y_k = e_1\beta$ gives us 2.18.

$$\begin{aligned} \Omega_k H_k y_k &= \Omega_k e_1 \beta \\ R_k y_k &= g_k \\ y_k &= R_k^{-1} g_k \end{aligned} \tag{2.18}$$

Since R_k, the transformation of a Hessenberg matrix by a series of Givens rotations, is upper triangular, then its inverse also is. Being an upper triangular matrix, the first $i-1$ elements of its ith line are zeros, so using Matlab index notation in 2.19.

$$(R_k^{-1})_{i,1:k}(g_k)_{1:k} = (R_k^{-1})_{i,i:k}(g_k)_{i:k} \tag{2.19}$$

Using this last result in 2.18 gives 2.20.

$$\begin{aligned} |\eta_i^{(k)}| &= \left\|(R_k^{-1})_{i,i:k}(g_k)_{i:k}\right\| \\ |\eta_i^{(k)}| &\leq \left\|e_k R_k^{-1}\right\| \left\|(g_k)_{i:k}\right\| \\ |\eta_i^{(k)}| &\leq \left\|e_k R_k^{-1}\right\| \left\|(g_k)_{i:k}\right\| \end{aligned} \tag{2.20}$$

Since $\left\|e_k R_k^{-1}\right\| \leq \left\|R_k^{-1}\right\| = \sigma_k(H_k)^{-1}$ and $\left\|(g_k)_{i:k}\right\| \leq \|\tilde{r}_{i-1}\|$ [11], the bound is given by 2.21.

$$\left\|\eta_i^{(k)}\right\| \leq \frac{1}{\sigma_k(H_k)} \|\tilde{r}_{i-1}\| \tag{2.21}$$

Putting 2.21 in 2.17 gives the results in 2.22. Setting $\delta_k \leq \epsilon$ and determining a bound for each $\|E_i\|$ gets us 2.23.

$$\delta_k \leq \sum_{i=1}^{k} \frac{\|E_i\|}{\sigma_k(H_k)} \|\tilde{r}_{i-1}\| \tag{2.22}$$

$$\|E_i\| \leq \frac{\sigma_k(H_k)\epsilon}{k \|\tilde{r}_{i-1}\|} \tag{2.23}$$

Since H_k is also one of the matrices being constructed throughout the method, a workaround is necessary to apply find these bounds in a pratical 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 [11], obtaining 2.24.

$$\|E_i\| \leq \ell_k \frac{1}{\|\tilde{r}_{i-1}\|} \epsilon \tag{2.24}$$

It should be noted [11] that among te initial bounds, some aren't really sharp, mainly 2.16 and 2.21, and further empirical analysis of these bounds could show a better theoretical bound can be found for both. A plot of these bounds for the cavity problem that will be studied is shown 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 2.18. 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 strucures 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 $ACAMethod$.

Chapter 3

Hierarchical Matrices and ACA Method

3.1 Low-rank Matrices

In reality, 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.

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

That way the rank, and precision are related logarithmically, and the rank required by a certain ϵ is 3.4.

$$k(\epsilon) = \min\{k \in \mathbb{N} : \sigma_{k+1} < \epsilon\sigma_1\} \tag{3.4}$$

3.2 ACA Method(Adaptative Cross Approximation)

As shown in the last section, the SVD methods 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 $A \in \mathbb{R}^{n\times m}$. 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
  TFind 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 $\hat{v}_k$ doesn't disappear **then**
    $j_k = argmax_j |(\hat{v}_k)_j|$; $v_k = (\hat{v}_k)_{j_k}^{-1} \hat{v}_k$
    $u_k = a_{1:n,j_k}$
    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 $\|u_k\| \|v_k\| \leq \epsilon$

And then, for the Frobenius Norm, it can be show [12](Section 3.4.1) that:

$$\frac{\|A - S_k\|_F}{\|A\|_F} = \epsilon \tag{3.5}$$

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 mentionned in the first section of this chapter, $A \in \mathbb{C}^{mxn}$ can be approximated by low-rank matrices only 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:

$$I \times J = \bigcup_{b \in P} b \tag{3.6}$$

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 wether 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 stablished that each one of the compressed blocks need to satisfy:

- If b is admissible, then its singular values decay exponentially, just as in 3.4;
- the admissibility condition for a block $t \times s$ can be veryfied 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 admissble 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 and edges E 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.

$S(t) = t^{'} \in V : (t, t^{'}) \in E$ is the set of sons of t and $\mathfrak{L}(T_I)$ denotes the leaves of T_I.

The cluster tree $T_{I \times J}$ for $I \times J$ is called a block cluster tree. This structure contains an admissible partition P of $I \times J$ in its leaves, $\mathfrak{L}(\mathbf{T}_{I \times J})$.

The set of hierarchical matrices in $\mathbf{T}_{I \times J}$ with rank k for each block A_b is defined in 3.7.

$$\mathfrak{H}(\mathbf{T}_{I \times J}, k) = \left\{ A \in \mathbb{C}^{I \times J} : rank A_b \leq k, \forall b \in P \right\} \tag{3.7}$$

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 σ for an inexact product, the algorithm, for each admissible block of the cluster tree, uses the right amount of columns of the outer product representation of the block as to reach the desired tolerance. The number of columns can be infered by using the list of residuals of the ACA Method.

So, the different pertubations 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 approximated by the tolerance σ. Calling the approximation of each block B_k as $\tilde{B}_k$, we have 3.8.

$$\frac{\left\| B_k q - \tilde{B}_k q \right\|}{\|B_k q\|} \leq \sigma, \qquad \tilde{B}_k = \sum_{i=1}^{k^{'}} u_i v_i^* \tag{3.8}$$

And then, since the Fobrenius norm of the hierarchical matrix is [9](Section 3.5.1):

$$\|E_k\|_F = \|\mathcal{A}_k - A\| = \sqrt{\sum_{b\in P} \left\|B_b - \tilde{B}_b\right\|_F^2} \tag{3.9}$$

But from the ACA Method 3.5, we know the method stops as to make each blocks' approximation inferior to ϵB_b, then:

$$\|E_k\|_F = \sqrt{\sum_{b\in P} \left\|B_b - \tilde{B}_b\right\|_F^2} \leq \sqrt{\sum_{b\in P} \epsilon^2 \|B_b\|_F^2} = \epsilon \sqrt{\sum_{b\in P} \|B_b\|_F^2} = \epsilon \|A\|_F \tag{3.10}$$

So, with the use of the relative errors and the Frobenius norm, $\|E_i\|$ appearing in the bounds of section 2.2 can be exchanged by ϵ_i, the tolerance used for that specific approximation in the ienxact products.

Chapter 4

Boundary Element Methods

4.1 Boundary Element Methods

Until now, we've disclosed most of the tools used in the projet. 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 struture permiting 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 systeam.

4.2 Mathematical model and Integral Equations

Boundary Element Methods(BEM) are numerical approximations of Boundary Integral equations, which themselves are tools for analysing 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 [4].

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 represente 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 boundarey, 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 extention [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 illustation). Then we choose a partition of this parameter domain and correponding 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, Galerkin's Method and least squares 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:

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

4.3.2 Galerkin's Method

In this method, we use 4.10 with 4.11 and take the product of the equation with a set of test functions of a finite dimensional function space.

If we use, for the test functions, the same set of basis functions in 4.11, we have:

$$\sum_{j=1}^{N} \left\langle \mu_h^k, A\mu_h^j \right\rangle \alpha_j = \left\langle \mu_h^k, f \right\rangle \quad , k = 1, \ldots, N \tag{4.13}$$

Where $< ., . >$ denotes the inner product in $\mathbf{L}^2(\Gamma)$:

$$\langle f, g \rangle := \int_\Gamma \overline{f(x)} g(x) \, d\Gamma x \tag{4.14}$$

4.3.3 Least squares method

In this method we minimize $A\mu_h - f$ in the $\mathbf{L}^2(\Gamma)$ norm, resulting in the equation:

$$\sum_{j=1}^{N} \left\langle A\mu_h^k, A\mu_h^j \right\rangle \alpha_j = \left\langle A\mu_h^k, f \right\rangle \quad , k = 1, \ldots, N \tag{4.15}$$

With all integrals solved numerically.

We then return to a linear system, as in 1.1.

Chapter 5

First 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, and first validating the speedup obtained in a simple matrix-vector product with different tolerances.

For the first test, we use Laplace's equation (first line in 4.1) to generate our coefficient matrix. The other examples will use Helmholtz's equation:

$$\Delta u + k^2 u = 0 \tag{5.1}$$

For the first two matrix-vector product examples, we choose a unit circle around the origin as the boundary, using the Inti library [7] to create the mesh.

For the complex problem, we choose a cavity scattering problem. The mesh uses a *.geo* file avaiable in [8] and a figure of the whole geometry can be seen in 5.1. We choose the incident wave's angle as $\frac{\pi}{4}$ rad.

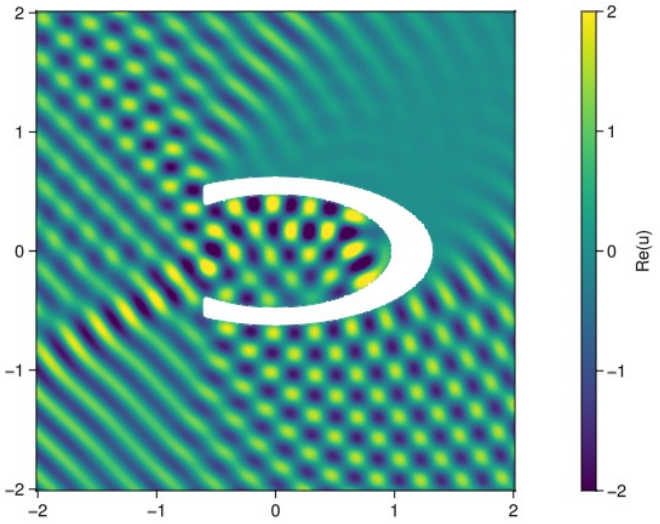


Figure 5.1: Geometry used in the test.

Since the solution we search 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:

$$\begin{aligned} \Delta u + k^2 u &= 0 \\ \partial_n u &= -\partial_n u_i \quad , x \in \Gamma \end{aligned} \tag{5.2}$$

Which gives us, with 4.9, all the mathematical formulation we need to continue with the applications.

Before going into each one of the results, we talk a little about both extremes of the performance evaluation in the inexact GMRES and matrix-vector problem.

If making the matrix-vector product's tolerance as 0 implies the use of the "original" matrix(remember we are always using hierarchical matrices, which are already a compression and therefore, not the orignal matrix), and then, an expected 0% speedup, how to give an upper bound to the acceleration we expect?

A good way to infer the maximum acceleration possible for the inexact products would be using only admissible rank 1 blocks and measuring its execution time.

For doing that, we change the product tolerance to *Infinity*, and the product will be realised with only rank-1 blocks, since it's programmed to get the first aproximation 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.

5.1 Tolerance heuristic

We mentioned the inexact product's tolerance multiple times throughout this study, but haven't really disclosed **how** to obtain it. As told in chapter 3, the inexact matrix-vector product is adjusted with the number of columns used by each one of the subblocks during the operation. To convert one of the expressions in Equation 2.23 or in Equation 2.24 to an integer, we use the following heuristic:

$$n_k = \min\left(\frac{\epsilon}{\min(\tilde{r}_{k-1}, 1)}, 1\right) \tag{5.3}$$

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

This expression is equivalent to take $\ell_k = 1$ in Equation 2.24. 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 5.4.

5.2 Laplace's results

Setting our product tolerance to *Infinity* and using only rank-1 blocks in the product, we got a () speedup.

All results are contained in 5.2, showing the evolution of the residual with the product tolerance aswell as the speedup to each of these values.

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 subblock might not vary much with each different tolerance.

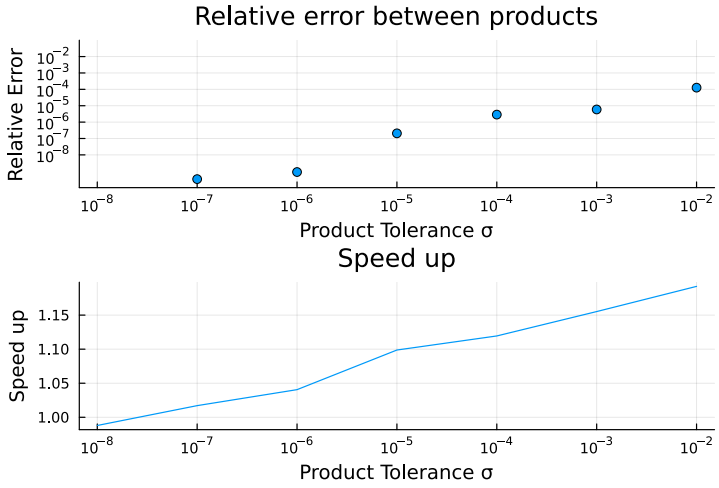
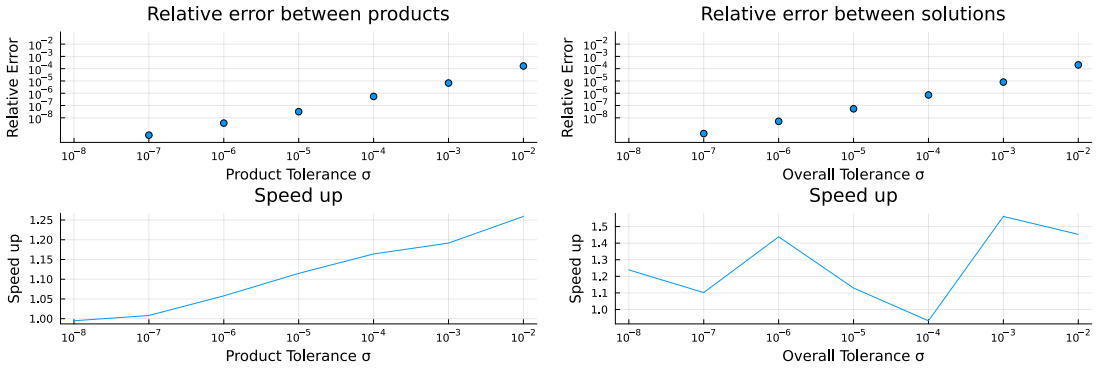


Figure 5.2: Speedup and residual evolution for the product between a 8000x8000 HMatrix and a vector.

5.3 Helmholtz's results

For a maximum speedup bound in the product, the infinity tolerance brought a () speedup.

For the unitary circle boundary the results can be seen in 5.3, for both the matrix-vector product and an application of the Inexact GMRES with few iterations.



(a) Results for the product of a 70000x 70000 HMatrix and a vector.

(b) Results for an initial application of the Inexact GMRES algorithm.

Figure 5.3: Results for the application of the Inexact GMRES algorithm with a 70000x70000 HMatrix.

5.3.1 Cavity scattering

For the cavity, we start out by running smaller examples and study the bounds found of chapter 2.

First we view the variations of the norms 2.23 and 2.24, shown in Figure 5.4.

For the bounds in 2.17, we have Figure 5.5.

As we can look from both Figure 5.4 and Figure 5.5, even if choosing $\ell_k = 1$ makes the inexact product's tolerance bigger, it still follows the restriction in the bound 2.17. A test with these different choices for the product tolerance did not show a big variation between the iterations necessary for each one of them.

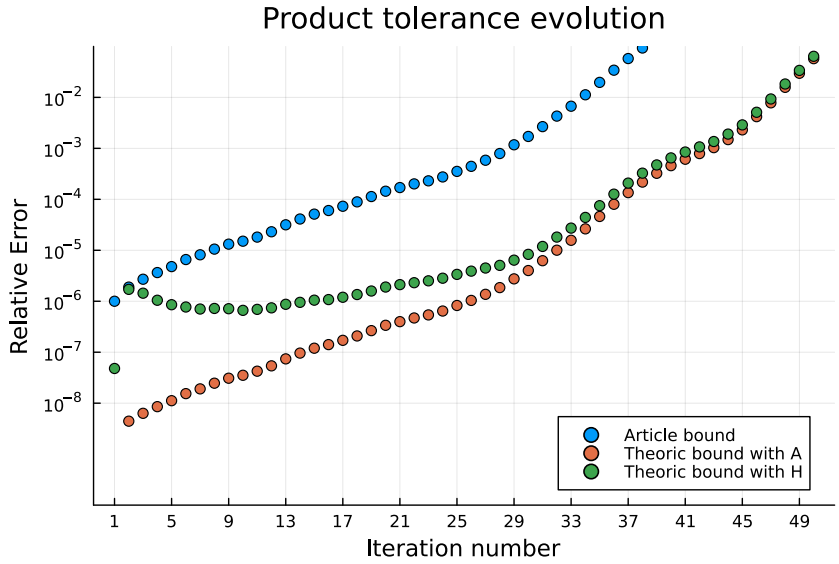


Figure 5.4: Evolution for each of the pertubation's bounds found in chapter 2.

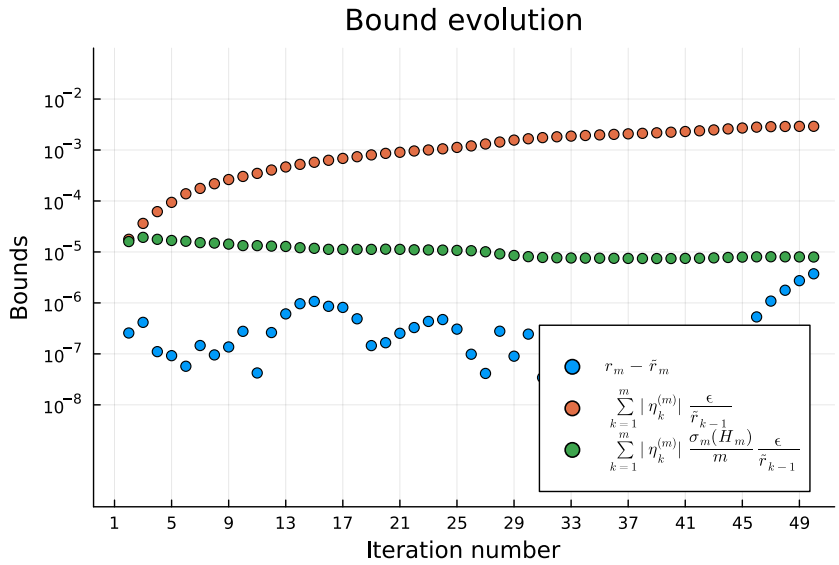


Figure 5.5: Evolution for each of the residual's bounds found in chapter 2.

It could mean less strict tolerances can still lead to the algorithm's convergence, what can be used to accelerate the algorithm even further.

As for the speedup results, we have Figure 5.6.

An evolution of the number of iterations in face of the different tolerances passed to the algorithm is in 5.7.

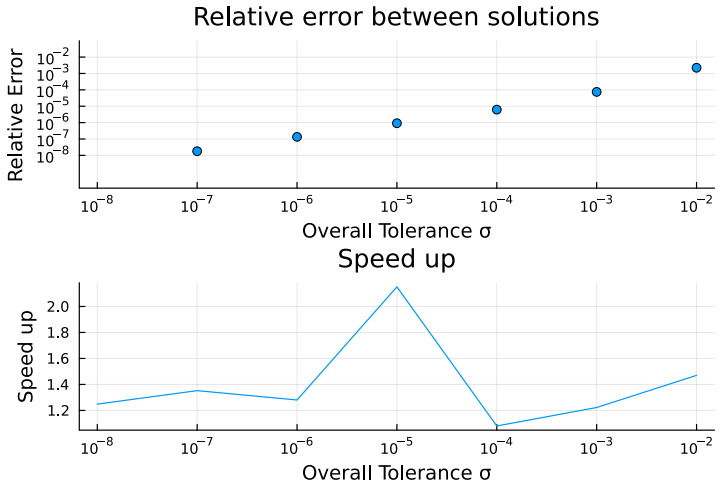


Figure 5.6: Speedup witnessed in the application of the Inexact GMRES in a 50000x50000 matrix.

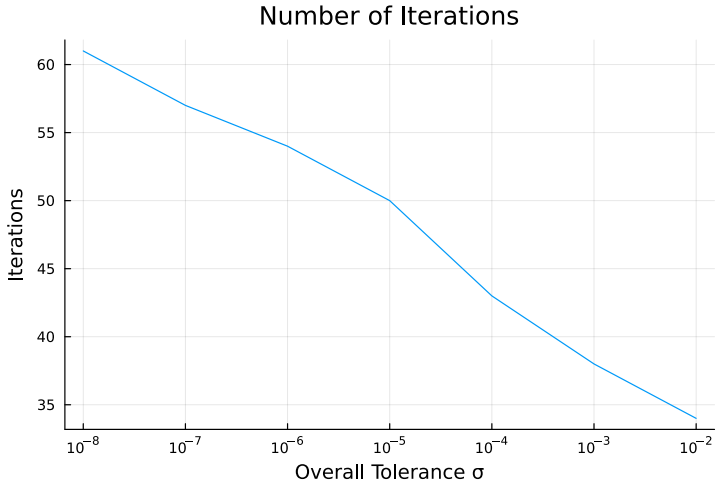


Figure 5.7: Evolution of the quantity of iterations needed for convergence and overall tolerance passed as an argument.

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

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

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

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

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

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

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

[11] 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.

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