Année Pré-doctorale de Recherche à l'Etranger

Politecnico di Torino

Internship report : $\mathcal{H}$-matrices and Adaptive Cross Approximation

Jessie Levillain

Tutor : Prof. Fabio Freschi

October 1st 2019 - June 30th 2020

Acknowledgements

I would like to express my deep gratitude to Fabio Freschi and Lionel Pichon for giving me this opportunity to do an internship in Turin. I am particularly grateful for Fabio's patient guidance at work, but also his and Vincenzo's advice and help in everyday life. Fabio's willingness to give his time so generously has been very much appreciated. Special thanks to Pia for the administrative work in Politecnico, and to Luca Giaccone for his help with GitHub. I would also like to thank the whole CADEMA team and PhDs for welcoming me and fully integrating me in the research group.

Abstract

This report deals with hierarchical methods to store matrices resulting from hybrid formulations of physical problems. The problem studied here is a nonlinear eddy current problem, solved numerically by using a coupling of boundary element methods and surface impedance boundary conditions. The method showed the benefits of the two formulations: the SIBC allowed the use of analytical or semi-analytical solution of the electromagnetic field in thin conductive layers, also in case of nonlinear material, while the BEM rigorously accounted for the regularity conditions of the magnetic field at infinity. The equations are translated in terms of a matrix equation which are then solved by the iterative solver GMRES. Because the numerical solution is based on the use of dense BEM matrices, both matrix filling and inversion times are critical. For this reason, matrices are stored as a hierarchical system of blocks which can be approximated by the use of low rank matrices. This $\mathcal{H}$-matrix theory enabled the use of algebraic matrix operations with almost linear complexity. The whole project was implemented and tested in MATLAB language, and matrix blocks were created taking into account both the geometric aspect of the physical problem, and hierarchical matrix theory.

Résumé

Ce rapport traite de méthodes hierarchiques pour stocker des matrices résultant de formulations hybrides de problèmes physiques. Le problème en question est un problème non linéaire lié aux courants de Foucault, résolu numériquement par un couplage d'une méthode d'élements finis de frontière et de conditions d'impédance de surface aux bords du domaine. Cette méthode comporte les avantages des deux formulations : les conditons d'impédance de surface permettent d'utiliser des solutions analytiques ou semi analytiques du champ électromagnétique dans les couches conductrices de fine épaisseur, tandis que la méthode des éléments finis de frontière retranscrit rigoureusement les conditions de régularité du champ magnétique à l'infini. Les équations résultantes sont ensuite traduites sous la forme d'un système matriciel, qui est résolu par une méthode itérative de généralisation de minimisation du résidu. Comme la solution est basée sur l'utilisation de matrices denses et de grande taille, leur remplissage ainsi que leur inversion posent problème au niveau de la complexité en temps. C'est pourquoi les matrices sont stockées selon un système hierarchique de blocs qui sont approximés par des matrices de faible rang. cette théorie des matrices hiérarchiques, ou $\mathcal{H}$-matrices, permet d'utiliser des opérations algébriques sur ces matrices tout en gardant une complexité en temps et en espace presque linéaire. L'ensemble du projet a été implémenté et testé en langage MATLAB, et les blocs hiérachiques ont été construits en prenant en compte à la fois la théorie des $\mathcal{H}$-matrices et l'aspect géométrique du problème physique.

Contents

Introduction

Physical models of real-world problems often lead to integral equations or to boundary value problems of elliptic partial differential equations. In the context of this report, boundary value problems problems are reformulated in terms of boundary integral equations. The physical model explored in this report is a nonlinear eddy current problem, which plays a key role in developing wireless power transmission devices. Since such problems can most often not be solved explicitly, their numerical solution is done by approximating the exact solution from finite dimensional spaces.

A numerical procedure, called Boundary Element Method (BEM), which has been developed in the 1950s, is often more convenient than the traditional Finite Element Method (FEM) and has become a powerful tool for numerical studies of physical phenomena in both bounded and unbounded domains. However, when it comes to the specific problem presented here, standard volume-based techniques are not suitable, as they require a large amount of elements in the mesh to detect the small penetration depth. That is why an hybrid method is presented here, using both BEM and surface impedance boundary conditions (SIBC), to avoid the fine discretization inside the conductive material. SIBC uses analytical or semi-analytical formulations of the electromagnetic phenomena instead, making the method efficient in multi-scale, multi-domain problems [1].

This numerical procedure, applied to boundary integral equations leads to a linear system of algebraic equations, with mostly dense matrices involved, i.e. whose entries do not vanish. This leads an asymptotically quadratic memory requirement for the whole procedure. Moreover, the success of these numerical methods coupled with the fast development of computers has led to a constant demand for higher accuracy and more sophisticated tasks, which entails the creation of very large dense matrices. A new way of using and storing these matrices thus needs to be used. Fortunately, all boundary element matrices can be decomposed into a hierarchical system of blocks which can be approximated by the use of low rank matrices. This hierarchical matrix structure leads to a matrix format called hierarchical matrices, or $\mathcal{H}$-matrices, which represents each entry of a fully populated matrix by low-rank matrices. Low-rank matrices are obtained by using various approximation methods, such as a truncated singular value decomposition (SVD) or more optimized routines such as adaptive cross approximation (ACA), which will be explored in chapter 2. In addition to efficiently storing matrices, $\mathcal{H}$-matrices provide approximate variants of the usual matrix operations such as addition, multiplication, and inversion with almost linear complexity.

After focusing on an eddy current problem leading to BEM-SIBC formulations, methods to hierarchically store the resulting matrices will be explored and implemented, so as to finally solve the original physical problem.

Chapter 1

Hybrid solutions for nonlinear eddy currents problems

In the last decade, the development of wireless power transmission devices increased the challenges of electromagnetic simulations. In particular, when high power is required, for example for automotive applications, the electromagnetic structure present many peculiar characteristics that do not fit with standard simulation tools:

- the frequency range involves low-medium frequencies, up to 100 kHz, due to the availability of suitable power electronics;
- the domain under study is characterized by open boundaries, whereas the active structures have limited extension;
- the thickness of active structures like car bodies is much smaller than other dimensions (order of millimeters compared with meters);
- the penetration depth is much smaller than the thickness.

For these reasons, ad-hoc formulations are required to efficiently and effectively solve such problems. In [1] the possibility of coupling nonlinear surface impedance boundary conditions (SIBC) with the boundary element method (BEM) was presented. The BEM-SIBC hybrid formulation was based on the definition of integral variables over two intertwined surface meshes, namely primal and dual meshes, linked by duality relations. This particular choice gave rise to a lumped circuit network of admittances in the conductive domain which was coupled with the boundary element formulated in terms of reduced scalar potential and its normal derivative. The method showed the benefits of the two formulations: the SIBC allowed the use of analytical or semi-analytical solution of the electromagnetic field in thin conductive layers, also in case of nonlinear material, while the BEM rigorously accounted for the regularity conditions of the magnetic field at infinity.

1.1 Preliminary notions

In this section, some basic laws and equations in electromagnetism will be given, without any proof. The reader is invited to refer to a physics course such as [2] for a more detailed approach. Then, we will dwell on a few other points, necessary to justify the BEM-SIBC hybrid formulation.

1.1.1 Some equations at the foundation of electromagnetism

In this section, a few basic equations of electromagnetism will be given, to be used later in this report.

1.1.1.1 Biot-Savart's law

The Biot-Savart Law relates magnetic fields to the currents which are their sources. Let C be a curve representing a coil, r' a point on C. The magnetic field $\overrightarrow{B}$ due to an element $\overrightarrow{dl}$ of a current-carrying wire in the air at a point r is given by :

$$\overrightarrow{B}(\overrightarrow{r}) = \frac{\mu_0}{4\pi} \int_C \frac{I\overrightarrow{dl} \wedge (\overrightarrow{r} - \overrightarrow{r'})}{|\overrightarrow{r} - \overrightarrow{r'}|^3} \tag{1.1}$$

where $\mu_0 = 4\pi \times 10^{-7} T \cdot m/A$ is the permeability of free space, and I the current going though the coil.

1.1.1.2 Maxwell's equations and Eddy currents

Maxwell's equations express the fluxes and circulations of the electric and magnetic field vectors in either integral or differential form. They will be presented here in the latter form.

1. Faraday's law of induction : Time-varying magnetic fields create an electric field.
$$\nabla \times \overrightarrow{E} = -\frac{\partial \overrightarrow{B}}{\partial t}; \tag{1.2}$$
2. Ampère-Maxwell law : Steady currents and time-varying electric fields create a magnetic field.
$$\nabla \times \overrightarrow{B} = \mu_0 \overrightarrow{J} + \frac{1}{c^2}\frac{\partial \overrightarrow{E}}{\partial t}; \tag{1.3}$$
3. Gauss's electric law : Electric charges create an electric field, whose electric flux across a closed surface is proportional to the charge enclosed.
$$\nabla \cdot \overrightarrow{E} = \frac{\rho}{\varepsilon_0} \tag{1.4}$$
where ε_0 is the electric constant, and ρ the total electric charge density;
4. Gauss' magnetic law : There are no magnetic monopoles, and the magnetic flux in an enclosed surface is naught.
$$\nabla \cdot \overrightarrow{B} = 0. \tag{1.5}$$

Eddy currents are induced currents caused by a varying magnetic field in a conductive region. Their existence and nature can be found thanks to Faraday's law of induction.

1.1.2 Green's formula

This section is a quick reminder on some useful integration theorems [3], which will be used in 1.3.

Theorem 1.1.1 (Green's formula)**.** *Let Ω be an open, bounded $\mathcal{C}^1$ subset of $\mathbb{R}^n$, $w \in C^1(\bar{\Omega})$. Then w verifies Green's formula:*
$$\int_{\Omega} \frac{\partial w}{\partial x_i}(x)dx = \int_{\partial\Omega} w(x)n_i(x)ds \tag{1.6}$$
where n_i is the ith component of the unitary exterior normal vector n.

Corollary 1.1.1.1 (Integration by parts)**.** *Let Ω be an open, bounded $\mathcal{C}^1$ subset of $\mathbb{R}^n$, $u, v \in C^1(\bar{\Omega})$. Then, u and v verify :*
$$\int_{\Omega} u(x)\frac{\partial v}{\partial x_i}dx = -\int_{\Omega} v(x)\frac{\partial u}{\partial x_i}dx + \int_{\partial\Omega} u(x)v(x)n_i(x)ds. \tag{1.7}$$

Theorem 1.1.2 (Stokes' theorem)**.** *Let Ω be an open, bounded $\mathcal{C}^1$ subset of $\mathbb{R}^n$, $\phi \in C^1(\bar{\Omega}, \mathbb{R})$ a scalar function and $\sigma \in C^1(\bar{\Omega})$. Then :*
$$\int_{\Omega} \nabla \cdot \sigma(x)\phi(x)dx = -\int_{\Omega} \sigma(x)\cdot\nabla\phi(x)dx + \int_{\partial\Omega} \sigma(x)n(x)\phi(x)ds. \tag{1.8}$$

1.2 Tonti diagrams and numerical methods for electromagnetics

Unless stated otherwise, in this section, n, p will always be positive integers, n representing the dimension of the considered space.

1.2.1 Classifying physical variables

Tonti diagrams are classification diagrams, i.e. graphical representations of physical theories in which the role and relationships of variables, space and time entities, as well as physical and material laws are explicitly represented [4]. In order to define these diagrams, physical variables first need to be classified in three categories:

- Configuration variables, which describe the configuration of the field or system;
- Source variables, which describe the sources of the field or the forces acting on the system;
- Energy variables, which are obtained as the product of a configuration and a source variable.

Variables from the same category are linked to one another by sum, difference, limit, differentiation and integration operations. Constitutive equations then link configuration with source variables, while also giving material and system parameters.

Moreover, some entities do not depend on local quantities and are naturally and intrinsically associated with space and time entities. They are denoted as global variables, and defined as domain functions. While local quantities are discontinuous at interfaces, global variables are naturally continuous.

1.2.2 Cell complexes

Differential formulations of physical laws such as in 1.1.1.2 are based on coordinate systems in which local scalar or vector functions are defined. However, when dealing with formulations based on global variables, one has to work on the cell complexes on which they are defined.
A cell complex is a subdivision of a space region into small elements called cells, or elements in numerical analysis. In this paper, the term cell will be used, and consequently the term cell complex, because this is the name given in algebraic topology, where a complete theory has been developed on this topic. In the algebraic formulation of physics cell complexes play the same role that coordinate systems play in the differential formulation. Cell complexes offer all the space elements needed for the algebraic formulation: points (vertices), lines (edges), surfaces (faces) and volumes (cells) [4].

Definition 1.2.1 (p-cells)**.** Vertices are called zero-dimensional cells, or 0-cells for short; edges, 1-cells; faces, 2-cells; volumes, 3-cells. These p-cells will be denoted by e^0, e^1, e^2, e^3 respectively, as shown in table 1.1.

Symbol	Space element	Cell complex	Algebraic topology
P	Point	Vertex	0-cell e^0
L	Line	Edge	1-cell e^1
S	Surface	Face	2-cell e^2
V	Volume	Cell	3-cell e^3

Table 1.1: Symbols and name of space elements

A cell complex can be created in two ways [4]:

1. Discretizing a space region by using the coordinate surfaces of a coordinate system, whichever the coordinate system. A coordinate cell complex obtained in this way is useful for deducing the differential formulation of physical equations from the algebraic formulation;

2. Discretizing a space region by subdividing it into elements of an arbitrary shape. The simplest subdivisions are formed by tetrahedra (in space) and triangles (in a plane). Since these elements are the simplest polyhedra and polygons, respectively, they are called simplices, and a cell complex formed by these simplices is called a simplicial complex.

In this project, only simplicial complexes will be used.

1.2.2.1 Dual cell complex

In the differential formulation, physical laws are expressed by differential operators, meaning that a differential equation implies a neighborhood of every point and not only the point itself. A neighborhood acts as an auxiliary region of the point, with an undefined extension. In a discrete formulation, this auxiliary region can be identified with the 'stencil' used in the forward and backward differences. As a general principle, the algebraic formulation requires a surrounding auxiliary region around the nodes of a cell complex. Thus, for every node of a cell complex, one can consider an auxiliary region around it. All these auxiliary regions constitute another cell complex, called a dual cell complex of the complex [4].
If the primal cell complex is denoted as K, then its dual will be denotes as $\tilde{K}$. It is a matter of convenience to consider the first complex as primal and the second as dual, or vice versa. From a physics point of view, it is often more convenient to consider as dual that complex whose cells contain the sources of the field.

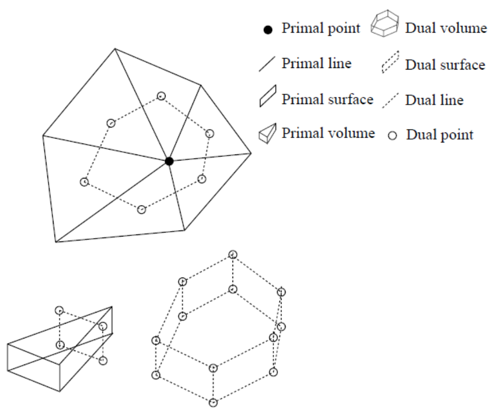


Figure 1.1: Primal and dual cells for a simplicial complex [5]

In this project, the dual of the simplicial complex considered later on is created using barycentric cells. In the barycentric subdivision of a plane cell complex, the dual of a 1-cell is composed of two line segments connecting the midpoint of the 1-cell with the two barycenters of the adjacent cells. In other words, the dual 1-cell is not a straight line [4]

1.2.2.2 Orientation of a p-cell

According to [5], the inner-orientation of a space entity can be defined as:

- A point P, which has inner orientation if it is defined as a sink $(+)$ or a source $(-)$;
- A line L, which has inner orientation if its boundary has inner orientation (i.e. a direction has been defined on it);
- A surface S, which has inner orientation if its boundary has inner orientation;
- A volume V , which has inner orientation if its boundary has inner orientation.

It should be noted that, while the definition of inner orientation is independent of the number of dimensions, the outer orientation depends on the number of dimensions in which the space element is embedded, as defined as :

- A volume $\tilde{V}$, which has outer orientation if normals have been defined (i.e inside and outside can be distinguished);
- A surface $\tilde{S}$, which has outer orientation if one side is labeled $+$ and the other $-$ (i.e. the direction of flow through it can be distinguished);
- A line $\tilde{L}$, which has outer orientation if a direction of rotation around it has been fixed;
- A point $\tilde{P}$, which has outer orientation if all lines originating from it have an outer orientation.

Given a cell complex, one can assign an inner orientation to all its p-cells as follows:

- All 0-simplices can be considered either as sinks $(+)$ or sources $(-)$: sinks are the common choice;
- All n-simplices must then be given an inner orientation. One of the two possible orientations of one n-simplex is chosen and propagated to all n-simplices. The propagation criterion is provided by the Möbius law of edges [6] two adjacent n-cells have compatible orientations when they induce opposite orientations on their common face.
- For cells of dimension $p \notin \{0, n\}$, a compatible orientation cannot, in general, be given, and orientation must be chosen arbitrarily for every p-cell. In computational physics, it is customary to orient 1-cells from the node with the smaller label to the node with the greater label.

A cell complex has an inner orientation when all the p-cells, with $0 \leq p \leq n$ have been oriented. Since physical variables refer to space elements endowed with an orientation, it will be useful to assign an orientation to all p-cells of a cell complex. If an inner orientation is assigned to every p-cell of the primal complex, this automatically induces an outer orientation on the dual cells of dimension $(n - p)$.

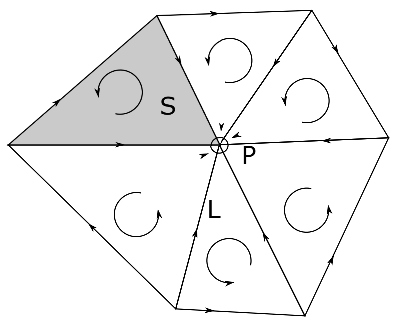


Figure 1.2: Oriented simplicial complex

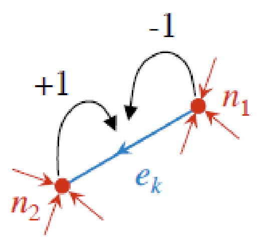
Definition 1.2.2 (Chain and cochains)**.** Let a general domain be partitioned by a set of p-cells c_k, each one endowed with its own orientation (inner/outer), denoted by an incidence number n_k 1.2.2.4. The aggregate of the p-cell c_k and the incidence number n_k is called chain :

$$\sum_k c_k n_k.$$

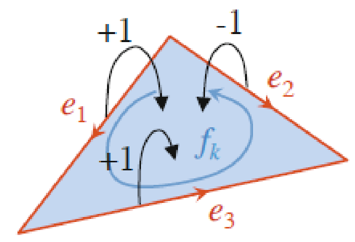
Given a global physical quantity associated with a generic p-cell of a cell complex, the knowledge of the amount of a physical variable associated with a set of p-cells is called p-cochain.

1.2.2.3 Cofaces and coboundary

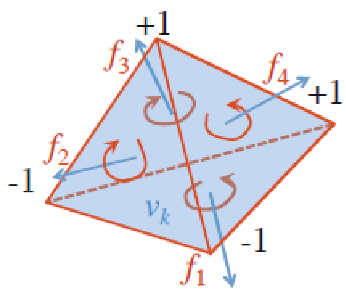
Definition 1.2.3 (Cofaces and coboundary)**.** The faces of a p-cell are those $(p-1)$-cells that are incident on that cell. The $(p+1)$-cells that share a generic common p-cell are called cofaces. The set of all the cofaces of a p-cell forms the coboundary of that cell.



(a) $e_k = (-1)n_1 + (+1)n_2$



(b) $f_k = (+1)e_1 + (-1)e_2 + (+1)e_3$



(c) $v_k = (-1)f_1 + (-1)f_2 + (+1)f_3 + (+1)f_4$

Figure 1.3: Coboundary process [5]

Starting from a p-cochain, a procedure called the coboundary process gives rise to a $(p+1)$-cochain in the following way :

- First, the value of every physical variable associated with any p-cell is transferred to all the cofaces of the p-cell, with the correct incidence number;
- Then, for every $(p+1)$-cell all these values are added together. This gives rise to a new physical variable, with the same physical dimensions, associated with every $(p+1)$-cell.

The coboundary process is a purely topological operation, thus unrelated to specific physical theories but common to different problems. As a consequence, in a multi-physics problem, the discrete operators that are representative of a gradient, curl and divergence remain the same for all the problems.

1.2.2.4 Incidence numbers and incidence matrices

The notion of incidence numbers between nodes and branches and between branches and meshes is introduced in analogy to the theory of oriented graphs. Incidence numbers have values in $\{-1, 0, +1\}$. Their sign indicates whether the mutual orientations of two p-cells are compatible or not, while their number gives an information about whether those p-cells are mutually incident [4]. Incidence numbers can be collected to form incidence matrices. The notion of incidence number can be extended both to a primal complex endowed with an inner orientation and to a dual complex. All the incidence number of a cell complex can be assembled in a matrix, leading to the following discrete counterparts of the continuous differential operators:

- $G, \tilde{G}$, primal and dual edge-node matrices, corresponding to a discrete gradient;
- $C, \tilde{C}$, primal and dual face-edge matrices, corresponding to a discrete curl;
- $D, \tilde{D}$, primal and dual volume-face matrices, corresponding to a discrete divergence.

Some equations between those matrices then follow :

$$\tilde{D}_{n_{\tilde{V}} \times n_{\tilde{S}}} = -G^T_{n_P \times n_L} \tag{1.9}$$

$$\tilde{C}_{n_{\tilde{S}} \times n_{\tilde{L}}} = C^T_{n_L \times n_S} \tag{1.10}$$

$$\tilde{G}_{n_{\tilde{L}} \times n_{\tilde{P}}} = D^T_{n_S \times n_V} \tag{1.11}$$

where n_P, n_L, n_S and n_V are the numbers of points, lines, surfaces and volumes in the cell complex and $n_{\tilde{P}}, n_{\tilde{L}}, n_{\tilde{S}}$ and $n_{\tilde{V}}$ are the number of points, lines, surfaces and volumes in its dual complex.

Remark The boundary of a boundary is an empty set.

1.2.3 Principles of the Classification Theory

Theorem 1.2.1 (First Principle of the Classification Theory [5])**.** *1. Global configuration variables are associated with space and time elements endowed with inner orientation;*

2. Global source variables are associated with space and time elements endowed with outer orientation

This principle is justified by the fact that source variables are used in balance equations which require the definition of normals.

Theorem 1.2.2 (Second Principle of the Classification Theory)**.** *In every physical theory there are physical laws that link the global variables that refer to an oriented space-time element with others that refer to its oriented boundary (for instance Stokes and divergence theorems).*
Physical laws of this type are "topological equations", link physical variables of the same kind, as defined in 1.2.1, and do not involve metrical notions, nor material parameters. They are thus valid in any medium.

1.2.4 Tonti diagrams

The previous sections lead to the construction of classification diagrams, i.e. graphical representations of physical theories in which the role and relationships of variables, space and time entities, as well as physical and material laws are explicitly represented [7].

This kind of diagram has a general form like in 1.4 and its features are:

- Constitutive relations that go from front to back layers represent irreversible laws;
- On the left side, quantities represent primal space and time entities, while dual ones are on the right side;
- Different vertical level refer to different space entities;
- Vertical connections represent topological operators (partial differential operators);
- Front-to-back connections on the same side represent the time difference action (time derivatives);
- Front-to-back connections on opposite sides stand for irreversible material laws (constitutive equations);
- Horizontal connections on the same plane represent reversible material laws (constitutive equations);

A more mathematical way to view this [8] would be to define a vertical connection by a derivation, a horizontal one by a Hodge star operator [9] and to associate to each level a p-form [10], see appendix F for more information on these concepts. These notions of algebraic topology will not be detailed here, as it is not the main goal of this project.

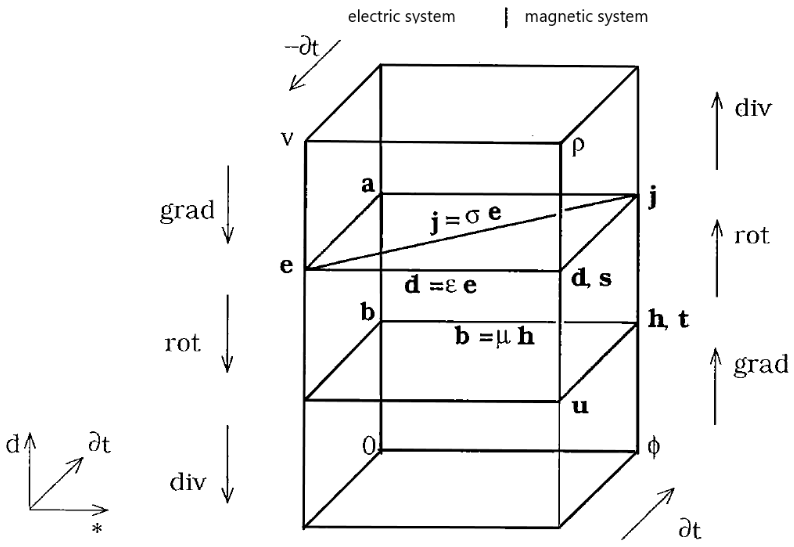


Figure 1.4: Example Tonti diagram for electromagnetism [8]

1.2.5 Resulting numerical scheme

The notions explained in the previous section lead to a technique called the "Cell Method", which leads directly to a numerical scheme. The Cell Method (CM) can be done in five steps [5]:

1. First the problem is expressed in terms of domain quantities (global variables). This follows from the first and second principles of the classification theory presented in the previous chapter;
2. The previous step allows the construction of the Tonti diagram for the problem at hand. The coboundary process states which matrices connect the global variables at different levels of the diagram, while the construction of the material matrices can be done using constitutive equations of the problem. This process is detailed in [5];
3. Then, matrix equations are taken directly from the Tonti diagram ;
4. Using the previous equations and the relations between the operators from equation 1.11, a final system of linear equations is obtained;
5. After the previous system has been solved by means of a suitable numerical technique for large sparse algebraic systems of equations, other quantities can be obtained very easily.

1.2.6 Hybrid BEM-SIBC numerical scheme

The method chosen in section 1.3 uses the same kind of mesh, and equations from the Tonti diagram for the SIBC part, but sill differs slightly in the other half. Indeed, the second part of the following hybrid method is the BEM, which only discretizes the boundary of the domain of interest. BEM is a variation of the Finite Element Method (FEM), which requires a fundamental solution to the partial differential equation (PDE), but works well on infinite or semi-infinite domains, while giving rise to smaller matrices than the FEM. This kind of method has first been used in [11], but presents challenges for nonlinear problems, which will be explored in the next sections.

BEM schemes are applicable in boundary value problems for a PDE, with an explicit fundamental solution that is know:

$$\Delta u = 0 \text{ in a domain } \Omega \subset \mathbb{R}^n$$
$$u = g \text{ on the boundary } \Gamma = \partial\Omega.$$

This solution then needs to be represented in the domain by means of boundary potentials, which are actually well-known in our case, for electromagnetics. The representation formulas give the solution in the interior of the domain. If one takes boundary values in the representation formula, one obtains boundary integral equations. One then assumes that Γ can be decomposed into a finite number of subsets, each of which has a regular parameter representation by some parameter domain in $\mathbb{R}^{n-1}$. For the numerical solution, a system of discrete equations on the boundary elements is then computed. However, if the integral equation is of the form $Au = f$ on Γ, convergence proofs ans asymptotic error estimates are only available under the assumption that A verifies certain conditions of ellipticity [12].

1.3 Description of the electromagnetic system and its resulting equations

This section is largely based on [1] which is the base that gives rise to this project. The process detailed below has for sole purpose to fully introduce the problem of interest. Convergence of the methods or existence or solutions will only be assumed using the guarantee that electromagnetic equations such as the ones used here are sufficiently smooth given their intrinsic definition, and that the CM converges towards a solution of this problem, as it does for similar examples in [4, 5].

1.3.1 Description of the problem

In this report, the domain under study can be simple examples such as a filamentary coil on a conductive sphere, or a more complex one, based on real-life problems, such as a car body. Either way, it can be divided in three complementary regions :

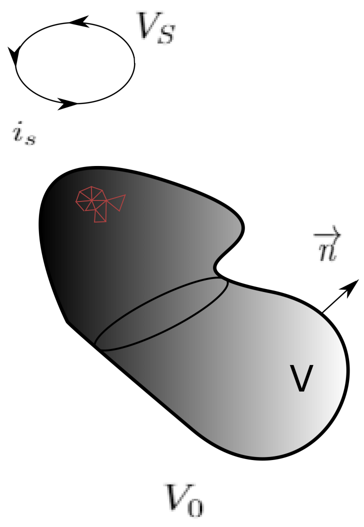


Figure 1.5: Domain division: known sources V_S , eddy currents V and non-conductive V_0 regions.

- the source domain, V_S with imposed currents, e.g., filamentary coils with known currents and/or massive conductors with known current density distribution;
- the domain V where eddy currents are induced, with finite conductivity σ and permeability μ_R (possibly nonlinear);
- the non conductive ($\sigma = 0 \quad S/m$) unbounded domain V_0, rigorously accounted by the Green's formula in the integral formulation.

The study is restricted to cases where the penetration depth (either in the linear and in the nonlinear cases) is at least 5 times smaller when compared to the geometric dimensions of the conductive region so that the current density profile is completely developed. It is thus possible, under this hypothesis, to consider only the boundary $S = \partial V$ of V. This boundary is discretized with a simplicial mesh $\mathcal{G}$, referred to as primal mesh , made of oriented faces, edges and nodes. From this mesh it is possible to derive a secondary mesh $\tilde{\mathcal{G}}$, namely dual mesh , obtained by barycentric subdivision of $\mathcal{G}$, where dual nodes are located in correspondence of the primal face barycenters, dual edges connect adjacent nodes and pass through the primal edge mid-points 1.2. Dual faces are built onto these dual edges and extend through the depth of V until all the field quantities vanish. The orientation of the dual geometric entities in $\tilde{\mathcal{G}}$ is induced by the corresponding elements in $\mathcal{G}$ [1]. Figure 1.6 shows these primal and dual complexes.

1.3.2 Electromagnetic variables

The electromagnetic variables are defined following the previous structure :

- the electric voltage is defined along the primal edges L by

$$e_k = \int_{L_k} \overrightarrow{E_0} \cdot \overrightarrow{dL},$$

where $\overrightarrow{E_0}$ is the value of the electric field on the surface of the conductive object;

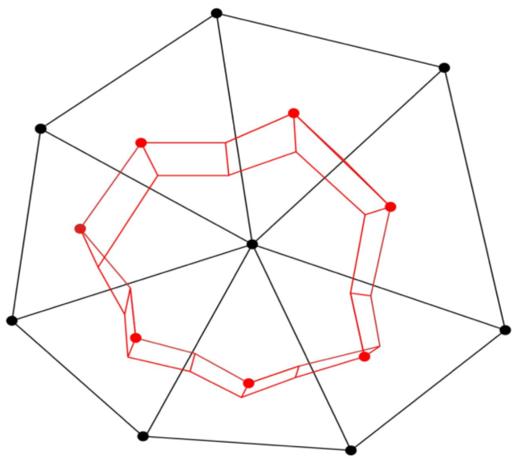


Figure 1.6: Primal (black) and barycentric dual (red) mesh

- the magnetic flux is defined through primal faces S by

$$b_m = \int\limits_{S_m} \overrightarrow{B_0} \cdot \overrightarrow{dS},$$

where $\overrightarrow{B_0}$ is the surface magnetic flux density;

- the magnetic scalar potential ψ is defined on dual nodes, while its normal derivative with respect to the surface mesh $\frac{\partial \psi}{\partial n} = \partial_n \psi$ is assumed uniform over primal faces;
- the magneto-motive forces are defined along the dual edges $\tilde{L}$:

$$h_k = \int\limits_{\tilde{L}_k} \overrightarrow{H_0} \cdot \overrightarrow{dL},$$

with $\overrightarrow{H_0}$ the surface magnetic field. A magneto-motive force is a physical force generating a magnetic flux;

- the electric current i_k is obtained by integration of the current density $\overrightarrow{J}(z)$ through the generic dual face $\tilde{S}_k$:

$$i_k = \int\limits_{\tilde{S}_k} \overrightarrow{J}(z) \cdot \overrightarrow{dS},$$

$\overrightarrow{J}(z)$ depends on the position z along the depth of V to account for the magnetic diffusion.

1.3.3 BEM formulation for the unbounded region

The homogeneous unbounded region V_0 is external to the magnetic-conductive domain. Its fields are thus analyzed with a standard BEM formulated in terms of reduced magnetic scalar potential ψ and its normal derivative $\partial_n \psi$. Indeed, in an unbounded and homogeneous region such as V_0 or the air, the induced magnetization can be represented as the gradient of a scalar potential, called the reduced scalar potential [13, 14]. This reduced potential satisfies the equations $\overrightarrow{H} = -\nabla\psi$ and $\nabla^2\psi = 0$. By defining the free space Green's function $G(\overrightarrow{r}, \overrightarrow{r'}) = \frac{1}{4\pi|\overrightarrow{R}|}$ where $\overrightarrow{R} = \overrightarrow{r} - \overrightarrow{r'}$ is the vector from the source to the observation point, and applying the second Green's theorem to the Laplace equation $\nabla^2\psi = 0$, it follows that :

$$c(\overrightarrow{r'})\phi(\overrightarrow{r'}) = \int\limits_{S=\partial V} G(\overrightarrow{r}, \overrightarrow{r'})\partial_n\psi(\overrightarrow{r})d\Gamma - \int\limits_{S=\partial V} \psi(\overrightarrow{r})H(\overrightarrow{r}, \overrightarrow{r'})d\Gamma, \tag{1.12}$$

where $c(\overrightarrow{r'})$ is the free term coefficient and can be interpreted as the faction of $\psi(\overrightarrow{r'})$ that lies inside V, i.e. its value depends on the problem and on the position of $\overrightarrow{r'}$ in the chosen geometry. [15] If the vector $\overrightarrow{r'}$ is directed to a point of S lying on a smooth surface, then $c(\overrightarrow{r'}) = \frac{1}{2}$. $H(\overrightarrow{r}, \overrightarrow{r'}) = \frac{\partial G(\overrightarrow{r}, \overrightarrow{r'})}{\partial n(\overrightarrow{r})} = \nabla G(\overrightarrow{r}, \overrightarrow{r'}) \cdot \overrightarrow{n}(\overrightarrow{r})$, is the normal derivative of $G(\overrightarrow{r}, \overrightarrow{r'})$. A system of linear, algebraic equations is obtained after discretizing the closed boundary S with triangular elements, as explained in 1.3.2, and applying equation 1.12

$$-H\psi + W\partial_n\psi = 0 \tag{1.13}$$

with for any elements i, j the generic entries of the matrices H and W :

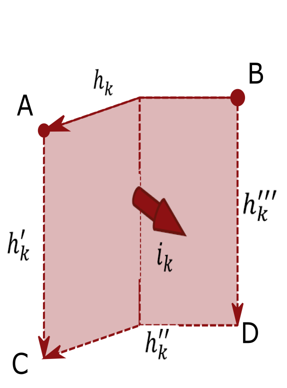
$$\begin{aligned} H_{i,j} &= \tfrac{1}{2}\delta_{ij} - \int\limits_{S_j} \tfrac{\partial G(\overrightarrow{r_i},\overrightarrow{r_j})}{\partial n} dS = \tfrac{1}{2}\delta_{ij} - \tfrac{1}{4\pi}\int\limits_{S_j} \nabla\left(\tfrac{1}{|\overrightarrow{r_i}-\overrightarrow{r_j}|}\right)\cdot\overrightarrow{n_j}dS \\ W_{i,j} &= \int\limits_{S_j} G(\overrightarrow{r_i},\overrightarrow{r_j})dS = \tfrac{1}{4\pi}\int\limits_{S_j}\tfrac{1}{|\overrightarrow{r_i}-\overrightarrow{r_j}|}dS \end{aligned} \tag{1.14}$$

where δ_{ij} is the Kronecker operator. Surface integrals of the Green's function and its derivative defined in the previous equations can be analytically computed, this is described in detail in [16, 17, 18] but will not be addressed here.

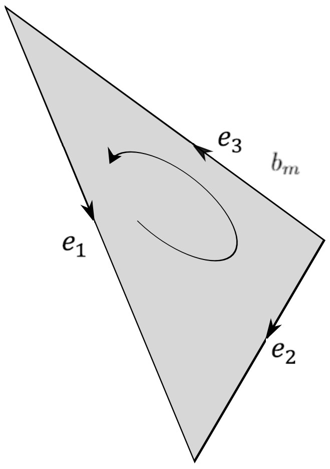
1.3.4 SIBC formulation in terms of integral variables

To define the SIBC in terms of integral variables, Ampère's and Faraday's laws are used. Ampère's law in integral form evaluated on the kth dual face $\tilde{S}_k$ reads :

$$\oint\limits_{\tilde{L}_k=\partial\tilde{S}_k} \overrightarrow{H}\cdot\overrightarrow{dL} = \int\limits_{\tilde{S}_k}\overrightarrow{J}\cdot\overrightarrow{dS} = i_k \tag{1.15}$$



(a)



(b)

Figure 1.7: (a) Ampère 's law applied on dual surfaces; (b) Faraday 's law imposed on primal surfaces.

The circulation of the magnetic field can be expressed as the sum of the magneto-motive forces along the boundary $\tilde{L}_k$ of $\tilde{S}_k$. Using the notations defined in figure 1.7: $h_k + h'_k - h''_k - h'''_k = i_k$.
However, the magneto-motive forces along the segments BC and DA are zero because the line is orthogonal to the magnetic field parallel to the surface. In addition, because the segment CD lies in the region where all fields are zero, its contribution to the magnetic field circulation vanishes and the only active path is along the surface dual edge AB [1]:

$$h'_k = h''_k = h'''_k = 0 \Rightarrow h_k = i_k$$

or, in matrix form

$$h = i. \tag{1.16}$$

For Maxwell's equations in frequency domain at the angular frequency ω, the Faraday's law applied to a triangular face S_m in $\mathcal{G}$ is :

$$\oint\limits_{L_m=\partial S_m}\overrightarrow{E}\cdot\overrightarrow{dL} =_j \omega\int\limits_{S_m}\overrightarrow{B}\cdot\overrightarrow{dS} = -j\omega b_m \tag{1.17}$$

Using the definition of the electric voltage, the circulation of the electric field can be written :

$$e_1 - e_2 + e_3 = \sum_{k=1}^{N_{edges}} c_{mk}e_k = -j\omega b_m, \tag{1.18}$$

where $c_{mk} \in \{-1, 0, +1\}$ is the incidence number of the kth edge with respect to the mth face in $\mathcal{G}$ (figure 1.7). Collecting all coefficients c_{mk} in the matrix C that represents the discrete counterpart of the differential curl operator as defined in 1.2, Faraday's law reads

$$Ce = -j\omega b. \tag{1.19}$$

1.3.4.1 Constitutive equation : linear case

Constitutive equations link variables defined on the primal mesh $\mathcal{G}$ with those defined on the dual mesh $\tilde{\mathcal{G}}$. Using integral variables, constitutive equations are represented by square matrices that encode the material properties and the metric of the problem [4]. This formulation requires a link between the currents i through the dual faces, defined in equation 1.16, and the voltages e along the primal edges, in equation 1.19. The current can be calculated by integration of the current density through dual faces :

$$i'k = \int_{S'_k} \sigma \overrightarrow{E}(z) \cdot \overrightarrow{dS} = \int_0^\infty \int_{L'_k} \sigma \overrightarrow{E}(z) \cdot \overrightarrow{dL} dy \tag{1.20}$$

The prime ($\prime$) symbol indicates that this is a partial contribution coming from the portion of the dual face that lies inside a single triangle (which will then be assembled with that of adjacent triangle hinged on the same edge). The electric field can be expressed as: $\overrightarrow{E}(z) = \overrightarrow{E_0} f(z)$, where $\overrightarrow{E_0}$ is the surface electric field and $f(z)$ accounts for the spatial decay of the electric field along the material depth z . In the linear case, $f(z)$ can be obtained analytically [19], thus: $\overrightarrow{E}(z) = \overrightarrow{E_0}(z)e^{-(1+j)\frac{z}{\delta}}$, where $\delta = \sqrt{2/(\omega\mu\sigma)}$ is the penetration depth.
In the surface triangular mesh, $\overrightarrow{E_0}$ is interpolated through edge elements $\overrightarrow{\omega_m}$ [20], using the voltages e_m along the primal edges as weights : $\overrightarrow{E_0} = \sum_{m=1}^{3} \overrightarrow{\omega_m} e_m$. Thus, using equation 1.20, this becomes :

$$i'_k = \sum_{m=1}^{3} e_m \int_0^\infty f(z)dz \int_{l'_k} \sigma \overrightarrow{\omega_m} \cdot \overrightarrow{dL} = \sum_{m=1}^{3} e_m \frac{\delta}{1+j} m'_{km} = \sum_{m=1}^{3} Y'_{km} e_m \tag{1.21}$$

Where the term $m'_{km} = \int_{\tilde{L}'_k} \sigma \overrightarrow{\omega_m} \cdot \overrightarrow{dL}$ is the classic coefficient of the constitutive matrix in 2d using edge elements.
Assembling element-by-element the admittance matrix Y using the entries Y'_{km} :

$$i = Ye. \tag{1.22}$$

In the linear case, the entries of the admittance matrix may be obtained as the entries of the conductance matrix of a triangular mesh using the equivalent material property $\sigma/(1 + j)$ and the penetration depth δ as thickness.

1.3.4.2 Constitutive equation : nonlinear case

In a nonlinear magnetic material, local waveforms are periodic but not sinusoidal, which makes the use of complex quantities incorrect. To over- come these problems and make use of the complex representation of quantities, it is possible to introduce a fictitious inhomogeneous material that accounts for the nonlinear relationship of the field quantities. This material is described by an equivalent magnetic characteristic [21]. Several methods are available to solve this kind of problems, but here, only the first harmonic equivalent characteristic will be considered to reconstruct the distorted periodic solution. The choice is made to make a fair comparison of the results with the reference solution provided by the software FEMM [22] [1] that implements the first harmonic equivalent magnetic characteristic.
Assuming a purely sinusoidal magnetic field $H(t) = \hat{H}cos(\omega t)$, the equivalent characteristic is built considering only the first harmonic $\hat{B}_1$ of the distorted magnetic flux density $B(t)$:

$$\mu_{eq}(H) = \frac{\hat{B}_1}{\hat{H}} = \frac{\frac{2}{T}\int_0^T B(t)cos(\omega t)dt}{\hat{H}} \tag{1.23}$$

The procedure is then repeated for all values of $\hat{H}$ of the original magnetic characteristic.
In the nonlinear case, the function $f(z, H_0)$ describing the electric field depends on the strength of the tangential surface magnetic field H_0 and, consequently, its expression and that of the integral factor $F(H_0)$ cannot be

[1]FEMM is a suite of programs for solving low-frequency electromagnetic problems on two-dimensional planar and asymmetric domains. The program currently addresses linear/nonlinear magnetostatic problems, linear/nonlinear time harmonic magnetic problems, linear electrostatic problems, and steady-state heat flow problems.

determined analytically. To overcome this problem, a nonlinear 1-dimensional diffusion problem is solved for different values of tangential component of the surface magnetic field H_0 :

$$\frac{dH}{dz} = -\sigma E(z) \tag{1.24}$$

$$\frac{dE}{dz} = -j\omega\mu_{eq}(H)H(z) \tag{1.25}$$

with the boundary conditions $H(z=0) = H_0$ and $H(z \to \infty) = 0$.

Solving this type of equations is a classical problem, which is not detailed here. Having then solved 1.25, the nonlinear actor $F(H_0)$ is defined as :

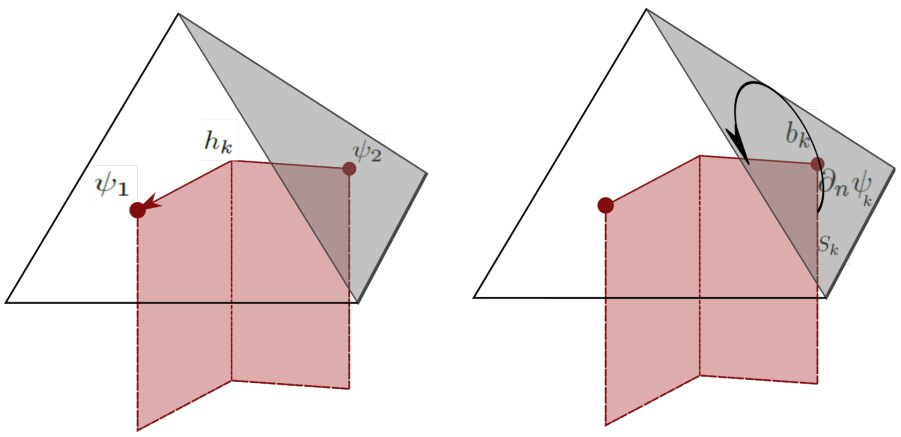
$$F(H_0) := \int\limits_0^\infty f(z, H_0)dz = \frac{1}{E_0(H_0)} \int\limits_0^\infty E(z, H_0)dz \tag{1.26}$$

Under the assumption of using the equivalent nonlinear characteristic identified by equation 1.23, at a specific working frequency, any nonlinear material can be characterized by a nonlinear complex factor $F(H_0)$ [1].

1.3.5 Interface conditions

Interface conditions impose the continuity of the magnetic field and magnetic flux density at the interface between the BEM and the SIBC formulations. The magnetic field at a point lying on the surface of the region V comes from two contributions: one originated by the known sources $\overrightarrow{H_S}$ and the other created by the magnetization $\overrightarrow{H_M}$: $\overrightarrow{H} = \overrightarrow{H_S} + \overrightarrow{H_M}$. In the air region, the magnetization field $\overrightarrow{H_M}$ is expressed as the gradient of the reduced magnetic scalar potential ψ : $\overrightarrow{H_M} = -\nabla\psi$, whereas the contribution of the sources in V_S is evaluated using the Biot-Savart's law :

$$\overrightarrow{H_S} = \frac{1}{4\pi} \int\limits_{\tilde{L}_k} \frac{\overrightarrow{J} \times \overrightarrow{R}}{|\overrightarrow{R}|^3} dV \tag{1.27}$$



(a) (b)

Figure 1.8: (a) Continuity of the tangential component of the magnetic field; (b) continuity of the orthogonal component of the magnetic flux density.

The continuity of the tangential component of $\overrightarrow{H}$ on the surface of the domain is imposed using the magneto-motive force as global variable. Integrating along a dual edge $\tilde{L}_k$ leads to:

$$h_k = \int\limits_{\tilde{L}_k} \overrightarrow{H} \cdot \overrightarrow{dL} = \int\limits_{\tilde{L}_k} \left(-\nabla\psi + \overrightarrow{H_S}\right) \cdot \overrightarrow{dL} = -(\psi_1 - \psi_2) + h_{Sk} \tag{1.28}$$

where 1 and 2 are the endpoints of the dual edge $\tilde{L}_k$ as shown in 1.8. In matrix form, 1.28 becomes $h = -C^T\psi + h_S$. C is primal face-to-edge incidence that corresponds to the surface discrete curl. Using the transpose operator, the dual edge-to-node incidence matrix (discrete dual gradient) is obtained [5, 23]. The continuity of the orthogonal component of the magnetic flux density in terms of global variables becomes the continuity of the magnetic flux through primal faces. Considering the kth triangular face:

$$b_k = \int_{S_k} \overrightarrow{B} \cdot \overrightarrow{dS} = \int_{S_k} \mu_0 \left(-\nabla\psi + \overrightarrow{H_S}\right) \cdot \overrightarrow{dS} = \mu_0(-\partial_n\psi + H_{Snk})S_k \tag{1.29}$$

Which becomes, by collecting the contribution of all triangles :

$$b = \mu_0 S(-\partial_n\psi + H_{Sn}) \tag{1.30}$$

being S a diagonal matrix with the area of the primal faces.

1.3.6 Final equation

Using the BEM equation 1.13, the topological equations $h = i$ and $Ce = -j\omega b$, the constitutive equation $i = Ye$ and the two continuity equations 1.28 and 1.30, the final system becomes :

$$\begin{pmatrix} -H & W \\ CZC^T & j\omega\mu_0 S \end{pmatrix} \begin{pmatrix} \psi \\ \partial_n\psi \end{pmatrix} = \begin{pmatrix} 0 \\ CZh_S + j\omega\mu_0 SH_{Sn} \end{pmatrix} \tag{1.31}$$

As a preliminary calculation of the impedance matrix $Z = Y^{-1}$ is needed, using a direct solver becomes impractical for large-scale problems. For this type of problems, Z is thus not fully computed but stored as an LU factorization instead, and iterative solvers such as GMRES [24] are used. See appendix E for more information on this type of solver.

Definition 1.3.1 (Schur complement)**.** Let $k \leq n \in \mathbb{N}$, $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ be an $n \times n$ block matrix with A of size k, and D invertible.
The Schur complement of the block D of the matrix M is the $(n-k) \times (n-k)$ matrix defined by

$$M/D := A - BD^{-1}C.$$

If A is invertible, the Schur complement of the block A of the matrix M is the $k \times k$ matrix defined by

$$M/A := D - CA^{-1}B.$$

Moreover, the spectral properties of this final equation can be further improved, as suggested by [25], by calculating the Schur complement of the lower-right bock, and solving with respect to $\partial_n\psi$:

$$\partial_n\psi = -H_{Sn} + (j\omega\mu_0 S)^{-1}(CZC^T\psi - CZh_S) \tag{1.32}$$

Then, by substitution :

$$\left(H + W(j\omega\mu_0 S)^{-1}CZC^T\right)\psi = W(j\omega\mu_0)^{-1}CZh_S + WH_{Sn} \tag{1.33}$$

As S is diagonal, calculating the Schur complement is relatively fast.

1.3.7 Nonlinear solution

In case of nonlinearity, the solution of equation 1.31 becomes iterative. In particular, the admittance matrix Y and its factorization Z depend on the working point of the magnetic characteristic. Thanks to the formulation of SIBC in terms of nonlinear admittance, the nonlinear term is integrated in a complex nonlinear factor F that can be mapped as a function of the tangential component of the magnetic field H_0 for each material at the specified frequency. The smoothness of the nonlinearities of F enables the use of a simple iterative scheme, where the admittance matrix Y is updated at each nonlinear iteration [1], as shown in 1 :

Algorithm 1: Nonlinear solution of SIBC-BEM [1]

```
$\varepsilon = \infty$, $\psi = 0$, $H_0 = 0$
$m'_{km} \leftarrow$ geometry
while $\varepsilon > \varepsilon_{max}$ and $k \leq k_{max}$ do
    $F_{km} \leftarrow F(H_0)$                                  (local nonlinear factor)
    $Y \leftarrow y'_{km} = F_{km} m'_{km}$                      (admittance matrix)
    $Z \leftarrow Y^{-1}$                                        (in LU form)
    $f \leftarrow W(j\omega\mu_0)^{-1} C Z h_S + W H_{Sn}$       (update right-hand side)
    $A \leftarrow H + W(j\omega\mu_0 S)^{-1} C Z C^T$            (update matrix)
    $\psi \leftarrow A\psi = f$                                  (GMRES solution)
    $\varepsilon = \|\psi - \psi^{(k-1)}\|_2 / \|\psi\|_2$       (residual error)
    $h \leftarrow C^T \psi + h_S$                                (equation 1.28)
    $H_0 = h/\tilde{L}$
end
return $\psi$
```

For the first step, the algorithm assumes a null tangential magnetic field, and the corresponding value of the parameter $F(H_0)$ corresponding to its value in the linear case. After computing the first solution, equation 1.28 gives rise to the vector of magneto-motive forces h, and the new vector of the tangential field is given by $H_{0,k} = h_k/\tilde{L}_k$.The iterations stop when the relative error in L_2-norm between two solutions is lower than a specified threshold or the number of iterations exceed the maximum value.

1.3.8 Preconditioner

In some cases, the problem may become ill-conditioned due to a poor mesh quality. This is caused by a high aspect ration of elements, usually when the geometry is characterized by one dimension much lower than the other, such as thin sheets [26]. A block-triangular preconditioner P of equation 1.31 is then used :

$$P = \begin{pmatrix} -\hat{H} & W \\ 0 & j\omega\mu_0 S \end{pmatrix} \tag{1.34}$$

where $\hat{H}$ is a suitable approximation of H. It can be proven that the corresponding preconditioner P_S for the Schur complement reduced system 1.33 is $P_S = \hat{H}$. A sparse approximation of the dense matrix $\hat{H}$ is proposed as preconditioner. It is built by recursively removing the small entries from matrix, but keeping the error in Frobenius norm $\|\cdot\|_F$ within a specified threshold ε : $\|H - \hat{H}\|_F \leq \varepsilon \|H\|_F$. $\hat{H}$ is then factorized using a sparse LU factorization and used as left preconditioner for equation 1.33.

1.4 State of the art

Some benchmarks in [1] were conducted on a cylindrical conductive disc on which currents were induced by a massive coil. However, benchmarks in this project were done on several stacked spheres of same size of both conductive and non-conductive magnetic materials of radius $r = 15$ mm, on which currents were induced by a coil of radius $r_c = 20$ mm, around the center of the system, as show in figure 1.9.

The coil is fed with a 85 kHz current, with a 100 A intensity. The relative magnetic permeability μ_R is set to 1000, while the electrical conductivity is $\sigma = 1 \cdot 10^6\,\mathrm{S\,m^{-1}}$. There were usually three spheres, but test were conducted on varying number of spheres and different material combinations as well.

Unless stated otherwise, the reference mesh for the benchmark is made of 816 nodes, 2,430 edges and 1,620 triangles. The hybrid scheme is compared to the 2d-axisymmetric finite element solution obtained using the freeware software FEMM (in dashed lines) in figure 1.11. Convergence rate of the method can be observed in figure 1.10.

Further comparisons with the FEMM solution in terms of of complexity, global and local accuracy are available on benchmarks on the cylindrical conductive disc in [1]. Because the numerical solution is based on the use of dense BEM matrices, both matrix filling and inversion times are critical. For this reason a suitable sparsification method is required to solve real world problems. Using a specific matrix structure, namely $\mathcal{H}$-matrices, would thus greatly improve matrix storage, while theoretically compute operations with an almost linear time complexity (for a $N \times N$ matrix, almost linear complexity is defined as a complexity of $\mathcal{O}(N \log^q N)$ with $q > 0$). The goal of this project is thus to implement a suitable $\mathcal{H}$-matrix library in MATLAB language, to solve matrix systems from BEM-SIBC methods in $\mathcal{H}$-matrix format, or at least use $\mathcal{H}$-matrix results as preconditioners.

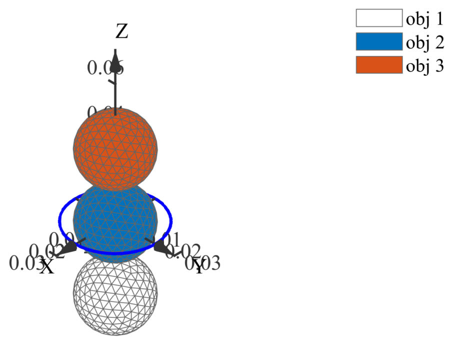


Figure 1.9: Example domain of study : obj 1, obj 2 and obj 3 are disconnected regions which can be made of the same or of different materials

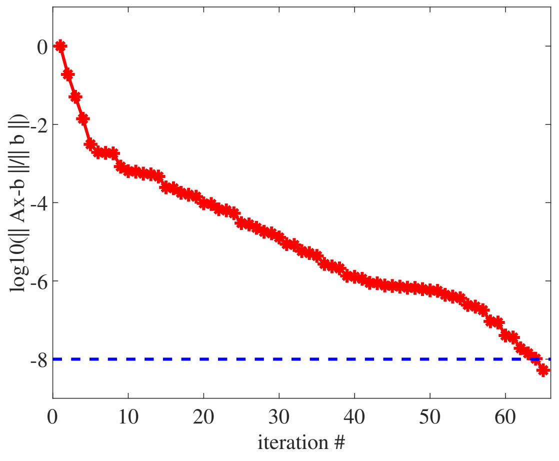


Figure 1.10: Convergence of GMRES solver on BEM-SIBC matrix system

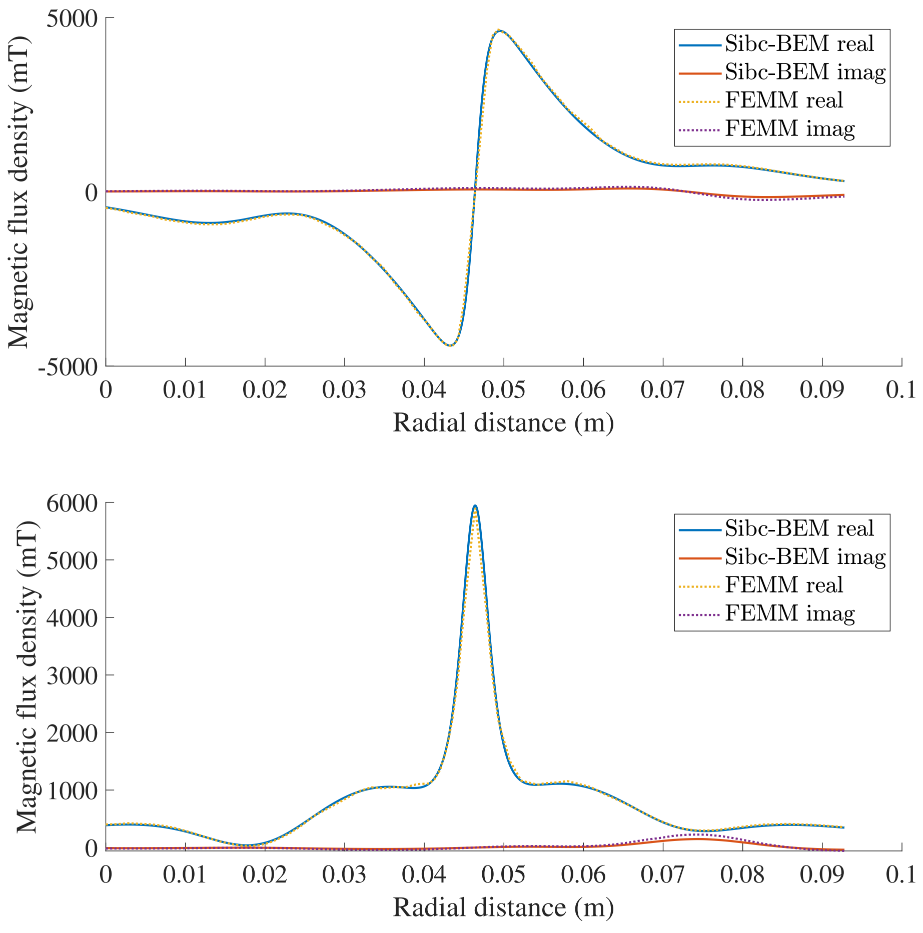


Figure 1.11: BEM-SIBC hybrid formulation and FEMM on the x and z components of the magnetic flux density

Chapter 2

$\mathcal{H}$-matrix theory

Let, for this chapter and all following chapters, unless specified otherwise, $\mathcal{I}, \mathcal{J}, \mathcal{K} \subset \mathbb{N}$, $N, M, n \in \mathbb{N}$, $t \times s \in \mathcal{P}(\mathcal{I} \times \mathcal{J})$ and $0 \leq k \leq N$.

2.1 Hierarchical matrices

The main idea behind $\mathcal{H}$-matrices is to reduce computational time and matrix storage by approximating a matrix $A \in \mathbb{C}^{N \times M}$ by low-rank blocks.

2.1.1 Reminders on Singular Value Decomposition (SVD)

Theorem 2.1.1 (Singular Value Decomposition : existence and uniqueness [27])**.** *Every matrix $A \in \mathbb{C}^{n \times m}$ has a singular value decomposition, defined as : $A = U\Sigma V^H$ where $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{n \times n}$ are unitary, $\Sigma \in \mathbb{R}^{n \times m}$ is diagonal, and its entries are non-negative and in non-increasing order.*
Furthermore, the singular values $\{\sigma_j\}_j$ are uniquely determined.

Proof. To prove the existence of the SVD, we isolate the direction of the largest action of A, and then proceed by induction on the dimension of A.
Set $\sigma_1 = \|A\|_2$. By a compactness argument, there must be a vector $v_1 \in \mathbb{C}^n$ with $\|v_1\|_2 = 1$ and $\|u_1\|_2 = \sigma_1$ where $u_1 = Av_1$. Consider any extensions of v_1 to an orthonormal basis $\{v_j\}$ of $\mathbb{C}^n$ and of u_1 to an orthonormal basis $\{u_j\}$ of $\mathbb{C}^m$, and let U_1 and V_1 denote the unitary matrices with columns u_j and v_j, respectively. Then we have :

$$U_1^H A V_1 = S = \begin{pmatrix} \sigma_1 & \omega^H \\ 0 & B \end{pmatrix},$$

where 0 is a column vector of dimension $m-1$, ω^H is a row vector of dimension $n-1$, and B has dimensions $(m-1) \times (n-1)$. Furthermore,

$$\left\| \begin{pmatrix} \sigma_1 & \omega^H \\ 0 & B \end{pmatrix} \begin{pmatrix} \sigma_1 \\ \omega \end{pmatrix} \right\| \geq \sigma_1^2 + \omega\omega^H = (\sigma_1^2 + \omega\omega^H)^{1/2} \left\| \begin{pmatrix} \sigma_1 \\ \omega \end{pmatrix} \right\|,$$

implying $\|S\|_2 \geq (\sigma_1^2 + \omega\omega^H)^{1/2}$. Since U_1 and V_a are unitary, $\|S\|_2 = \|A\|_2 = \sigma_1$, so this implies $\omega = 0$. If $n = 1$ or $m = 1$, we are done. Otherwise, the submatrix B describes the action of A on the subspace orthogonal to v_1. By the induction hypothesis, B has an SVD : $B = U_2 \Sigma_2 V_2^H$. Now it is easily verified that

$$A = U_1 \begin{pmatrix} 1 & 0 \\ 0 & U_2 \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \Sigma_2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & V_2 \end{pmatrix}^H V_1^H$$

is an SVD of A, completing the proof of existence.

For the uniqueness claim, algebraically, we can argue as follows. First we note that σ_1 is uniquely determined by the condition that it is equal to $\|A\|_2$, as follows from the definition. Now suppose that in addition to v_1, there is another linearly independent vector ω with $\|\omega\|_2 = 1$ and $\|A\omega\|_2 = \sigma_1$. Define a unit vector v_2, orthogonal to v_1, as a linear combination of v_1 and ω,

$$v_2 = \frac{\omega - (v_1^H \omega) v_1}{\|\omega - (v_1^H \omega) v_1\|_2}.$$

Since $\|A\|_2 = \sigma_1$, $|Av_2\|_2 \leq \sigma_1$;but this must be an equality, for otherwise, since $\omega = v_1c + v_2s$ for some constants c and s with $|c|^2 + |s|^2 = 1$, we would have $\|A\omega\|_2 < \sigma_1$. This vector v_2 is a second right singular vector of A corresponding to the singular value σ_1; it will lead to the appearance of a vector y (equal to the last $n-1$ components of $V_1^H v_2$) with $\|y\|_2 = 1$ and $\|By\|_2 = \sigma_1$. We conclude that, if the singular vector v_1 is not unique, then the corresponding singular value σ_1 is not simple. To complete the uniqueness proof we note that, as indicated above, once σ_1, v_1, and u_1 are determined, the remainder of the SVD is determined by the action of A on the space orthogonal to v_1. Since v_1 is unique up to sign, this orthogonal space is uniquely defined, and the uniqueness of the remaining singular values and vectors now follows by induction. □

Remark With this proof from [27], an additional point is also proven for square matrices : if A is square and the $\{\sigma_j\}$ are distinct, the left and right singular vectors $\{u_j\}$ and $\{v_j\}$ are uniquely determined up to complex signs (i.e., complex scalar factors of absolute value 1).

Theorem 2.1.2 (Eckart-Young-Mirsky (partially admitted))**.** *Let $k \geq 0$, $M \in \mathbb{C}^{n\times m}$ be a matrix and $M = U\Sigma V^H$ its SVD. Then, for any rank-k matrix A, we have :*

$$\|M - A\|_2 \geq \|M - U_k\Sigma_k V_k^H\|_2 = \sigma_{k+1}$$

$$\|M - A\|_F \geq \|M - U_k\Sigma_k V_k^H\|_F = \sqrt{\sum_{i=k+1}^{m} \sigma_i^2}$$

With V_k, Σ_k, U_k being subblocks of respectively V, Σ, U containing the first k rows and columns of the orignial matrix.

Proof. This theorem will only be proven for the euclidean norm here.
Since $rank(A) \leq k$, $dim(Ker(A)) \geq n-k$. It follows that $dim(Ker(A)) + dim(Im(V_{k+1}) \geq n-k+k+1 = n+1$, thus there exists $x \in Ker(A) \cap Im(V_{k+1})$. Then $\|(M-A)x\|_2^2 = \sum_{i=1}^{k+1} \sigma_i^2 < x, v_i >^2 \geq \sigma_{k+1}^2 \sum_{i=1}^{k+1} < x, v_i >^2 = \sigma_{k+1}^2$. □

2.1.2 A quick introduction to ACA method and $\mathcal{H}$-matrices

Given a matrix $A \in \mathbb{C}^{N\times M}$ and a precision ϵ, the best approximation, according to section 2.1.1, $\tilde{A}$ of A such that

$$\|A - \tilde{A}\|_2^2 \leq \epsilon^2 \tag{2.1}$$

would be a partial singular value decomposition (SVD) of A :

$$A \simeq \tilde{A} = \tilde{A}(r) = \sum_{i=1}^{r} \sigma_i u_i v_i^\top$$

where $\sigma_i \in \mathbb{R}_+$, $u_i \in \mathbb{C}^N$, $v_i \in \mathbb{C}^M$, $i = 1, ..., r$ and the rank $r = r(\epsilon)$ chosen such that (2.1) is verified.

However, computing the SVD is far too expensive (assuming that $N \sim M$, time complexity would be of $\mathcal{O}(N^3)$). That is where the idea of separating A into blocks stems from : each sub-block is smaller and easier to approximate if needed.

2.1.3 Low rank approximation of matrix blocks

For $k \in \mathbb{N}$, let $\mathbb{C}_k^{N\times M} = \{A \in \mathbb{C}^{N\times M} | rank(A) \leq k\}$.

Theorem 2.1.3 (Outer-product form)**.** *A matrix $A \in \mathbb{C}^{N\times M}$ belongs to $\mathbb{C}_k^{N\times M}$ if and only if there are $U \in \mathbb{C}^{N\times k}$ and $V \in \mathbb{C}^{M\times k}$ such that*

$$A = UV^H$$

This representation is called outer-product form.

Proof. Let $A \in \mathbb{C}^{N\times M}$ be a matrix. Let us first suppose that there are $U \in \mathbb{C}^{N\times k}$ and $V \in \mathbb{C}^{M\times k}$ such that $A = UV^H$. Then, by using some basic rank properties, it follows that

$$\begin{aligned} rank(A) = rank(UV^H) &\leq \min(rank(U), rank(V^H)) \\ &\leq \min(\min(N,k), \min(M,k)) \leq k \end{aligned} \tag{2.2}$$

Thus $A \in \mathbb{C}_k^{N\times M}$.

Let us assume now that $A \in \mathbb{C}_k^{N\times M}$. One version of this proof can be written using linearly independent column vectors to represent A, which is actually quite similar to the path that is taken here : to make another good use of section 2.1.2, this proof will use the theorems previously proven. According to theorem 2.1.1, there exists $U \in \mathbb{C}^{m\times m}$ and $V \in \mathbb{C}^{n\times n}$ unitary, $\Sigma \in \mathbb{R}^{n\times m}$ diagonal, and its entries are nonnegative and in nonincreasing order such that $A = U\Sigma V^H$. Then, since A is of rank $\leq k$, it has maximum k nonzero singular values which are stored in Σ as $\sigma_1, ..., \sigma_k$. Thus, $\|A - U_k\Sigma_k V_k^T\|_2 = \sigma_{k+1} = 0$. This entails that $A - U_k\Sigma_k V_k^T = 0$ using the properties of $\|\cdot\|_2$. Thus, if $\tilde{U} = U_k\Sigma_k \in \mathbb{C}^{N\times k}$ and $\tilde{V} = V_k \in \mathbb{C}^{M\times k}$, $A = \tilde{U}\tilde{V}^H$. □

Definition 2.1.1. A matrix $A \in \mathbb{C}_k^{N\times M}$ is called a matrix of low rank if

$$k(N+M) < NM$$

Low-rank matrices will always be represented in outer-product form [28].

2.1.4 Singular Value Decomposition of low-rank matrices

For a low-rank matrix $A = UV^H \in \mathbb{C}_k^{N\times M}$, a method using QR decompositions was introduced in [29] and enables an efficient and not-to-expensive computation of the SVD. Assume the QR decompositions $U = Q_U R_U$ of $U \in \mathbb{C}^{N\times k}$ and $V = Q_V R_V$ of $V \in \mathbb{C}^{N\times k}$ computed. The outer product $R_U R_V^H$ of the two $k\times k$ upper triangular matrices R_U and R_V is then decomposed using the SVD

$$R_U R_V^H = \hat{U}\hat{\Sigma}\hat{V}^H.$$

Since $Q_U\hat{U}$ and $Q_V\hat{V}$ both have orthonormal columns,

$$A = UV^H = (Q_U\hat{U})\hat{\Sigma}(Q_V\hat{V})^H$$

is an SVD of A.

2.1.5 Block cluster tree and partitioning

Let $\mathcal{I} = \{1, ..., N\}$ and $\mathcal{J} = \{1, ..., M\}$. Typically, A can be approximated by low-rank matrices only on certain sub-blocks of an appropriate partition of $\mathcal{I}\times\mathcal{J}$ [30].

Finding a suitable partition $P \subset \mathcal{P}(\mathcal{I}\times\mathcal{J})$ implies two contrary conditions : having a partition fine enough so that block can be successfully approximated and having as few blocks as possible. A good partition should allow approximants of logarithmic-linear complexity and be computed with almost linear complexity.

The admissibility condition is a set of conditions a block $b \in P$ should meet in order to be approximated by a matrix of low-rank :

- if b is admissible, then the singular values of A_b decay exponentially;
- the admissibility condition can be checked for each block $t\times s \in \mathcal{P}(\mathcal{I}\times\mathcal{J})$ with a complexity of $\mathcal{O}(|t|+|s|)$;
- if b is admissible, then any subset of b is, too.

Definition 2.1.2. Let $n_{min} \in \mathbb{N}$. A partition $\mathcal{P}$ is called admissible if each block $t\times s \in P$ is either small or admissible. It is called small if $min\{|t|, |s|\} \leq n_{min}$.

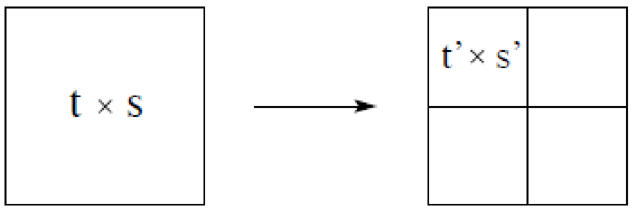


Figure 2.1: Block division

The candidates $(t,s) \subset \mathcal{I}\times\mathcal{J}$ for a suitable partition of $\mathcal{I}\times\mathcal{J}$ will be stored in cluster trees $\mathcal{T}_\mathcal{I}$ and $\mathcal{T}_\mathcal{J}$. Assuming $n = 2^p$ with $p \in \mathbb{N}$ and $n_{min} \in \mathbb{N}$ is the minimum block size for this example, the root of $\mathcal{T}_\mathcal{I}$ is the index set $\mathcal{I}_1^{(0)} = \{1, ..., n\}$. $I_1^{(0)}$ has two successors $I_1^{(1)} = \{1, .., \frac{n}{2}\}$ and $I_2^{(1)} = \{\frac{n}{2}+1, .., n\}$. Each node t with more than n_{min} indices has exactly two successors; each one containing respectively the first and second half of its indices. If t does not have more than n_{min} indices, then it is a node; n_{min} thus controls the depth of the cluster tree. We denote as $S(t)$ the sons of a node t [31].

To find an admissible partition of $\mathcal{I} \times \mathcal{J}$, a so-called block cluster tree $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ is used. To restrict the number of possible partitions, only the ones which are induced from subdividing rows and columns of the original matrix are considered. These $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ are defined by the following mapping $S_{\mathcal{I}\times\mathcal{J}}$:

$$S_{\mathcal{I}\times\mathcal{J}} = \begin{cases} \emptyset & \text{if } t \times s \text{ is admissible or } S_{\mathcal{I}} = \emptyset \text{ or } S_{\mathcal{J}} = \emptyset \\ S_{\mathcal{I}}(t) \times S_{\mathcal{J}}(s) & \text{else} \end{cases} \tag{2.3}$$

The leaves of $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ thus constructed are an admissible partition.

2.1.6 The set of Hierarchical Matrices

Definition 2.1.3. Let $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ be a block cluster tree for the index sets $\mathcal{I} \times \mathcal{J}$. the set of $\mathcal{H}$-matrices is defined as :

$$\mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k) := \{A \in \mathbb{C}^{\mathcal{I}\times\mathcal{J}} \mid rank(A|_{t\times s}) \leq k \text{ for all } t \times s \text{ admissible leaves of } \mathcal{T}_{\mathcal{I}\times\mathcal{J}}\}.$$

Elements from $\mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$ will often be called $\mathcal{H}$-matrices.

For an optimal matrix storage and treatment, the outer-product form from 2.1.3 should be used for admissible blocks. Some easy consequences follow :

Lemma 2.1.4. *Let $A \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$. Then*

1. *any submatrix A_b, $b \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$, belongs to $\mathcal{H}(\mathcal{T}_b, k)$;*
2. *the transpose A^T and the Hermitian transpose A^H belong to $\mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$, provided that the admissibility condition is symmetric; i.e., any bloc $t \times s$ is admissible if $t \times s$ is admissible.*

For two index sets $\mathcal{I}$ and $\mathcal{J}$ and P a partition of $\mathcal{I} \times \mathcal{J}$, let

$$\pi_{\mathcal{I}} = \{t \in \mathcal{T}_{\mathcal{I}} \mid \exists s \in \mathcal{T}_{\mathcal{J}}, t \times s \in P,\ \forall t' \subset t,\ \forall s' \in \mathcal{T}_{\mathcal{J}} : t' \times s' \notin P\}$$

$\pi_{\mathcal{I}}$ is thus the finest partition of $\mathcal{I}$ made from clusters appearing in the partition P.

2.2 Cross approximation of admissible $\mathcal{H}$-matrix blocks

In this section, one of the main methods we used to approximate an admissible matrix block up to a given error ϵ will be explored. The other one employed in this project was Singular Value Decomposition (SVD) as explained in 2.1.2, which is far less computationally expensive if used only on admissible blocks rather than on the whole matrix [28, 31].

Let $\epsilon > 0$ be a given error and $k \in \mathbb{N}$ a maximum rank for this section.

2.2.1 Cross approximation

Let $t \times s \subset \mathcal{I} \times \mathcal{J}$, and $A \in \mathbb{C}^{t\times s}$ be a matrix block. An idea presented in [32] under the name *skeleton approximation* is to choose small subsets $\tilde{s} \subset s$ and $\tilde{t} \subset t$ of pivot columns and rows so that for a matrix $G \in \mathbb{C}^{\tilde{t}\times\tilde{s}}$, there holds:

$$\|A - \tilde{A}\|_2 \leq \epsilon$$
$$\tilde{A} = A|_{t\times\tilde{s}} G A|_{\tilde{t}\times s},\ rank(\tilde{A}) \leq min(\#\tilde{s}, \#\tilde{t})$$

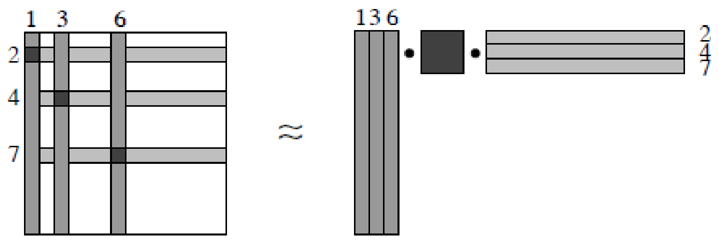


Figure 2.2: Cross approximation of rank 3

Theorem 2.2.1 (Existence of cross-approximations)**.** *Let $A, R \in \mathbb{C}^{t\times s}$ be matrices with $\|A - R\| \leq \epsilon$ and $rank(R) \leq k$. Then there exists a subset $\tilde{s} \subset s$ of pivot columns and a subset $\tilde{t} \subset t$ of pivot rows ans a matrix $G \in \mathbb{C}^{\tilde{t}\times\tilde{s}}$ with*

$$\|A - A|_{t\times\tilde{s}}GA|_{\tilde{t}\times s}\|_2 \leq \epsilon(1 + 2\sqrt{k}(\sqrt{\#t} + \sqrt{\#s})).$$

$A|_{t\times\tilde{s}}GA|_{\tilde{t}\times s}$ is called a pseudoskeleton component.

Proof. Let $A, R \in \mathbb{C}^{t\times s}$ be matrices with $\|A - R\| \leq \epsilon$ and $rank(R) \leq k$. The theorem will first be proven for a $\tilde{t}, \tilde{s}$ and G which verify the inequality

$$\|A - A|_{t\times\tilde{s}}GA|_{\tilde{t}\times s}\|_2 \leq \epsilon(1 + (\sqrt{e(k, \#t)} + \sqrt{e(k, \#s)}))^2),$$

with

$$e(k, \#t) = \frac{1}{\underset{B\in\mathbb{C}^{\#t\times k}, BB^H=1}{\min} \ \underset{P \text{ submatrix of } B}{\max} \sigma_{min}(P)}$$

where $\sigma_{min}(P)$ denotes the minimal singular value of P. Estimates for $e(k, \#t)$ and $e(k, \#s)$ will then complete the proof.
Consider the decomposition $R = U\Sigma V$ the SVD of R defined like in theorem 2.1.1. Let $\hat{U}, \hat{V} \in \mathbb{R}^{k\times k}$ be submatrices of U, V respectively such that

$$\|\hat{U}^{-1}\|_2 \leq e(k, \#t) \tag{2.4}$$

$$\|\hat{V}^{-1}\|_2 \leq e(k, \#s) \tag{2.5}$$

We know select k rows and k columns, denoted as $\tilde{t} \subset t$ and $\tilde{s} \subset s$, determined by the choice of $\hat{U}$ and $\hat{V}$, respectively. Let $F = A - R$, $C = A|_{t\times\tilde{s}}$ and F_c denote $t \times \tilde{s}$ submatrices, $M = A|_{\tilde{t}\times s}$ and F_M denote $\tilde{t} \times s$ submatrices of A and F respectively, which correspond to the selected rows and columns. Let $\hat{A}$ and $\hat{F}$ denote the $\tilde{t} \times \tilde{s}$ submatrices which occupy the intersections of these rows and columns in A and F. Then any pseudoskeleton component CGM, $G \in \mathbb{C}^{\tilde{t}\times\tilde{s}}$, can be presented in the following form :

$$CGM = (U\Sigma\hat{V} + F_C)G(\hat{U}\Sigma V + F_M) = U\Sigma\hat{V}G\hat{U}\Sigma V + E \tag{2.6}$$

with

$$\begin{aligned} E &= (U\Sigma\hat{V} + F_C)GF_M + F_CG(\hat{U}\Sigma V + F_M) - F_cGF_M \\ &= U\hat{U}^{-1}(\hat{U}\Sigma\hat{V}G)F_M + F_C(G\hat{U}\Sigma\hat{V})\hat{V}^{-1}V + F_CGF_M \\ &= U\hat{U}^{-1}(\tilde{R}G)F_M + F_C(G\tilde{R})\hat{V}^{-1}V + F_CGF_M \end{aligned} \tag{2.7}$$

where $\tilde{R} = \hat{U}\Sigma\hat{V} = \hat{A} - \hat{F}$. The first equations can also be rewritten in terms of $\tilde{R}$:

$$R = U\hat{U}^{-1}\tilde{R}\hat{V}^{-1}V$$

$$CGM = U\hat{U}^{-1}(\tilde{R}G\tilde{R})\hat{V}^{-1}V + E$$

Consider now the SVD of $\tilde{R}$: $\tilde{R} = \tilde{U}\tilde{\Sigma}\tilde{V}$ with $\tilde{\Sigma} = (diag)(\tilde{\sigma_1}, ..., \tilde{\sigma_r})$, $\tilde{U}$ and $\tilde{V}$ unitary. Let $\tau > 0$ be a threshold which will be specified later. Introducing the notation :

$$\tilde{\Sigma}_\tau = \text{diag}(\tilde{\sigma}_{\tau i}), \quad \tilde{\sigma}_{\tau i} = \begin{cases} \tilde{\sigma} & \text{if } \tilde{\sigma} \geq \tau, \\ 0 & \text{otherwise,} \end{cases} \tag{2.8}$$

$$\tilde{\Sigma}^+_\tau = \text{diag}(\tilde{\sigma}^+_{\tau i}), \quad \tilde{\sigma}^+_{\tau i} = \begin{cases} \tilde{\sigma}_i^{-1} & \text{if } \tilde{\sigma}_i \geq \tau, \\ 0 & \text{otherwise,} \end{cases} \tag{2.9}$$

$$\tilde{R}_\tau = \tilde{U}\tilde{\Sigma}_\tau\tilde{V}, \tag{2.10}$$

$$\tilde{R}^+_\tau = \tilde{V}^H\tilde{\Sigma}^+_\tau\tilde{U}^H, \tag{2.11}$$

One can see that $\tilde{R}\tilde{R}^+_\tau\tilde{R} = \tilde{R}_\tau$, and moreover, this implies that $\|\tilde{R}\tilde{R}^+_\tau\|_2 \leq 1$, $\|\tilde{R}^+_\tau\tilde{R}\|_2 \leq 1$. If we set $G = \tilde{R}^+_\tau$ then the three previous relations imply that $\|E\|_2 \leq \epsilon(\|\tilde{U}^{-1}\|_2 + \|\tilde{V}^{-1}\|_2 + \frac{\epsilon}{\tau})$. Using this inequality in conjunction with R and CGM rewritten in terms of $\tilde{R}$, and $\tilde{R}\tilde{R}^+_\tau\tilde{R} = \tilde{R}_\tau$, one can get the estimate :

$$\|A - CGM\|_2 \leq \epsilon + \tau\|\tilde{U}^{-1}\|_2\|\tilde{V}^{-1}\|_2 + \frac{\epsilon^2}{\tau} + \epsilon\|\tilde{U}^{-1}\|_2 + \epsilon\|\tilde{V}^{-1}\|_2.$$

Setting $\tau = \frac{\epsilon}{\sqrt{\|\tilde{U}^{-1}\|_2\|\tilde{V}^{-1}\|_2}}$ completes the first part of the proof.

A technical lemma will help getting the right estimate for the second part (proof in appendix B):

Lemma 2.2.2. *For $k, \in \mathbb{N}, k \leq n$, $e(k,n)$ defined above satisfies $e(k,n) \leq \sqrt{k(n-k)+1}$.*

Thus, using lemma B.0.1 and the first part of the proof, it follows that :

$$\begin{aligned}
\|A - A|_{t\times\tilde{s}} G A|_{\tilde{t}\times s}\|_2 &\leq \epsilon(1 + (\sqrt{e(k,\#t)} + \sqrt{e(k,\#s)})^2) \\
&\leq \epsilon(1 + 2(e(k,\#t) + e(k,\#s)) \\
&\leq \epsilon(1 + (2(\sqrt{k(\#t-k)+1} + \sqrt{k(\#s-k)+1})) \\
&\leq \epsilon(1 + (2\sqrt{k}(\sqrt{\#t - k + 1/k} + \sqrt{\#s - k + 1/k})) \\
&\leq \epsilon(1 + (2\sqrt{k}(\sqrt{\#t} + \sqrt{\#s}))
\end{aligned}$$

□

Remark To match with the estimations given by the SVD, similar results can also be proved using the Frobenius norm, see [28], or just by using norm equivalents ($\forall n \in \mathbb{N}$, $\forall A \in \mathbb{C}^{n\times n}$, $\|A\|_2 \leq \|A\|_F \leq \sqrt{n}\|A\|_2$).

In order to fully explain the adaptive cross approximation algorithm, fully- and partially pivoted cross approximations will first be examined. They both successfully compute rank one approximations.

2.2.2 Cross Approximation with full pivoting

Let $A \in \mathbb{C}^{t\times s}$ be given. A rank one-approximation of A using full pivoting can be obtained in two steps, provided that all entries of A have been previously computed :

1. Find $(i,j) \in t \times s$ such that $|A_{i,j}|$ is the maximal entry of A. Set $\delta = A_{i,j}$;
2. Compute $a_k = A_{k,j},\ k \in t, \quad b_l = \frac{A_{i,l}}{\delta},\ l \in s$

Then $R_1 = ab^H$ is a matrix of rank one meeting the conditions of theorem 2.2.1. If $k > 1$, and rank one to $k-1$ approximations of A, $a^1(b^1)^H, ..., a^{k-1}(b^{k-1})^H$ have already been computed,then a rank k approximation R_k can be obtained by applying the above steps to $A - \sum_{N=1}^{k-1} a^N(b^N)^H$.

This process can be summed up in the following algorithm, which takes A ans k as an input and overwrites it to give $R_k = \sum_{N=1}^{k} a^N(b^N)^H$, the rank k approximation of A, as an output :

Algorithm 2: Cross approximation with full pivoting

```
for $N = 1, ..., k$ do
    Compute the maximal entry in modulus
        $(i_N, j_N) = \underset{(i,j)}{\operatorname{argmax}} |A_{i,j}|,\ \delta = A_{i_N,j_N}$
    if $\delta = 0$ then
        the algorithm terminates with the exact rank $N-1$ representation $R_{N-1}$ of the input matrix $A$.
    else
        Compute the entries of the vectors $a^N, b^N$ : $(a^N)_i = A_{i,j_N},\ i \in t, \quad (b^N)_j = A_{i_N,j}/\delta,\ j \in s$.
        Subtract the rank one approximation $Ai, j = A_{i,j} - (a^N)_i(b^N)_j,\ i \in t, j \in s$
    end
end
```

Lemma 2.2.3 (Exact reproduction of rank k matrices)**.** *Let $n, m \in \mathbb{N}^*$. Let $A \in \mathbb{C}^{n\times m}$ be a matrix of rank exactly k. Then $R_k = \sum_{N=1}^{k} a^N(b^N)^H$ from algorithm 2 equals A. Proof in appendix B.*

Lemma 2.2.4 (Interpolation property)**.** *Let $A \in \mathbb{C}^{t\times s}$ be a matrix of rank at least $k \geq 1$ and R_k the cross approximation from algorithm 2. Then, for any row pivot index i^* and any column pivot index j^* there holds :*

$$R_k e_{j*} = A_{t\times\{j^*\}} \quad \textit{and} \quad e_{i*}^H R_k = A_{\{i^*\}\times s}$$

i.e., R_k exactly reproduces the pivot columns and rows of A. Proof in appendix B.

Lemma 2.2.5. *The cross approximation of a matrix $A \in \mathbb{C}^{t\times s}$ of rank at least $k \geq 0$ by the matrix R_k from 2 is of the form*

$$R_k := \sum_{N=1}^{k} a^N (b^N)^H = A|_{t\times\mathcal{P}_{cols}} (A|_{\mathcal{P}_{rows}\times\mathcal{P}_{cols}})^{-1} A|_{\mathcal{P}_{rows}\times s}$$

Proof in appendix B.

Unfortunately, the determination of pivot indices in this algorithm makes it of quadratic complexity $\mathcal{O}(k\#t\#s)$ for a matrix in $\mathbb{C}^{t\times s}$ of rank k, which does not make it efficient enough for large scale problems.

2.2.3 Cross approximation with partial pivoting

In algorithm 2, determining the pivot pairs and updating the matrix are the bottleneck of the process. The first step in optimizing the previous algorithm is thus to change the determination of the pivot pair, so that computing all the entries of the matrix A is not needed. A is also no longer updated inside the agorithm, and a temporary representation of the residual is used instead.

The main idea behind the new pivot choice is partial pivoting : maximizing $|A_{i,j}|$ only for one of the two indices i or j and keeping the other one fixed. In other words, determining the maximal element in modulus over one row or column.

Algorithm 3: Cross approximation with partial pivoting

```
Set $i^* = min\{i \in t\}, N = 1, n = \#t$ and $\mathcal{P} = \{\}$ the set of row pivot indices. while $N \leq k$ do
    Compute the maximal entry in modulus
        $$j^* = \underset{j\in s}{\text{argmax}} |A_{i^*,j}|,\ \delta = A_{i^*,j^*} - \sum_{\nu=1}^{N-1} (a^\nu)_{i^*} (b^\nu)_{j^*}$$
    if $\delta = 0$ then
        if $\#\mathcal{P} =$ then
            the algorithm terminates with the exact rank $N-1$ representation $R_{N-1}$ of the input matrix $A$.
        end
    else
        Compute the entries of the vectors $a^N, b^N$ : $(a^N)_i = A_{i,j^*} - \sum_{\nu=1}^{N-1} (a^\nu)_i (b^\nu)_{j^*},\ i \in t,$
        $(b^N)_j = (A_{i^*,j} - \sum_{\nu=1}^{N-1} (a^\nu)_{i^*} (b^\nu)_j)/\delta,\ j \in s.$
    end
        $$\mathcal{P} = \mathcal{P} \cup \{i^*\}$$
    Choose $i^* \in t \setminus \mathcal{P}$, e.g., $i^* = \text{argmax}_{i\in t\setminus\mathcal{P}} |(b^N)_{i,j^*}|$
end
```

For $\delta \neq 0$ the arguments of lemma B.0.2 apply and the rank of the remainder is reduces by one. However, this condition $\delta \neq 0$ can also be a bottleneck. This technique is thus not suited for sparse matrices, but can be efficient on dense BEM ones.

2.2.4 Adaptive Cross Approximation (ACA)

Given an accuracy ϵ, another variant of the previous algorithms would be to determine a rank k such that the approximation is at a distance ϵ of the original matrix A using a chosen norm. The rank is determined adaptively, hence the name adaptive cross approximation. Just like before, the chosen norm here will be $\|\cdot\|_2$ and all results can easily be adapted to $\|\cdot\|_F$ to compare to the approximation of a matrix by the SVD.

A good heuristic is to estimate the remainder $\|A - R_k\|_2$ by a rank one approximation R. Since, in the algorithm, a successive rank one approximation is used, one can estimate :

$$\|M - R_k\|_2 \lesssim \|M - R_{k-1}\|_2 \approx \|R_k - R_{k-1}\|_2 = \|a^k (b^k)^T\|_2$$

With a^k and b^k constructed by in algorithm 3, let :

$$\varepsilon_{abs}(k) = \|a^k\|_2 \|b^k\|_2, \quad \varepsilon_{rel}(k) = \|a^k\|_2 \|b^k\|_2 / \|a^1\|_2 \|b^1\|_2.$$

For $k = 0$, $\varepsilon_{rel}(0) = \varepsilon_{abs}(0) = \infty$. These functionals can be used to estimate the (absolute or relative) error $\|A - R_k\|_2$
The stopping criterion $N \leq k$ is then replaced by either $\varepsilon_{rel}(N-1) \leq \epsilon$ or $\varepsilon_{abs}(N-1) \leq \epsilon$.

If the matrix A stems from the evaluation of a smooth function at some points in $\mathbb{R}^d, d \geq 1$, then [33], [34] prove that this adaptive cross approximation algorithm converges. We will not cover this proof in detail in this report, as the matrices used here stem from basic smooth functions, such as $f : x \mapsto \frac{1}{x+\varepsilon}, \varepsilon > 0$ on $\mathbb{R}_+$ for the simplest examples. In this version, a maximum rank for the approximation was also added, to ensure that the algorithm would not be too computationally expensive, in case the ACA method had convergence issues.

Here is the algorithm, from [28], that was implemented in `Matlab` for this project :

Algorithm 4: Adptive Cross approximation with partial pivoting

```
Inputs : $f$ kernel function from which stems $A$ ,$t$,$s$,$\eta$ precision ,$k_{Max}$ maximum rank allowed
Outputs : $a^k$, $b^k$ approximation vectors, $k$ rank of the approximation, $\varepsilon$ corresponding approximation
  error
Set $i^* = min\{i \in t\}, N = 1, n = \#t$ and $\mathcal{P} = \{\}$ the set of row pivot indices, $k = 0$, $\varepsilon = \infty$.
while $\varepsilon > \eta$ and $k < k_{Max}$ do
    Compute the maximal entry in modulus
        $$j^* = \underset{j \in s}{\operatorname{argmax}} |A_{i^*,j}|, \ \delta = A_{i^*,j^*} - \sum_{\nu=1}^{N-1} (a^\nu)_{i^*} (b^\nu)_{j^*}$$
    if $\delta = 0$ then
        if $\#\mathcal{P} =$ then
            the algorithm terminates with the exact rank $N-1$ representation $R_{N-1}$ of the input matrix
              $A$.
        end
    else
        Compute the entries of the vectors $a^N, b^N$ : $(a^N)_i = A_{i,j^*} - \sum_{\nu=1}^{N-1} (a^\nu)_i (b^\nu)_{j^*}$, $i \in t$,
          $(b^N)_j = (A_{i^*,j} - \sum_{\nu=1}^{N-1} (a^\nu)_{i^*} (b^\nu)_j)/\delta$, $j \in s$.
    end
    Update rank $k = k + 1$.
    Update $\varepsilon$ according to stopping criterion above.
        $$\mathcal{P} = \mathcal{P} \cup \{i^*\}$$
    Choose $i^* \in t \setminus \mathcal{P}$, e.g., $i^* = \operatorname{argmax}_{i \in t \setminus \mathcal{P}} |(b^N)_{i,j^*}|$
end
```

2.3 $\mathcal{H}$-matrix algebra

2.3.1 Rounded addition

The sum of two matrices $A, B \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$ will usually be in $\mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, 2k)$ but not in $\mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$. The reason for this is that $\mathbb{C}_k^{N\times M}$ is not a linear space : if two matrices of rank lower or equal to k are added, the rank of their sum is only lower or equal to $2k$. That is why a truncated (or rounded) addition is used.

Definition 2.3.1 (Truncation T_k and T_ε)**.** Let $A \in \mathbb{C}^{N\times M}$ be a matrix, and $\|\cdot\|$ a matrix norm over $\mathbb{C}^{N\times M}$.

1. Let $k \in \mathbb{N}$. One can define the truncation operator $T_k : \mathbb{C}^{N\times M} \to \mathbb{C}_k^{N\times M}, A \mapsto \tilde{A}$ where $\tilde{A}$ is a best approximation of A in the set $\mathbb{C}_k^{N\times M}$ (not necessarily unique) ;

2. Let $\varepsilon > 0$. One can define the truncation operator $T_\varepsilon : \mathbb{C}^{N\times M} \to \mathbb{C}_k^{N\times M}, A \mapsto \hat{A}$ where $\hat{A}$ is a best approximation of A in the set $\mathbb{C}_k^{N\times M}$ and $k = \min\{\hat{k} \in \mathbb{N}^* | \exists \hat{A} \in \mathbb{C}_{\hat{k}}^{N\times M}, \|A - \hat{A}\| \leq \varepsilon\|A\|\}$.

$\tilde{A}$ is called a truncation of rank k, and $\hat{A}$ an adaptive truncation of A with accuracy ε.T_ε^{abs} is the respective truncation operator with absolute truncation error ε.

Definition 2.3.2 (Extension of the truncation $\mathcal{T}_k$ for $\mathcal{H}$-matrices)**.** Let $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ be a block cluster tree. The truncation operator is defined as :

$$T_k : \mathbb{C}^{N\times M} \to \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k), \ A \mapsto \tilde{A}$$

blockwise for all leaves $t \times s \in \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ by

$$\tilde{A}|_{t\times s} = \begin{cases} T_k(\tilde{A}|_{t\times s}) & \text{if } t \times s \text{ is admissible} \\ \tilde{A}|_{t\times s} & \text{otherwise.} \end{cases}$$

Lemma 2.3.1. *The operator T_k maps a matrix $A \in \mathbb{C}^{\mathcal{I}\times\mathcal{J}}$ to a best approximation $\tilde{A} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$ with respect to the Frobenius norm :*

$$\left\|A - \tilde{A}\right\|_F = \min_{M' \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}},k)} \|A - A'\|_F$$

Proof in appendix B.

Definition 2.3.3 (Formatted addition)**.** Let $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ denote a block cluster tree and $k \in \mathbb{N}$. The formatted addition of $\mathcal{H}$-matrices $A, B \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$ is defined by:

$$A \oplus B = T_k(A + B).$$

If the rank k in consideration is not evident then it can be written $\oplus_k$ instead of $\oplus$.

2.3.2 Agglomerating low-rank blocks

For memory-saving purposes, neighboring low-rank blocks can be unified. Assume a 2×2 block matrix

$$\begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} \approx UXV^H$$

consisting of four low-rank matrices $A_i = U_i X_i V_i^H$ with U_i, V_i, $i = 1, ..., 4$ each having k orthnonormal columns, is to be approximated by a single matrix $A = UXV^H \in \mathbb{C}_k^{N\times M}$. Since

$$\begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} = \begin{pmatrix} A_1 & \\ & \end{pmatrix} + \begin{pmatrix} & A_2 \\ & \end{pmatrix} + \begin{pmatrix} & \\ A_3 & \end{pmatrix} + \begin{pmatrix} & \\ & A_4 \end{pmatrix},$$

this problem may be regarded as a rounded addition of four low-rank matrices [30].Therefore, a best approximation in $\mathbb{C}_k^{N\times M}$ can be computed using the truncated SVD on a QR decomposition of low-rank matrices like in section 2.1.4.
In this rounded addition, the presence of zeros needs to be taken into account. Since

$$\begin{pmatrix} A_1 & A_2 \\ A_3 & A_4 \end{pmatrix} = \hat{U}\hat{X}\hat{V}^H, \text{ where } \hat{U} = \begin{pmatrix} U_1 U_2 & \\ & U_3 U_4 \end{pmatrix}, \hat{V} = \begin{pmatrix} V_1 & V_2 \\ V_3 & V_4 \end{pmatrix},$$

and $\hat{X} = (X_i, i = 1, .., 4)$, it satisfies to compute the QR decompositions $[U_1, U_2] = Q_1 R_1$, $[U_3, U_4] = Q_2 R_2$, $[V_1, V_3] = Q_3 R_3$ and $[V_2, V_4] = Q_4 R_4$.
Let $R_i = [R_i', R_i'']$ be partitioned with $R_i', R_i'' \in \mathbb{C}^{k\times 2k}$, then

$$\hat{U}\hat{X}\hat{V}^H = \begin{pmatrix} Q_1 & \\ & Q_2 \end{pmatrix} \begin{pmatrix} R_1' X_1 R_3'^H & R_1'' X_2 R_4'^H \\ R_2' X_3 R_3''^H & R_2'' X_4 R_4''^H \end{pmatrix} \begin{pmatrix} Q_3^H & \\ & Q_4^H \end{pmatrix}.$$

2.3.3 Multiplication

Definition 2.3.4 (Formatted multiplication)**.** Let $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ denote a block cluster tree and $k \in \mathbb{N}$. The formatted multiplication of $\mathcal{H}$-matrices $A, B \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$ is defined by :

$$A \odot B = T_k(A \cdot B).$$

The formatted multiplication is easy to write formally, but much harder to compute. Indeed, the block structure of the original $\mathcal{H}$-matrices is not retained in the multiplication, and can become rather complicated.

Two options are then available: Either trying to preserve the block structure and agglomerate subblocks inside of it, or creating a product block cluster tree. The first solution was the one originally implemented in the `Matlab` library from part 4, but it ended up not being the optimal solution for this project. The second one, used in both libraries explored in this paper, will be explained more in detail.

Definition 2.3.5 (Product tree)**.** The product tree $\mathcal{T}_{\mathcal{IJK}}$ of $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ and $\mathcal{T}_{\mathcal{J}\times\mathcal{K}}$ is inductively defined by:

1. $\mathcal{I}\times\mathcal{K}$ is the root of $\mathcal{T}_{\mathcal{IJK}}$;
2. The sets of sons of blocks $t \times s \in \mathcal{T}_{\mathcal{IJK}}$ from the lth level of $\mathcal{T}_{\mathcal{IJK}}$ is

$$S_{\mathcal{IJK}}(t\times s) = \{t'\times s' | \exists r \in \mathcal{T}_{\mathcal{J}}^{(l)}, r' \in \mathcal{T}_{\mathcal{J}}^{(l+1)} \ : t'\times r' \in S_{\mathcal{J}\times\mathcal{K}}(r\times s)\}.$$

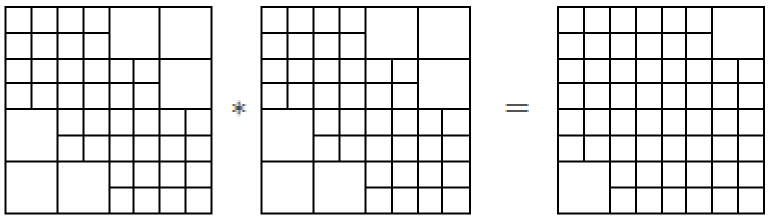


Figure 2.3: Example product tree

2.3.4 Inversion

Definition 2.3.6 (Preliminary formatted inversion)**.** Let $\mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ be a block cluster tree and $k \in \mathbb{N}$. The preliminary formatted inversion operator is defined as

$$\widetilde{Inv} : \{A \in \mathbb{C}^{N\times N} | rank(A) = N\} \to \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k), \quad A \mapsto T_k(A^{-1}).$$

Fully inverting a whole matrix is far too computationally expensive, so the algorithms presented in section 3 will compute an approximation of the inverse. While not necessarily being the best approximation, it will not need to invert the matrix exactly. This inversion will be done by the use of the equation :

$$A^{-1} = \begin{pmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}S^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}S^{-1} \\ -S^{-1}A_{21}A_{11}^{-1} & S^{-1} \end{pmatrix}, \quad A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \in \mathbb{C}^{N\times N} \tag{2.12}$$

where $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$. If the inversion of the submatrices A_{11} and S is already done, only multiplications and additions of subblocks need to be performed, and they can be replaced by their formatted versions. This recursively leads to an approximate inverse $Inv_{\mathcal{H}}(A)$, called the formatted inverse.

Most often, computing the actual inverse will be avoided, and replaced by other methods such as an LU decomposition.

2.3.5 LU Decomposition

Sometimes, only a method to perform the matrix-vector multiplication $b \mapsto A^{-1}b, \quad A \in \mathbb{C}^{N\times N}$ (i.e. to solve the system $Ax = b$) is needed. In that case, computing an LU decomposition of A is sufficient. The original LU decomposition is fully explained in C. Only an approximate one, for $\mathcal{H}$-matrices, called $\mathcal{H}$-LU will be studied in this section.

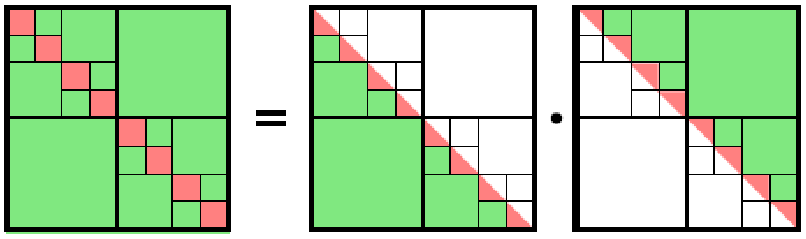


Figure 2.4: Example $\mathcal{H}$-matrix structure of an LU decomposition

This decomposition is of the form $A \approx LU$ where L and U are stored in the $\mathcal{H}$-matrix format.

Let $A \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{I}})$, and $t \in \mathcal{T}_{\mathcal{I}} \setminus \mathcal{L}(\mathcal{T}_{\mathcal{I}})$. Exploiting the hierarchical structure of A_{tt} :

$$A_{tt} = \begin{pmatrix} A_{t_1t_1} & A_{t_1t_2} \\ A_{t_2t_1} & A_{t_2t_2} \end{pmatrix} = \begin{pmatrix} L_{t_1t_1} & 0 \\ L_{t_2t_1} & L_{t_2t_2} \end{pmatrix} \begin{pmatrix} U_{t_1t_1} & U_{t_1t_2} \\ 0 & U_{t_2t_2} \end{pmatrix},$$

where $t_1, t2 \in \mathcal{T}_{\mathcal{I}}$ denote the sons of t in $\mathcal{T}_{\mathcal{I}}$. Hence, the LU decomposition of a block A_{tt} is recursively defined by :

1. Compute $L_{t_1t_1}$ and $U_{t_1t_1}$ from the LU decomposition $L_{t_1t_1}U_{t_1t_1} = A_{t_1t_1}$;
2. Compute $U_{t_1t_2}$ from $L_{t_1t_1}U_{t_1t_2} = A_{t_1t_2}$;
3. Compute $L_{t_2t_1}$from $L_{t_2t_1}U_{t_1t_1} = A_{t_2t_1}$;
4. Compute $L_{t_2t_2}$ and $U_{t_2t_2}$ from the LU decomposition $L_{t_2t_2}U_{t_2t_2} = A_{t_2t_2} - L_{t_2t_1}U_{t_1t_2}$.

If a block $t \times t \in \mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ is a leaf, the usual pivoted LU decomposition is used.
Steps 1. and 4. require the computation of two LU decompositions of half the original size. Step 2. needs to be solved as a triangular $\mathcal{H}$-matrix system for a matrix. If $t \times s \in \mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ is not a leaf, the following method is used to solve $L_{tt}B_{ts} = A_{ts}$ for B_{ts} (with t_1, t_2 and s_1, s_2 the sons of t and s respectively) :

$$\begin{pmatrix} L_{t_1t_1} & L_{t_1t_2} \\ L_{t_2t_1} & 0 \end{pmatrix} = \begin{pmatrix} B_{t_1s_1} & B_{t_1s_2} \\ B_{t_2s_1} & B_{t_2s_2} \end{pmatrix} \begin{pmatrix} A_{t_1s_1} & A_{t_1s_2} \\ A_{t_2s_1} & A_{t_2s_2} \end{pmatrix}$$

leads to

$$L_{t_1t_1}B_{t_1s_1} = A_{t_1s_1}, \tag{2.13}$$
$$L_{t_1t_1}B_{t_1s_2} = A_{t_1s_2}, \tag{2.14}$$
$$L_{t_2t_2}B_{t_2s_1} = A_{t_2s_1} - L_{t_2t_1}B_{t_1s_1}, \tag{2.15}$$
$$L_{t_2t_2}B_{t_2s_2} = A_{t_2s_2} - L_{t_2t_1}B_{t_1s_2}, \tag{2.16}$$

which is again a form of step 2. If $t \times s$ is a leaf, the usual block-forward substitution is applied. Step 3. can be solved similarly to step 2. by using block-backward substitution.

If the matrix A is symmetric positive definite, an $\mathcal{H}$-matrix version of the Cholesky decomposition can be implemented instead, as explained in [30]. This will most often not be the case in this project, so it will not be covered here.

2.3.6 Using $\mathcal{H}$-matrices for preconditioning

Solving an equation such as $Ax = b$ usually leads to considering a sequence of systems $A_nx_n = b_n$, $n \to \infty$, where each $A_n \in \mathbb{C}^{N\times N}$ is invertible. Using a direct solver such as Gaussian elimination to a fully populated system leads to huge complexities in time and space.
In principle, the hierarchical LU decomposition from section 2.3.5 can be used to compute an approximate LU decomposition with almost linear complexity. $\mathcal{H}$-matrices, however, can be multiplied with a vector with complexity $\mathcal{O}(kn \log n)$. Iterative Krylow-subspace methods such as GMRES will hence be faster provided that the number of iterations is small enough.
The convergence of Krylow-subspace methods is determined by spectral properties of the matrix A, which may thus cause the $\mathcal{H}$-matrix approximant $A_{\mathcal{H}}$ to be ill-conditioned, which is why the use of a preconditioner is required here. Since $\mathcal{H}$-matrices provide efficient approximations to the LU decomposition, they are particularly suited for preconditioning. If $A_{\mathcal{H}} \approx L_{\mathcal{H}}U_{\mathcal{H}}$, a preconditioner $C = (L_{\mathcal{H}}U_{\mathcal{H}})^{-1}$ can be used on the left- or right-hand side to solve the respective equivalent systems: $CA_{\mathcal{H}}x = Cb$ or $A_{\mathcal{H}}C\tilde{x} = b$, with the solution obtained as $x = C\tilde{x}$. It can be proven [30] that a low-accuracy approximate inverse or LU is sufficient to to guarantee a bounded number of preconditioned iterations of appropriate iterative schemes. Moreover, this number of iterations solely depends on the chosen accuracy for the solver.

Chapter 3

The $\mathcal{H}$-matrix library in Matlab language : `hmtxLib`

The code implementing the BEM-SIBC method is developed in the Matlab environment, with some computational kernels written in C language. The $\mathcal{H}$-matrix library in Matlab language, namely `hmtxLib`, is the first library that was implemented and used in this project for mathematical operations on $\mathcal{H}$-matrices.

This library is a full implementation of $\mathcal{H}$-matrices and of their arithmetics. Some of the core functions were coded to create or change the data structure, fill the matrix, check memory usage, plot the structure and check whether an input is an $\mathcal{H}$-matrix. Then, another set of functions was created for compression purposes: a function coarsening the matrix, hence reducing the accuracy and precision, a reduced singular value decomposition (rSVD) following [35] for an easy storage of the matrix, and the ACA method from [28]. Finally, various matrix operations, such as addition, multiplication, getting the upper- or lower-triangular part of the matrix, or transposition were implemented. The latest codes were created to solve matrix systems. After computing the $\mathcal{H}$-inverse of the $\mathcal{H}$-matrix, matrix-vector solvers and $\mathcal{H}$-LU decomposition were implemented.

3.1 $\mathcal{H}$-matrix implementation : creating and filling an $\mathcal{H}$-matrix structure

In this library, $\mathcal{H}$-matrices are created using a function called `hmtx_cluster` which creates an $\mathcal{H}$-matrix structure from a cluster of points, and an optional second cluster of points. In the previous electromagnetic model 1.3, two points in the system which are at a shorter distance interact more than two points which are further apart. The goal of this library is to create, from these clusters, an efficient matrix representation of the value of interest between all the points.

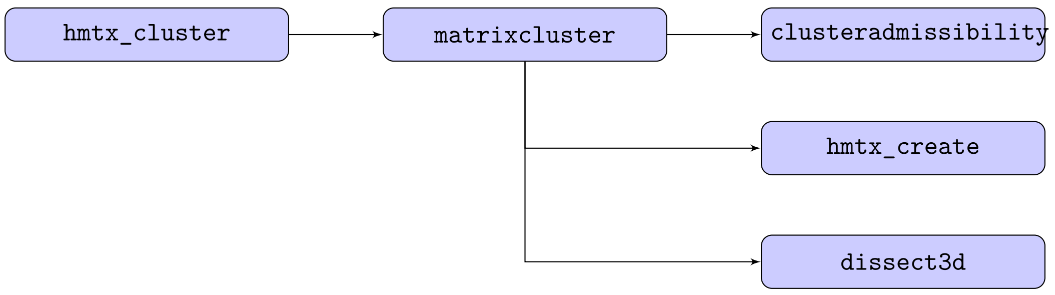


Figure 3.1: Library structure for matrix cluster creation

An $\mathcal{H}$-matrix M needs to be stored in an efficient way, so as to actually reduce calculation time and memory usage. Each matrix is thus stored by $n \times m$ blocks.
For each $M|_{n\times m}$ block there are three available data types:

- An $n \times m$ non-admissible block that can be subdivided, `supermatrix`, which is stored as a 2×2 cell array. Inside of each cell, there is another block;
- A full $n \times m$ block, `fullmatrix`, which is a `Matlab` array;

- For admissible blocks, a rank-k block, `rkmatrix`, in which only U (an $n \times k$ matrix, where k is the rank of the low-rank approximation) and V (an $m \times k$ matrix) from the ACA decomposition are stored.

Each block H is itself an $\mathcal{H}$-matrix, which has several attributes, assigned in the function `hmtx_create`:

- `H.type` which can be `supermatrix`, `fullmatrix` or `rkmatrix`;
- `H.irow` and `H.jcol`, global indices of rows and columns of H (following the reordering explained below);
- `H.nrow` and `H.ncol`, number of rows and columns of H;
- `H.M` containing, if H is not admissible, a 2×2 cell array if H can be subdivided, a full `Matlab` array otherwise;
- `H.U, H.V, H.k` containing, if H is admissible, the ACA approximation $H = UV^H$ and the `rkmatrix` rank;
- `H.eps` the tolerance of the ACA method ε;
- `H.kMax` the maximum `rkmatrix` rank allowed.

Blocks are divided using `dissect3d` 5, a function which takes points coordinates and splits a cluster into two sets of points according to distance.

Algorithm 5: `dissect3d` : splitting a cluster of points into two sub-clusters according to distance

```
Inputs : $P$ points coordinates
Outputs : $idx1$, $idx2$ indices of the two sub-clusters
Compute $X_c$ cluster center
Compute $C$ covariance matrix of the cluster
Get eigenvalues $D$ and eigenvectors $V$ of $C$
$\lambda = \max(D)$, $V_{max} = \max(V)$                    Maximum eigenvalue and eigenvector
$idx1 = \{p \in P \mid (p - X_c)V_{max} > 0\}$
$idx2 = \{p \in P \mid (p - X_c)V_{max} \leq 0\}$
```

$idx1$ and $idx2$ then give rise to four sub-blocks `A11`, `A12`, `A21` and `A22`, in which the represented indices are stored as a list.

To determine block types, several conditions are then checked. First, using the function `clusteradmissibility`, which takes as inputs two clusters of points and an admissibility parameter η, the admissibility of the block is determined.

Algorithm 6: `clusteradmissibility`

```
Inputs : $P_1, P_2, \eta$ clusters of points and admissibility parameter (by default : $\eta = 2$)
Outputs : logical variable (1 if admissible, 0 otherwise)
Create cluster bounding boxes minP1, maxP1, minP2, maxP2
Compute $Z$ origin of $P_1$
Compute clusters diameters $D_1$ and $D_2$
Compute $d$ euclidean distance between the two clusters
return $\min(D_1, D_2) < \eta d$
```

If H is admissible, then it is of `rkmatrix` type. Otherwise, its size needs to be checked : if the block size is smaller than N_{min}, minimum block size allowed (by default, $N_{min} = 32$), then H is a `fullmatrix`. If it is bigger, H is a `supermatrix`.

Once the whole structure of the $\mathcal{H}$-matrix is created, it can then be filled using `hmtx_fill`. This function first fills the $\mathcal{H}$-matrix and then recompresses `rkmatrix` and `fullmatrix` blocks using `hmtx_compress`. It takes as inputs the cluster tree created in the previous function, the kernel function of the matrix and a tolerance ε.

If H is a `supermatrix`, the function is called recursively. If H is a `fullmatrix`, H is filled by evaluating the kernel function at each point of the cluster, and ε is set to 0. Lastly, if H is an `rkmatrix` and $\varepsilon = \infty$, then the ACA approximation is set as empty blocks and the rank k of the approximation is equal to 0. Otherwise, `H.U, H.V, H.k` and `H.eps` are updated using the ACA algorithm 4. H is then recompressed using `rSVD_rkmatrix` which computes a truncated SVD of H using ε and k_{Max} 2.1.4. `hmtx_compress` then loops over the $\mathcal{H}$-matrix

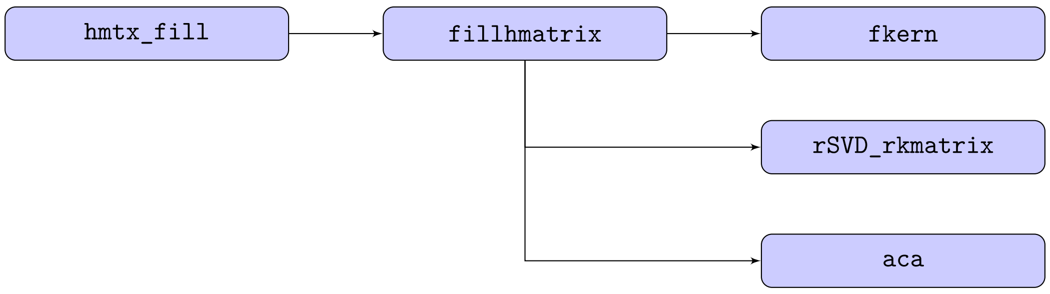


Figure 3.2: Library structure for $\mathcal{H}$-matrix filling

and unifies four adjacent `fullmatrix` blocks, or compresses four adjacent `rkmatrix` blocks, if it optimizes the storage, as explained in section 2.3.2.

Once the $\mathcal{H}$-matrix is created and filled, it can then be used in mathematical operations, be modified and plotted.

3.2 $\mathcal{H}$-matrix representation

These functions are at the core of the library, are they are used to evaluate results or prepare the matrix for an arithmetical operation.

3.2.1 Changing the type of an $\mathcal{H}$-matrix block with `hmtx_changeformat`

`hmtx_changeformat` converts a generic $\mathcal{H}$-matrix to another $\mathcal{H}$-matrix with a different cluster tree. This function takes as inputs and input $\mathcal{H}$-matrix and an output one with a different format. The output $\mathcal{H}$-matrix is then returned filled with the entries of the input one. depending on the type of the input and output matrices, different sub-routines are used :

- `super2full` which recursively converts every sub-block to a `fullmatrix` (using `rkmatrix2full`) and then compresses all four blocks into one using `hmtx_compress`;
- `super2rkmatrix` which works in the same way as the previous function, but using `full2rkmatrix` instead;
- `rkmatrix2full` which creates a full block in a matrix M from $M.U \times M.V'$, i.e. computing the multiplication of the ACA blocks;
- `full2rkmatrix` computes a truncated SVD of the input matrix M given a maximum rank. If the truncated SVD is USV, then $A.U = US$ and $A.V = V$;
- `rkmatrix2super` splits an `rkmatrix` into four sub-blocks of equal size. each block is itself an `rkmatrix` which is stored as a sub-block of the ACA of the original matrix;
- `full2super` works exactly like `rkmatrix2super`, but matrix blocks are `fullmatrix` sub-blocks of the original `fullmatrix`.

3.2.2 Copying a cluster tree with `hmtx_copystruct`

The function `hmtx_copystruct` takes as input an $\mathcal{H}$-matrix M and creates an empty matrix with the same structure that the one of M. This function recursively creates empty blocks that exaclty follow the structure of the input matrix.

3.2.3 Fullmatrix times $\mathcal{H}$-matrix product using `hmtx_MxH`

`hmtx_MxH` performs the fullmatrix times $\mathcal{H}$-matrix product. This recursive function takes an $\mathcal{H}$-matrix H and a full matrix M as inputs, and returns $A = MH$ using global indices of H stored in `H.irow` and `H.jcol`. At each step, the function adds to the (originally full of zeros) matrix A the product of M with a block of H, only taking in M the global indices present in the block of H.

3.2.4 Plotting an $\mathcal{H}$-matrix

An $\mathcal{H}$-matrix is plotted by outlining each sub-block with black lines. Full blocks are filled with white, while `rk` blocks are filled with a color corresponding to their rank, determined by a scale shown on the right of the matrix.

This matrix was created by calculating the euclidean distance between points on a meshed sphere.

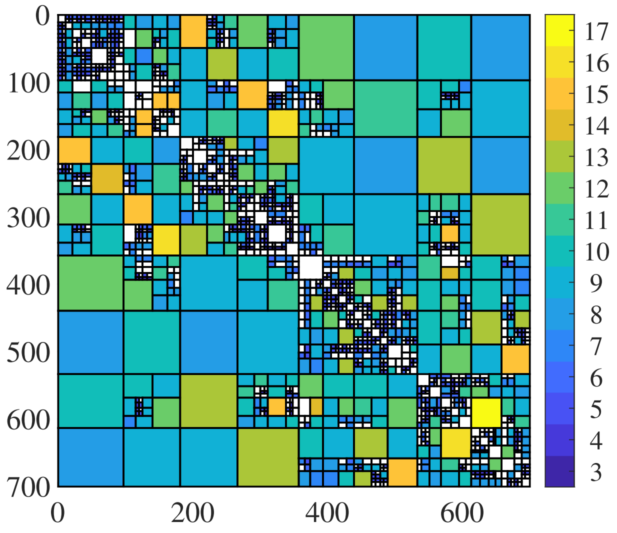


Figure 3.3: Example plot of a 700 × 700 $\mathcal{H}$-matrix, block colors correspond to the rank written on the scale, full blocks are shown in white

3.2.5 Coarsening an $\mathcal{H}$-matrix

The function `hmtx_coarsening` recursively coarsens an $\mathcal{H}$-matrix to a new tolerance or maximum rank. It simply removes lines and columns from the ACA decomposition of low-rank blocks.

3.3 Arithmetics

3.3.1 Simple operations

Two functions, `hmtx_tril` and `hmtx_triu` were coded to respectively extract the lower and upper triangular parts of an $\mathcal{H}$-matrix. Then, the transposition and the hermitian of a matrix were implemented. Those functions simply work recursively over the matrix blocks. The hermitian of a matrix H will be denoted as H' in the codes here.

3.3.2 Addition and subtraction

`hmtx_add` takes as inputs two $\mathcal{H}$-matrices A, B and a sign t ($+1$ or -1) and returns $A + tB$. A and B must have the same cluster tree in order for the addition to be successfully computed.

Algorithm 7: `hmtx_add`

```
Inputs : $A, B, t$
Outputs : $C = A + tB$
if $A$ and $B$ are supermatrices then
    Create $C$, supermatrix. Call the function recursively on each block:
    $C_{11} = A_{11} + tB_{11}$
    $C_{12} = A_{12} + tB_{12}$
    $C_{21} = A_{21} + tB_{21}$
    $C_{22} = A_{22} + tB_{22}$
else
    if $A$ and $B$ are fullmatrices then
        Create $C$, fullmatrix. C.M = A.M+t*B.M, C.eps = max(A.eps, B.eps)
    else
        Create $C$, rkmatrix. C.U = [A.U, t*B.U], C.V = [A.V, B.V], C.k=A.k+B.k. Recompress C
         using rSVD_rkmatrix.
    end
end
return C
```

This algorithm follows the rules of the rounded addition explained in section 2.3.1.

3.3.3 Multiplication

`hmtx_mult` takes as inputs two $\mathcal{H}$-matrices A, B, and the desired output cluster tree C and returns AB. In order to have a fully optimized algorithm, every possible combination of $\mathcal{H}$-matrix block type is tested.

Algorithm 8: `hmtx_mult`

```
Inputs : $A, B, C$
Outputs : $C = AB$
if $A, B$ are supermatrices then
    Create Ctmp as a supermatrix of size A.irow × B.jcol. for $i \in \{1,2\}$ do
        for $j \in \{1,2\}$ do
            Create $C_a$, $C_b$ as supermatrices of size A.irow × B.jcol. Call function recursively :
            $C_a = A_{i,1}B_{1,j}$
            $C_b = A_{i,2}B_{2,j}$
            Add resulting blocks : $\texttt{Ctmp}_{i,j} = C_a + C_b$
        end
    end
    Change matrix format to goal format : C = hmtx_changeformat(Ctmp,C)
end
if A is a fullmatrix and B is a supermatrix then
    Create Ctmp as a fullmatrix of size A.irow × B.jcol. Perform fullmatrix times $\mathcal{H}$-matrix
     product:
    Ctmp.M = hmtx_MxH(A.M, B)
    Change matrix format to goal format :C = hmtx_changeformat(Ctmp,C)
end
if A is a rkmatrix and B is a supermatrix then
    Create Ctmp as an rkmatrix of size A.irow × B.jcol. Perform fullmatrix times $\mathcal{H}$-matrix product:
    Ctmp.V = hmtx_MxH(A.V', B)'
    Ctmp.U = A.U
    Ctmp.k = A.k
    Recompress Ctmp using rSVD_rkmatrix. Change matrix format to goal format : C =
     hmtx_changeformat(Ctmp,C)
end
if A and B are fullmatrices then
    Create Ctmp as an fullmatrix of size A.irow × B.jcol. Ctmp.M = A.M*B.M
    Change matrix format to goal format : C = hmtx_changeformat(Ctmp,C)
end
if A and B are rkmatrices then
    Create Ctmp as an rkmatrix of size A.irow × B.jcol. Ctmp.U = A.U
    Ctmp.V = B.V*(B.U'*A.V)
    Ctmp.k = A.k
    Recompress Ctmp using rSVD_rkmatrix. Change matrix format to goal format : C =
     hmtx_changeformat(Ctmp,C)
end
if A is a rkmatrix and B is a fullmatrix then
    Create Ctmp as an rkmatrix of size A.irow × B.jcol. Ctmp.V = B.M'*A.V
    Ctmp.U = A.U
    Ctmp.k = A.k
    Recompress Ctmp using rSVD_rkmatrix. Change matrix format to goal format : C =
     hmtx_changeformat(Ctmp,C)
else
    For all other cases, mirror previous cases.
end
return C
```

3.3.4 Inversion

Using the equation 2.12, the function `hmtx_inv` inverts an $\mathcal{H}$-matrix. This function is a rough inversion of an $\mathcal{H}$-matrix and is too computationally expensive to be truly used. This was only coded for our tests and comparisons with the LU inversion.

Algorithm 9: `hmtx_inv`

```
Inputs : $M$
Outputs : $M^{-1}$
if $M$ is a supermatrix then
    Use equation 2.12, and perform operations using hmtx_add and hmtx_mult:
    Invert $M_{11}$.
    Create $X$ supermatrix used for temporary storage.
    $X_{12} = M_{11}^{-1}M_{12}$
    $X_{21} = M_{21}M_{11}^{-1}$
    $M_{22} = S = M_{22} - M_{21}X_{12}$
    $M_{22} = S^{-1}$
    $M_{12} = -X_{12}M_{22}$
    $M_{11} = M_{11} + M_{12}X_{21}$
    $M_{21} = -M_{22}X_{21}$
else
    if $M$ is a fullmatrix then
        Invert $M$.
    end
end
return $M$
```

3.3.5 LU decomposition

As explained in section 2.3.5, matrix-vector- and triangular system solvers need to be implemented before actually coding a $\mathcal{H}$-LU function. In algorithm 10, both types of systems can be solved, in just one function. This function can thus also serve as a matrix-vector solver if needed.

Algorithm 10: `hmtx_lsolve`

```
Inputs : $L, B$ with $L$ lower triangular
Outputs : $X$ such that $LX = B$
if $L, B$ are supermatrices then
    Perform operations using hmtx_add and hmtx_mult following section 2.3.5:
    $L_{11}X_{11} = B_{11}$
    $L_{11}X_{12} = B_{12}$
    $L_{22}X_{21} = B_{21} - L_{21}X_{11}$
    $L_{22}X_{22} = B_{22} - L_{21}X_{12}$
end
if $L, B$ are fullmatrices then
    $X = L\backslash B$.
end
if $L$ is a fullmatrix then
    if $B$ is an rkmatrix then
        $X.U = L\backslash B.U$
        $X.V = B.V$
    end
    if $B$ is an supermatrix then
        Create a temporary fullmatrix copy of $B$, solve for $X$ and convert $X$ back to a supermatrix
          using hmtx_changeformat.
    end
end
if $L$ is a supermatrix and $B$ isn't then
    Split $B$ into four blocks if it has more than 1 column. If it doesn't, then $B$ is a vector, and convert
      $L$ to fullmatrix format. Recursively call solving function.
end
return $X$
```

Solving a matrix system $XU = B$ for X, with U upper triangular is done essentially in the same way. The same function can even be used as, by transposing the equation $XU = B$ it becomes $U^TX^T = B^T$ with U^T lower triangular. A function called `hmtx _solve` is then used to solve a triangular system either from the right or left side, using `hmtx_lsolve`.

Algorithm 11: `hmtx_lu`

```
Inputs : $M$
Outputs : $L, U$
if $M$ is a supermatrix then
    Use equation 2.16, and perform operations using hmtx_add and hmtx_mult. Solve triangular
     systems for $L$ and $U$ blocks using hmtx_solve:
    $[L_{11}, U_{11}] = \mathtt{lu}(M_{11})$
    $L_{11}U_{12} = M_{12}$
    $L_{21}U_{11} = M_{21}$
    $L_{22}U_{22} = M_{22} - L_{21}U_{12}$
else
    if $M$ is a fullmatrix then
        $[L, U] = lu(M)$.
    end
end
return $L, U$
```

3.4 Preliminary results

The $\mathcal{H}$-matrix technique aims to enable operations of almost linear complexity. The following benchmarks were done on a Intel Core i7-8750H CPU @2.20 GHz, with 8 GB of RAM. As no other computers were available at the time, some tests on bigger $\mathcal{H}$-matrices could not be computed, due to a lack of memory on this one. All tests were computed with a prescribed accuracy of 10^{-4}.

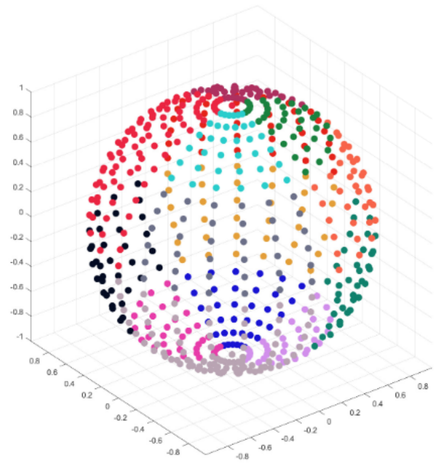


Figure 3.4: Charged points on a sphere

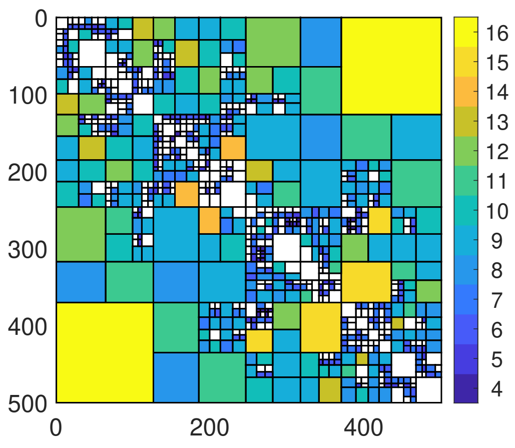


Figure 3.5: Corresponding $\mathcal{H}$-matrix, block colors correspond to the rank written on the scale, full blocks are shown in white

3.4.1 Memory and entry compression

Definition 3.4.1 (Sparsity and storage)**.** Let $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree based on $\mathcal{T}_{\mathcal{I}}$. The sparsity constant C_{sp} of $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ is defined by

$$C_{sp} = \max\{\max_{r\in\mathcal{T}_{\mathcal{I}}} \#\{s \in \mathcal{T}_{\mathcal{I}} \,|\, r \times s \in \mathcal{T}_{\mathcal{I}\times\mathcal{I}}\}, \max_{s\in\mathcal{T}_{\mathcal{I}}} \#\{r \in \mathcal{T}_{\mathcal{I}} \,|\, r \times s \in \mathcal{T}_{\mathcal{I}\times\mathcal{I}}\}\}. \tag{3.1}$$

If $\mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ is of depth p, and $k \in \mathbb{N}$, the storage requirements $N_{St}(\mathcal{T}_{\mathcal{I}\times\mathcal{I}}, k)$ for a matrix $M \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{I}}, k)$ are bounded by :

$$N_{St}(\mathcal{T}_{\mathcal{I}\times\mathcal{I}}, k) \leq 2C_{sp}(p+1)\max\{k, n_{min}\}\#\mathcal{I} \tag{3.2}$$

where n_{min} is the minimum block size.

The following graph 3.6 verifies this inequality experimentally, on the $\mathcal{H}$-matrices created in the `HmtxLib` library.

3.4.2 Primary (fast) functions

In this section, matrix-vector multiplication, transposition, scalar multiplication, addition or matrix-vector system solving are considered as primary operations, as opposed to longer, more expensive ones such as inversion.

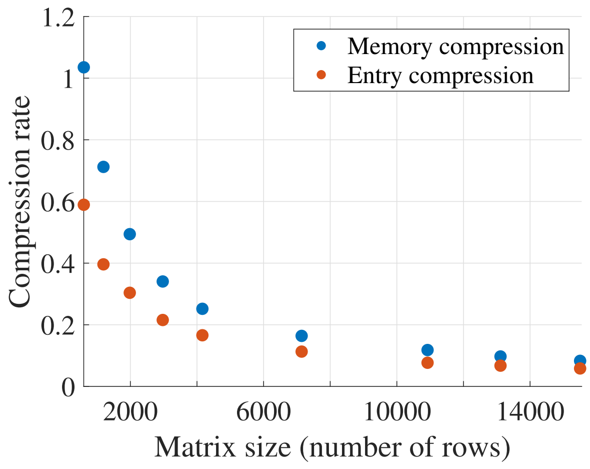


Figure 3.6: Memory and entry compression of the $\mathcal{H}$-matrix compared to the original matrix

Definition 3.4.2 (Matrix-Vector multiplication complexity)**.** Let $T = \mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree, $k \in \mathbb{N}$. The complexity $N_{Hv}(T,k)$ of the matrix-vector multiplication for a matrix $M \in \mathcal{H}(T,k)$ are bounded by

$$N_{Hv}(T,k) \leq N_{St}(T,k). \tag{3.3}$$

Definition 3.4.3 (Addition complexity)**.** Let $T = \mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree and $k \geq 1$. The complexity of the formatted addition of two matrices from $\mathcal{H}(T,k)$ is bounded by

$$N_{\oplus}(T,\, k) \leq 24kN_{St}(T,\, k) + 184k^3\#\mathcal{L}(T). \tag{3.4}$$

The tests conducted on a limited number of $\mathcal{H}$-matrices show below (figure 3.9) that the `HmtxLib` implementation show that complexity for primary functions such as addition or transposition are all of almost linear complexity.

The relative error remains below the prescribed accuracy of 10^{-4}, which shows these functions are fast but still sufficiently accurate.

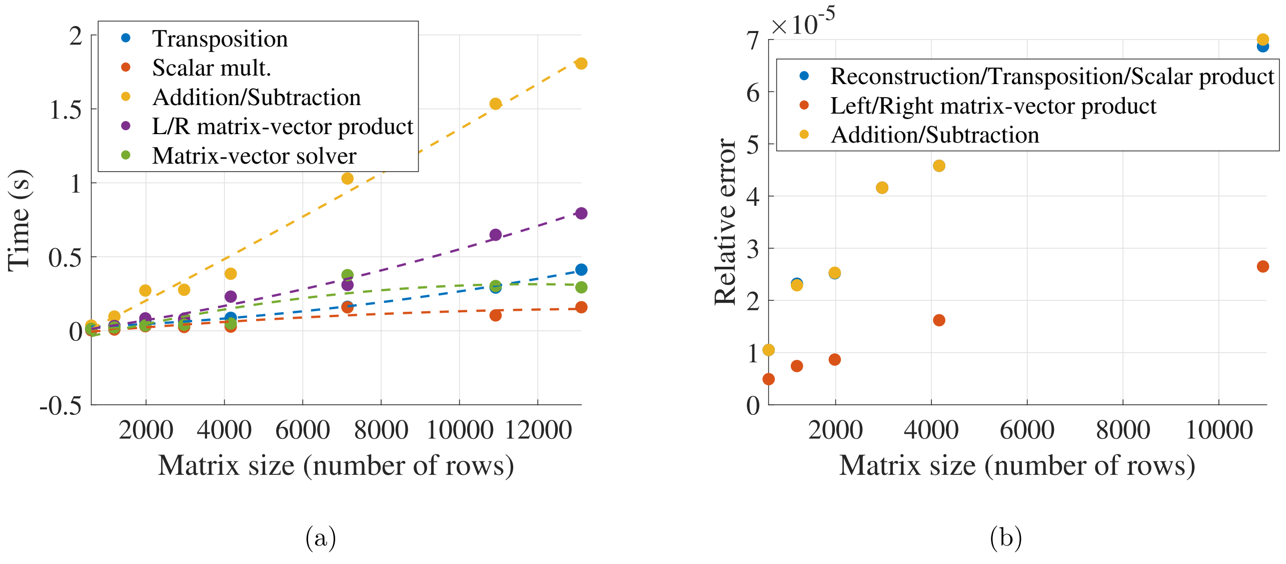


Figure 3.7: (a) Computation time of primary functions and (b) Accuracy of primary functions on $\mathcal{H}$-matrices

3.4.3 Secondary (slow) functions : multiplication, inversion, triangular system solvers and LU decomposition

Some more complex functions naturally require more computational time, and are thus classified as "secondary functions".

Definition 3.4.4 (Idempotency)**.** Let $T = \mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree based on $\mathcal{T}_{\mathcal{I}}$. The elementwise idempotency $C_{id}(r \times t)$ and idempotency constant $C_{id}(T)$ are defined by :

$$C_{id}(r \times t) = \#\{r' \times t' \,|\, r' \in S*(t) \text{ and } \exists s' \in \mathcal{T}_{\mathcal{I}} : r' \times s' \in T,\, s' \times t' \in T\}$$
$$C_{id}(T) = \max_{r \times t \in \mathcal{L}(T)} C_{id}(r \times t)$$

If the tree T is fixed, C_{id} can be used instead of $C_{id}(T)$.

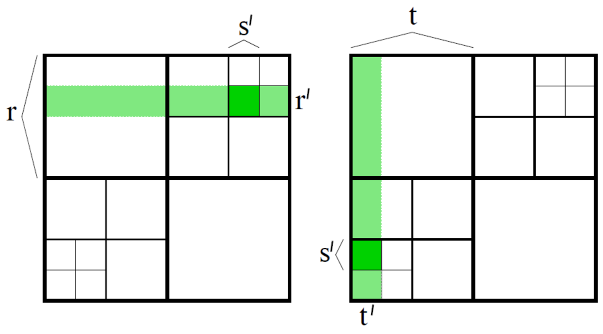


Figure 3.8: Example: The idempotency constant $C_{id}(r \times t)$ of the leaf $r \times t$ is 9 [35].

Definition 3.4.5 (Formatted multiplication complexity)**.** Let $T = \mathcal{T}_{\mathcal{I}\times\mathcal{I}}$ be a block cluster tree with idempotency constant C_{id}, sparsity constant C_{sp} and depth p. It is assumed once again that $n_{min} \leq k$. The exact multiplication is a mapping $\cdot : \mathcal{H}(T,k) \times \mathcal{H}(T,k) \to \mathcal{H}(T,\tilde{k})$ with some $\tilde{k}$ bounded by

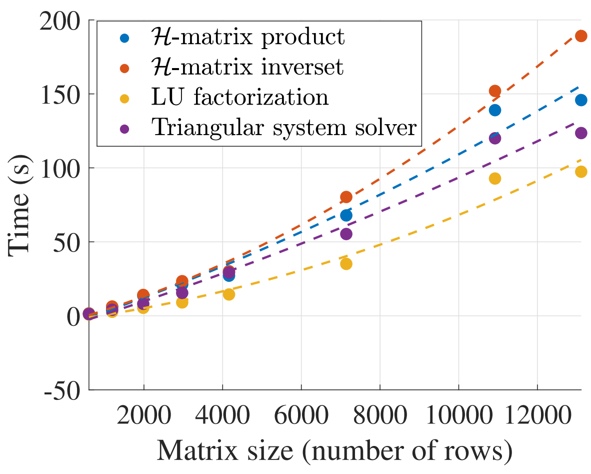
$$\tilde{k} \leq C_{id}C_{sp}(p+1)k. \tag{3.5}$$

The formatted multiplication used here ($\mathcal{H}(T,k) \times \mathcal{H}(T,k) \to \mathcal{H}(T,k')$ for any $k' \geq 0$) is defined as the exact multiplication followed by the truncation T_k from lemma B.0.5 and can be computed with complexity

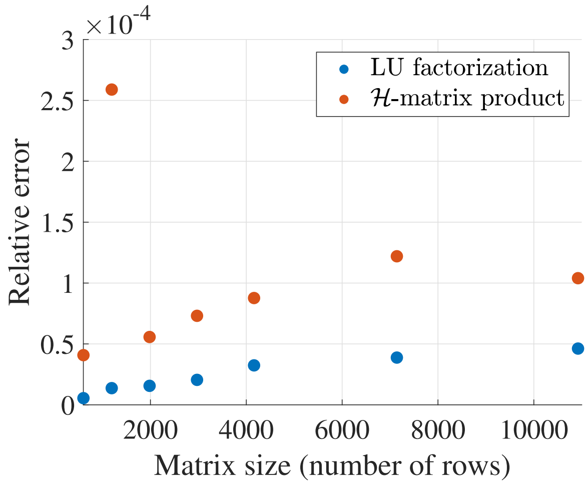
$$N_{\odot}(T,k) \leq 47C_{id}^3C_{sp}^3k^3(p+1)^3 \max\{\#\mathcal{I}, \#\mathcal{L}(T)\}. \tag{3.6}$$

While literature provides no complexity estimates for inversion or LU decomposition, the following statement can be made : If it is assumed, for `fullmatrix` blocks, that the complexity of the inversion is bounded by that of the multiplication, then the complexity of the formatted inversion is bounded by the one of the formatted multiplication. Moreover, it is always expected of a matrix inversion through LU decomposition to be much faster than a full-on one.

The results show, as expected, that while the formatted multiplication, and especially the formatted inversion, can be quite computationally expensive, the LU factorization takes a more reasonable time to compute.



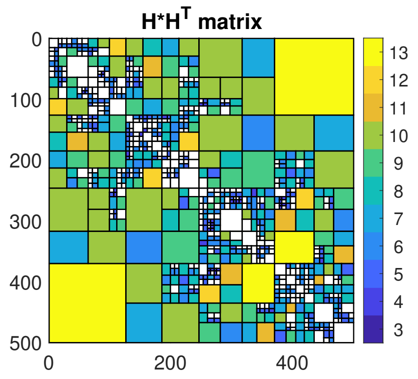
(a)



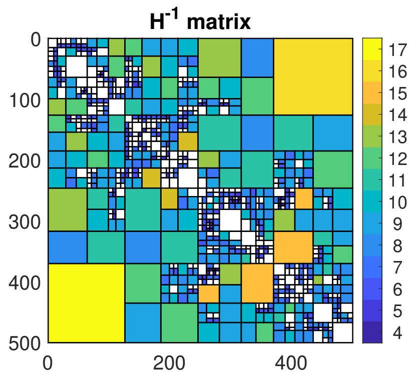
(b)

Figure 3.9: (a) Computation time of secondary functions and (b) Accuracy of secondary functions on $\mathcal{H}$-matrices

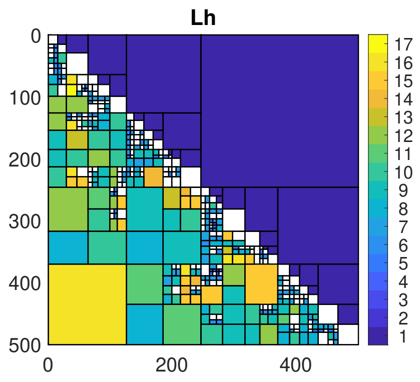
For both multiplication and LU factorization, the accuracy remains satisfying, i.e. the relative error stays below the prescribed tolerance of 10^{-4}. Plots of these operations are shown in figure 3.10.



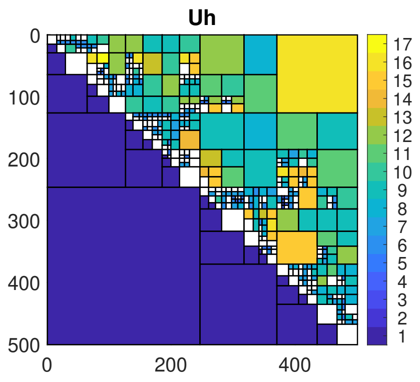
(a)



(b)



(c)



(d)

Figure 3.10: (a) Inversion, (b) Multiplication HH^T, (c) L factor, (d) U factor plots, block colors correspond to the rank written on the scale, full blocks are shown in white

Chapter 4

MATLAB toolbox for arithmetic operations of $\mathcal{H}$-matrices

While the first MATLAB library for $\mathcal{H}$-matrices remains the one for chapter 3, `hm-toolbox` was published on GitHub [36] a few weeks before all tests on `hmtxLib` were ready. This other library is a toolbox initially implementing the arithmetic of HODLR and HSS matrices, which are specific types of hierarchical and $\mathcal{H}^2$ matrices. A branch for $\mathcal{H}$-matrices in general was later added, which was compared with the library implemented in this project, in order to use the best one.

4.1 Library structure

A full description of the library can be found in [36]. As functions are essentially the same as the ones from `hmtxLib` but with a different syntax, it will not be explained in detail here. There are however a few differences between this library and the previous one, which makes the `hm-toolbox` software faster at first glance :

- Instead of using functions for arithmetic operations, MATLAB operators are overloaded. This means for example that writing $A * B$ will automatically compute the multiplication of MATLAB matrices, or $\mathcal{H}$-matrices, or of a MATLAB matrix with an $\mathcal{H}$-matrix, depending on the types of A and B;
- Unless a defined cluster tree is provided, a default cluster tree is used for every $\mathcal{H}$-matrix. This cluster tree exclusively has full blocks along the diagonal, and low-rank blocks off the diagonal.

However, this library structure hides a downside for the use of $\mathcal{H}$-matrices from non ideal problems. Indeed, using specific block types of specific sizes is not always optimal, and especially not in this project, and can use much more memory than a well optimized cluster tree. While inputting a prescribed structure can be done if one has a routine to create such a structure, arithmetics were clearly not coded having in mind the use of a non-default cluster tree. In the multiplication for example, if a full block is multiplied by a low-rank block, then the result becomes a full blocks. In tests from this project, this resulted in having an $\mathcal{H}$-matrix of full blocks. That is why this second library was reworked to see if it could be a faster way to solve our problems.

4.2 Work done on the library

Using `hm-toolbox` as a base, some modifications were made to optimize the library to solve real-world problems. Modifications include :

- An option for the maximum rank of a matrix was added in `hmatrixoption` and into the ACA decomposition;
- In the `hmatrix` class, a possibility to directly create a diagonal $\mathcal{H}$-matrix from a vector or a diagonal matrix was implemented;
- When multiplying a non-leafnode block with a leafnode one, the result now has the admissibility criterion and the format of the leafnode block. In the end, the algorithm for the multiplication was a hybrid between the original one and the one from `hmtxLib`. Other operations were then modified to fit with this new multiplication;
- A new function, `hmatrix_split` was created, to split a leafnode block into four given the sizes of the sub-blocks to be created;
- The function `cluster_mask` was written to change the structure of an $\mathcal{H}$-matrix given another $\mathcal{H}$-matrix which has the goal cluster.

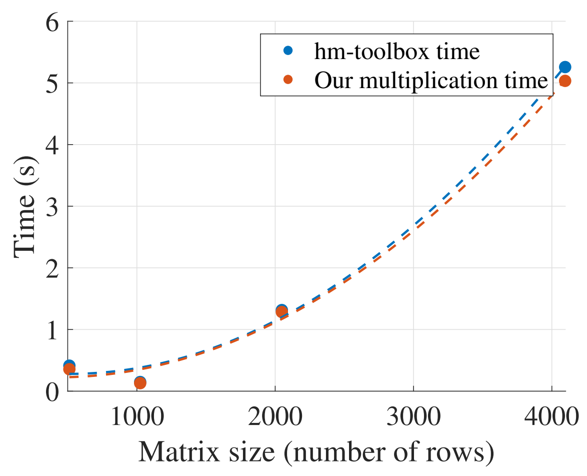
Using all the new routines with the hybrid multiplication ensured that results remained optimal in terms of memory occupation and accuracy, while still computing operations in a reasonable amount of time.

4.3 Benchmarks

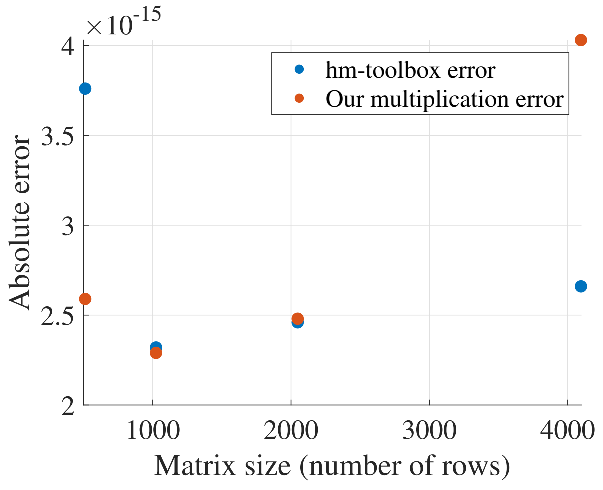
4.3.1 Tests from the original library on random matrices

This tests were done on the test files provided in the library, following sizes suggested in the library documentation [36], with initial block truncation at a 10^{-12} threshold. The aim is to keep a relative error below 10^{-4}. This is always true for both our functions and the original ones, so only absolute errors ($\|\mathcal{H}$-matrix$-$ MATLAB array$\|$) are compared here, to see up to where the accuracy can go.

For the multiplication of two banded matrices, our multiplication actually seems to become a little more optimal than the original one as matrix size increases. Although our function seems more accurate on smaller matrices, and much less on larger ones in this graph 4.1, relative error remains very low so this is not really significant here.



(a)



(b)

Figure 4.1: (a) Computation time of the product of two banded $\mathcal{H}$-matrices and (b) Absolute error

When multiplying to unstructured $\mathcal{H}$-matrices (thus using the default cluster tree), the two functions remain equivalent in time, but the original one has an increased absolute error for larger matrix sizes. This is also found in the next tests, such as the cube of an $\mathcal{H}$-matrix 4.2 or the LU decomposition 4.4.

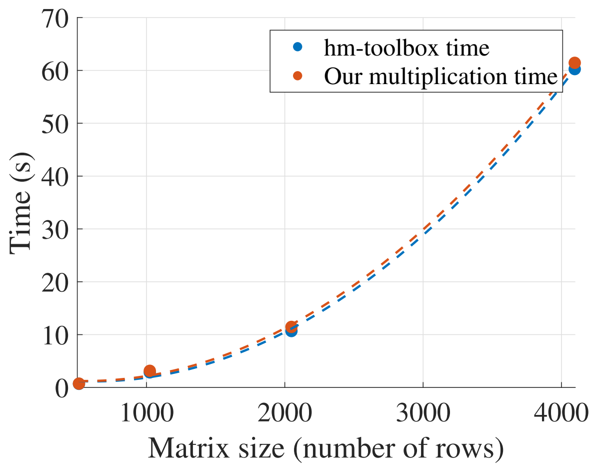


Figure 4.2: Computation time of the product of two unstructured $\mathcal{H}$-matrices

N (matrix size)	`hm-toolbox` error	our error
512	5.45×10^{-12}	5.90×10^{-12}
1024	1.16×10^{-11}	1.21×10^{-11}
2048	2.58×10^{-11}	2.75×10^{-11}
4096	2.21×10^{-4}	6.31×10^{-11}

Table 4.1: Absolute error of the product of two unstructured $\mathcal{H}$-matrices

When looking at the cube of an $\mathcal{H}$-matrix, while the gap between absolute errors is even more noticeable, our function is slightly slower, although computation times remain very close. This was to be expected, as block conversions and new ACA decompositions are needed in our multiplication when multiplying a leafnode block with a non-leafnode one. This is more common when doing several multiplications on entire $\mathcal{H}$-matrices, as there may be more block conversions or structural changes, which can be computationally expensive, but remain optimal in terms of memory.

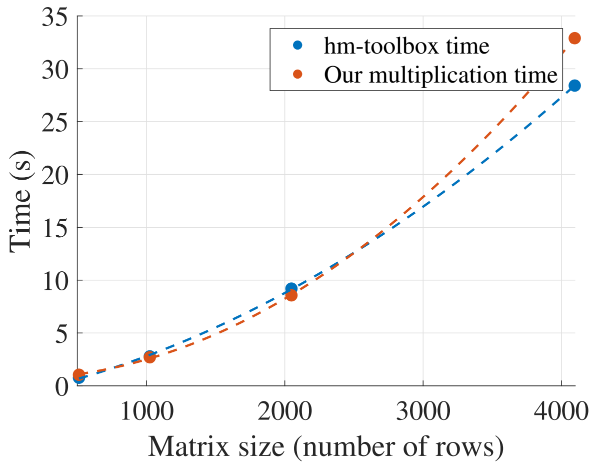


Figure 4.3: Computation time of the cube of an $\mathcal{H}$-matrix

N (matrix size)	`hm-toolbox` error	Our error
512	5.61×10^{-9}	5.23×10^{-9}
1024	2.27×10^{-8}	1.78×10^{-8}
2048	4.03	8.41×10^{-8}
4096	11.8	4.16×10^{-7}

Table 4.2: Absolute error of the cube of an $\mathcal{H}$-matrix

LU solvers are, once again, equivalent in terms of computation time, while the new multiplication makes the solver much more accurate, as shown in table 4.3.

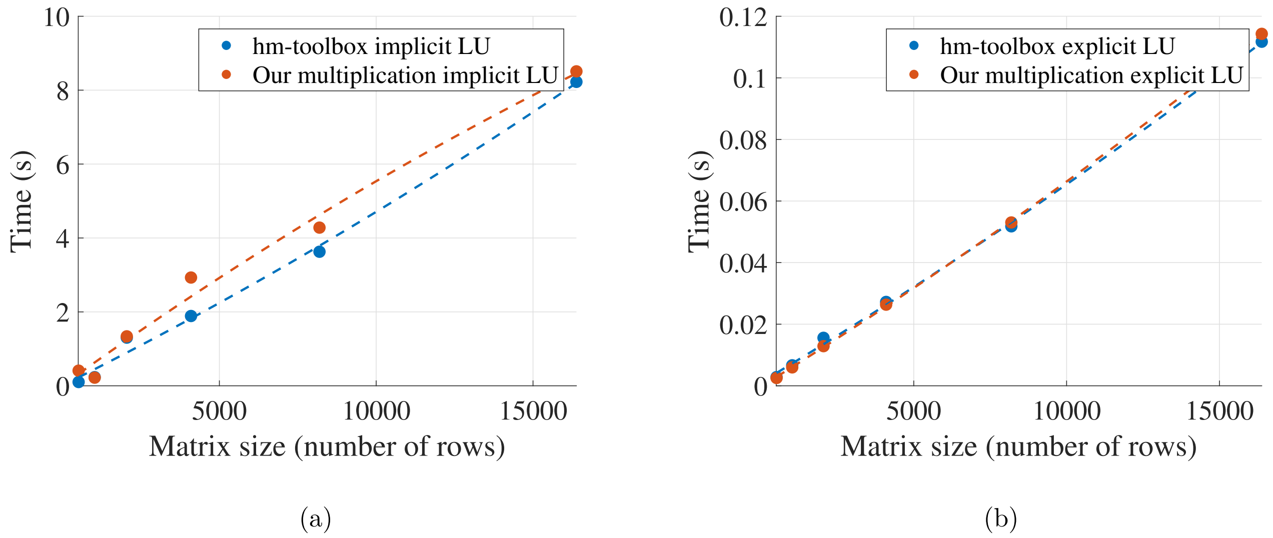


Figure 4.4: (a) Computation time of an implicit LU solver and (b) Computation time of an explicit LU solver

N (matrix size)	`hm-toolbox` error	Our error
512	5.24×10^{-10}	6.40×10^{-9}
1024	6.21×10^{-9}	3.69×10^{-9}
2048	1.24×10^{-2}	7.37×10^{-8}
4096	3.55×10^{3}	8.34×10^{-7}

Table 4.3: Absolute error of an explicit LU solver

4.3.2 Tests on simplified models for this project

Using all these new functions, a test was run on a matrix representing the electromagnetic interactions between points on a magnetic sphere with a coil 4.5, following the BEM-SIBC model [1], generated by `CreateSibcMatrices.m`. Two matrix sizes, $N = 540$ and $N = 1620$, were mainly used for these tests.

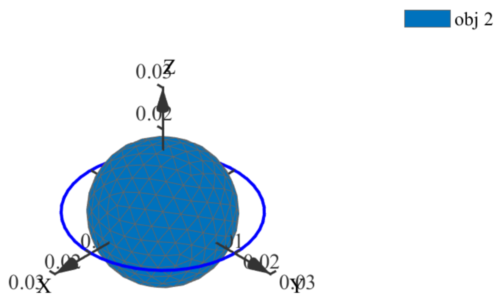


Figure 4.5: A meshed magnetic sphere with a coil represented by a blue circle

For the purpose of these tests, two re-orderings of the matrix entries were used, so as to find the optimal clustering :

- Reordering 1: The first one was based on creating an adjacency matrix of the cluster of points on the sphere, and applying reverse Cuthill-McKee (see appendixD) ordering to this adjacency matrix to minimize the band;
- Reordering 2: The second one was obtained by directly getting the cluster tree from our own library, `hmtxLib`, based on [35].

Then, we proceeded to turn our data into $\mathcal{H}$-matrices using another three different methods :

- Matrix type 1: One directly using `hmatrix(A)` to turn A into an $\mathcal{H}$-matrix, thus using default partitioning;
- Matrix type 2: Another using A as a handle to get the values from, following the chosen reordering;
- Matrix type 3: The last one still using a reordered A as a handle, but also giving the cluster tree from `hmtxLib` as an arbitrary partitioning. This one could thus only be used with the second reordering.

Here is a table of computation times for both versions of the multiplication, and also for `cluster_mask` applied on the new multiplication to have an output of the desired format.

N	Reordering	Matrix type	New multiplication	cluster_mask	Original multiplication
540	1	2	1.218047	0.197381	1.1911
540	1	1	1.009756	0.230722	1.4165
540	1	3	1.232777	0.2938	1.4414
540	2	2	0.923214	0.3073	1.2908
540	2	1	1.168926	0.119000	1.3187
1620	1	2	4.846048	0.402783	6.424122
1620	1	3	6.7908	1.181719	7.049392
1620	2	2	6.561993	0.851294	6.332305
1620	2	1	6.752903	1.170490	7.192651

Table 4.4: Computation time for different re-orderings and matrix constructions (in seconds)

According to table 4.4, the new multiplication is always slightly faster than the original one. It also prevents the issue that often happened with using a specific cluster tree with the original multiplication: mismatched block multiplications were transformed into full blocks, but they can now be admissible blocks, as explained above. With the addition of `cluster mask`, this guarantees that the structure of the matrix is always kept throughout the operations.

Regarding matrix storage (table 4.5), directly using our cluster tree and using it to fill the matrix is by far the most efficient method. The worst cases are always obtained by using the default cluster tree from the `hm-toolbox` library, no matter the re-ordering.

Reordering	Matrix Type	Bytes
1	3	11,441,133
1	2	13,103,253
2	2	19,368,549
1	1	25,408,669
None	1	25,408,669
2	1	25,408,669

Table 4.5: Size (in bytes) for different re-orderings and matrix constructions ($N = 1620$)

As the previous example was very specific, using a matrix composed of four very different blocks (2 full blocks, one bottom-left sparse block and one bottom-right diagonal block), some tests were then done on a simpler example. This new matrix represents the interactions between N charged points on a sphere, as shown in figure 4.6.

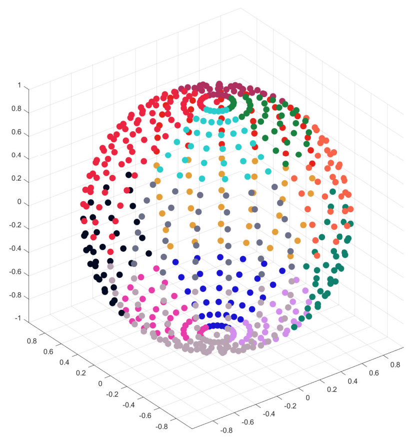


Figure 4.6: Charged points on a sphere for $N = 500$ (one color corresponds to one matrix block)

Both multiplication seemed pretty equivalent in terms of time, although, for larger matrices, the new one remained faster (figure 4.7). Moreover, regarding the accuracy, the new multiplication is, at all times, more accurate than the original one. The gap widens when the matrix gets larger, as shown by table 4.6.

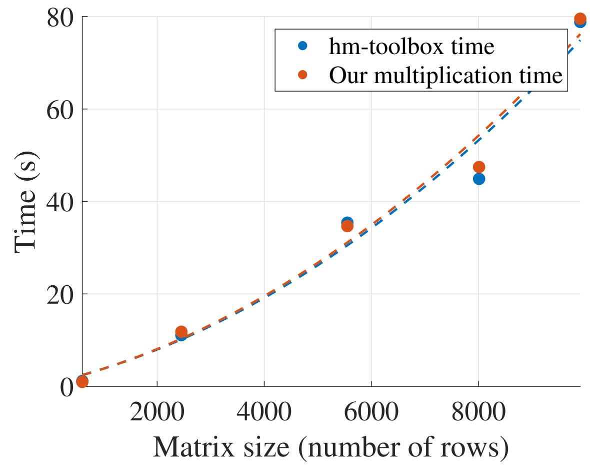


Figure 4.7: Computation time for different matrix sizes (in seconds) using our cluster tree

N	New multiplication	Original multiplication
602	0.5576515	0.4706100
2452	3.262714	3.491502
5552	4.752340	4.901622
8012	2.832364	3.416937

Table 4.6: $10^6\times$ relative error for different matrix sizes (with original tolerance of 1×10^{-6})

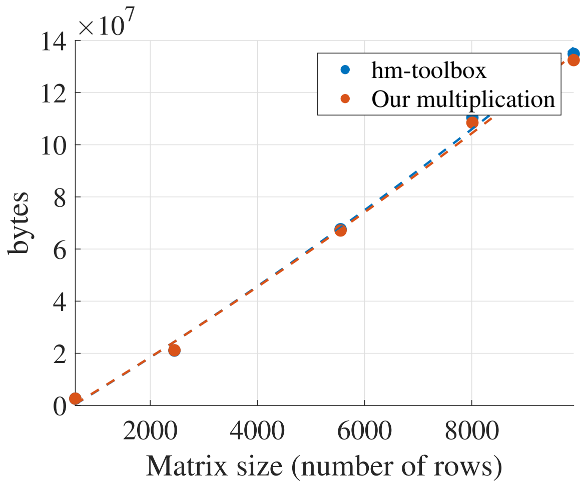


Figure 4.8: Memory occupation (in bytes) of multiplications of $\mathcal{H}$-matrices for different matrix sizes for the original multiplication and the new one

Memory occupation was equivalent for small matrices, but the newer code seemed to be more optimized for larger ones, as shown by figure 4.8. The test was also run on a 22352×22352 matrix, which took 0.378 GB, while running the code with the original multiplication caused the computer to run out of memory.

In conclusion, for larger matrices, which is the kind of matrices that truly need an $\mathcal{H}$-matrix partitioning, our additions to the library give faster, more accurate and storage-optimized results.

Chapter 5

Using $\mathcal{H}$-matrices for solving problems with hybrid BEM-SIBC formulation

The libraries previously introduced were then used to solve matrix systems in $\mathcal{H}$-matrix form, resulting from hybrid BEM-SIBC formulation of electromagnetic problems. While the ultimate goal is to use it on large complex systems like a car body, tests on the available computed were only run on smaller systems, such as several spheres and a coil of different magnetic-conductive and magnetic-non conductive materials, surrounded by air, as explained in section 1.4.

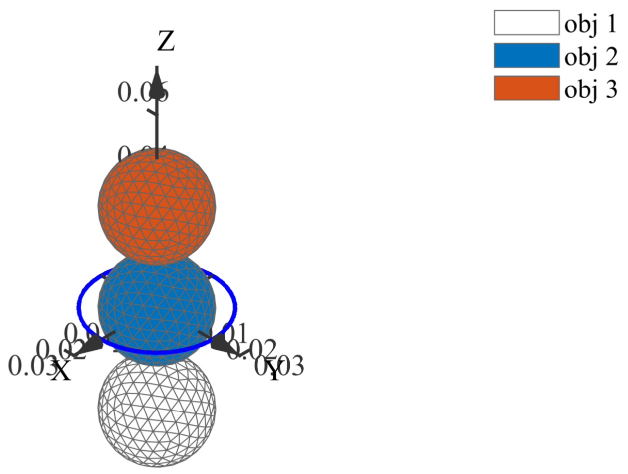


Figure 5.1: Example domain of study : obj 1, obj 2 and obj 3 are disconnected regions which can be made of the same or of different materials

5.1 Adapting $\mathcal{H}$-matrix cluster trees to fit with the geometry of the system

Using the equations from part 1.3, the resulting matrix equation accounts for electromagnetic equations in three regions :

1. The free air region, using standard BEM matrices H_0 and G_0 with one unknown for each face, using reduced scalar potential φ and its normal derivative, satisfying

$$H_0\varphi - G_0\partial_n\varphi = 0; \tag{5.1}$$

2. Each disconnected magnetic regions (only A and B in this case) giving rise to $B_1 = \begin{pmatrix} H_0^{AA} - I & \\ & H_0^{BB} - I \end{pmatrix}$,

$B_2 = \begin{pmatrix} G_0^{AA} & \\ & G_0^{BB} \end{pmatrix}$ with their sub-blocks directly coming from H_0 and G_0, and the relative magnetic permeability matrix $M_\mu = \begin{pmatrix} \mu_R^A I & \\ & \mu_R^B I \end{pmatrix}$. This satisfies the magnetic equation

$$B_1 \varphi^M + M_{mu}^{-1} B_2 \partial_n \varphi^M = M_\mu^{-1} B_2 H_{Sn}^M - B_1 h_{St}; \tag{5.2}$$

3. The magnetic-conductive regions using SIBC, treated as a single block, and verifying

$$CY^{-1}C^T \varphi^C + j\omega\mu_0 S \partial_n \varphi^C = CY^{-1} h_S + j\omega\mu_0 S H_{Sn}^C \tag{5.3}$$

with h_S, H_{Sn}^C known source terms, Y the sparse admittance matrix, C the face-to-edge connectivity matrix, S the face area diagonal matrix and $\omega = 2\pi f$, $\mu_0 = 4\pi \times 10^{-7}\,\mathrm{H\,m^{-1}}$, f being the frequency.

This gives rise to the final matrix equation

$$\left(\begin{array}{cc|cc} \multicolumn{2}{c|}{H_0} & \multicolumn{2}{c}{-G_0} \\ \hline B_1 & 0 & -M_\mu^{-1} B_2 & 0 \\ \hline 0 & CY^{-1}C^T & 0 & j\omega\mu_0 S \end{array}\right) \begin{pmatrix} \varphi \\ \partial_n \varphi \end{pmatrix} = \begin{pmatrix} 0 \\ -M_{mu}^{-1} B_2 H_{Sn} - B_1 h_{St} \\ CY^{-1} h_S + j\omega\mu_0 S H_{Sn} \end{pmatrix} \tag{5.4}$$

which can also be translated as, by separating magnetic-only and magnetic-conductive variables and expressing normal derivatives $\partial_n \varphi$ in terms of φ:

$$H_0 \varphi - G_0 \partial_n \varphi = 0 \tag{5.5}$$

$$A \begin{pmatrix} \varphi^M \\ \varphi \end{pmatrix} = b \tag{5.6}$$

with

$$A = \begin{pmatrix} H_0^M - G_0^M B_2^{-1} M_\mu B_1 & H_0^C - G_0^C (j\omega\mu_0 S)^{-1} CY^{-1} C^T \end{pmatrix}$$

and

$$b = \begin{pmatrix} G_0^M & G_0^C \end{pmatrix} \begin{pmatrix} H_{Sn}^M + B_2^{-1} M_\mu B_1 h_{St} \\ H_{Sn}^C + (j\omega\mu_0 S^{-1})^{-1} CY^{-1} h_S \end{pmatrix}.$$

As some matrix blocks are dense, other diagonal or even empty, the idea was to create a cluster tree for an $\mathcal{H}$-matrix which would keep the separation between those blocks. Each block would be an $\mathcal{H}$-matrix in itself. Inside blocks such as B_1 or B_2, the natural geometric division would also be preserved, in order to keep a diagonal bock matrix.

5.1.1 Using non-binary partitions

The first idea was to change the partitioning system, using a non-binary partition to have a first splitting corresponding to the number of disconnected magnetic regions, plus one block for conductive regions. This partitioning was implemented in the $\mathcal{H}$-matrix structure, but after taking a look at arithmetic operations, this first option was put aside. Indeed, adding may cases or parameters for each operation would have created a code far too long and complicated for a computing tool that was meant to be fast and memory-saving.

5.1.2 Binary splitting following disconnected regions

The other path that was explored was recursively following geometry to create a binary splitting for matrices such as B_1 or G_0. This led to having a few more off-diagonal blocks than needed for block diagonal matrices, but ended up being the most optimized choice. The routine takes as inputs all the classic inputs to create an $\mathcal{H}$-matrix in the `hm-toolbox` format, plus a list of indices at which there are geometry splits. It recursively splits a block in 4 sub-blocks at the geometry splitting point, until the end of the index list. Once everything has been split according to geometry, each sub-block created is then transformed into an $\mathcal{H}$-matrix block with the clustering from `hmtxLib`, as shown on G_0 in figure 5.2.

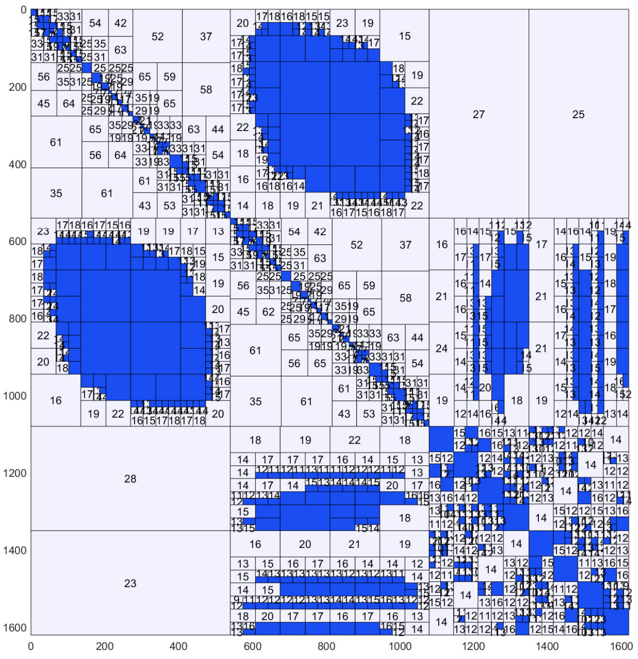


Figure 5.2: G_0 matrix structure plotted by `hm-toolbox`. Full blocks are filled with blue, while low-rank ones have their rank displayed inside.

5.2 Using the new cluster tree and $\mathcal{H}$-matrix algebra to solve the final equation

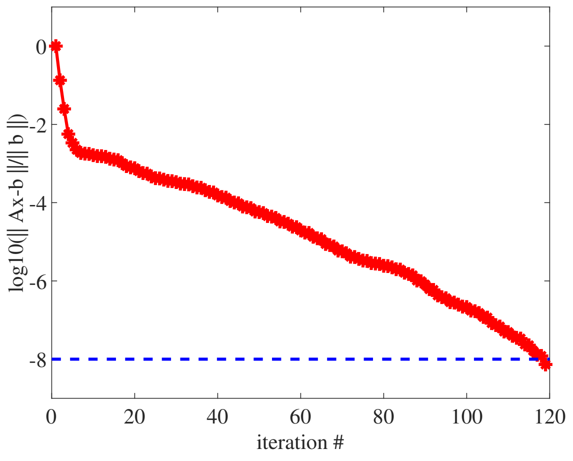
This new cluster tree was then used to convert matrices into $\mathcal{H}$-matrices, in the routine for solving the physics problem, which had been developed in earlier projects. B_1, B_2, G_0 and H_0 were thus saved as $\mathcal{H}$-matrices, with B_1 and G_0, and B_2 and H_0 having the same primary divisions in their cluster tree. The matrices from equation 5.6 were then created (only computing the LU decomposition of the admittance matrix Y is needed for the solver, not the full inversion, so this was not too expensive) using $\mathcal{H}$-matrix operations, and used in the solver, to give back a solution also as an $\mathcal{H}$-matrix.

5.3 Results

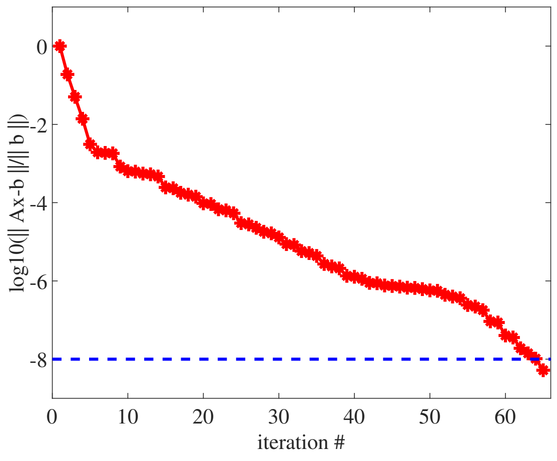
As shown by figure 5.3, the $\mathcal{H}$-matrix version of the solver actually required more iterations than the original one. Although it optimized storage, results also lacked accuracy, as seen in figure 5.4.

This can be explained by problems in the creation of the $\mathcal{H}$-matrices themselves. Indeed, even when creating diagonal $\mathcal{H}$-matrix blocks of B_1 with a tolerance of 1×10^{-6}, the relative error was of 0.0485. The reasons behind this huge reconstruction error are currently under study, as this problem happened with no previous tests. There is also currently no proper preconditioning, which needs to be incorporated if the system is to be solved iteratively.

However, if solving this error does not sufficiently reduce the number of iterations in GMRES, an approximate inversion of the matrix A by using a hierarchical LU decomposition with low accuracy can still serve as a preconditioner for an $\mathcal{H}$-matrix or even non $\mathcal{H}$-matrix solution.

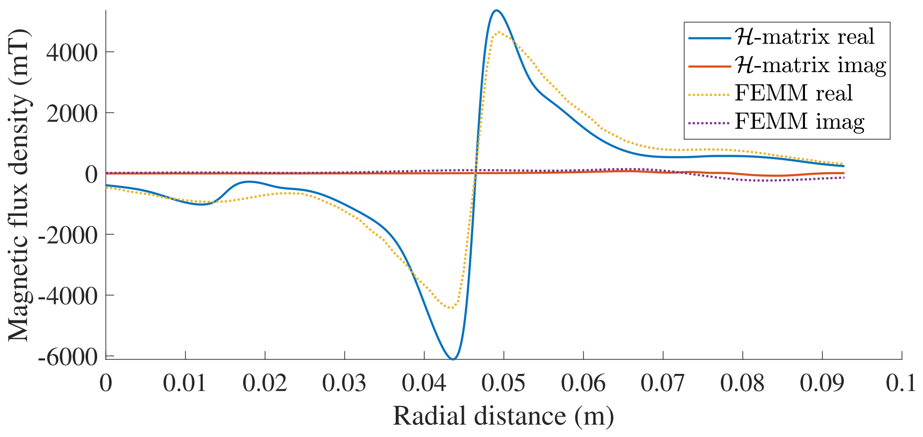


(a)



(b)

Figure 5.3: Number of iterations in GMRES using (a) $\mathcal{H}$-matrices (time = 7.88 seconds) and (b) without $\mathcal{H}$-matrices (time = 5.24 seconds)



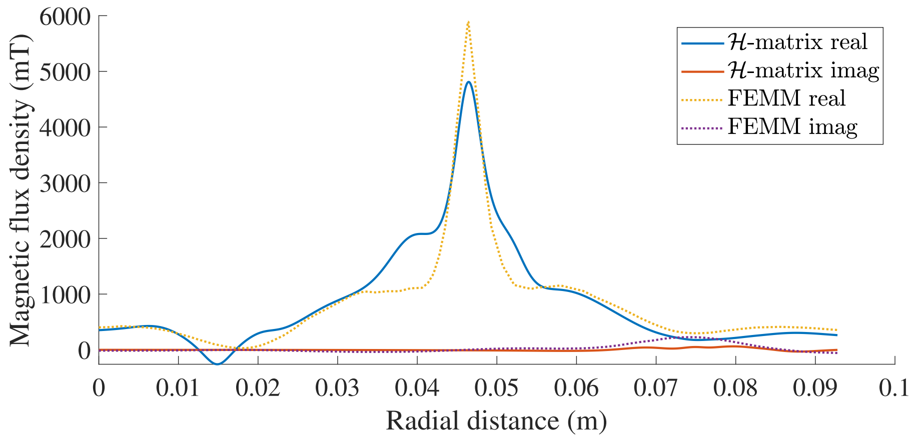


Figure 5.4: BEM-SIBC hybrid formulation with $\mathcal{H}$-matrices and FEMM on the x and z components of the magnetic flux density

Conclusion

In this report, different implementations of the $\mathcal{H}$-matrix representation have been implemented. The first one was meant to fully optimize storage and accuracy, creating blocks of various sizes and shapes, and shuffling matrix indexes, mostly leaving computational speed aside. The second library was originally very simple, and functioned by overloading MATLAB operators while keeping a fixed cluster tree, thus focusing more on speed than on the other aspects. The final library was an hybrid between the two, using the optimized cluster trees from the first one, while trying to keep fast and simple routines from the second one for algebraic operations. All methods have been tested in terms of efficiency and accuracy, on simple benchmarks, before applying any to the original physical problem.

The third version of the $\mathcal{H}$-matrix library was then modified once again to take into account the particular geometry of specific problems leading to BEM-SIBC formulations. The cluster tree structure was modified to keep sparse, dense or diagonal blocks separated, so as to keep blocks of zeros together, and to only have interactions in a single region or between two regions represented in each block. Resulting matrices from the BEM-SIBC formulation of the physical problem were then stored and used in $\mathcal{H}$-matrix format, and results were compared to the ones provided by the reference software, FEMM. As currently results are not optimal, changes in the $\mathcal{H}$-matrix format and proper preconditioning will be explored in the future.

Bibliography

[1] Fabio Freschi, Luca Giaccone, and Maurizio Repetto. Algebraic formulation of nonlinear surface impedance boundary condition coupled with BEM for unstructured meshes. *Engineering Analysis with Boundary Elements*, 88:104–114, March 2018.

[2] E.R. Dobbs. *Basic Electromagnetism.* Physics and Its Applications. Springer, Dordrecht, 1993.

[3] Grégoire Allaire. Approximation numérique et optimisation : Une introdution à la modélisation mathématique et à la simulation numérique, 2016.

[4] Enzo Tonti. *The Mathematical Structure of Classical and Relativistic Physics.* Springer New York, 2013.

[5] Piergiorgio Alotto, Fabio Freschi, Maurizio Repetto, and Carlo Rosso. *The Cell Method for Electrical Engineering and Multiphysics Problems.* Springer Berlin Heidelberg, 2013.

[6] Felix Klein. *Elementary Mathematics from a Higher Standpoint.* Springer Berlin Heidelberg, 2016.

[7] Discrete physics. `http://www.discretephysics.org/en/`.

[8] Zhuoxiang Ren. Habilitation à diriger des recheches : Contribution à la modélisation des systèmes électromagnétiques tridimensionnels. 1997.

[9] Rich Schwartz. The hodge star operator, course notes, 2015.

[10] Pierre Mounoud. Notes de cours : Formes différentielles. `https://www.math.u-bordeaux.fr/~pmounoud/tds/chap4.pdf`.

[11] S. Yuferev, N. Ida, and L. Kettunen. Invariant BEM-SIBC formulations for time- and frequency-domain eddy current problems. *IEEE Transactions on Magnetics*, 36(4):852–855, July 2000.

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

[13] J. Simkin and C.W. Trowbridge. Three-dimensional nonlinear electromagnetic field computations, using scalar potentials. *IEE Proceedings B Electric Power Applications*, 127(6):368, 1980.

[14] W.M. Rucker and K.R. Richter. Three-dimensional magnetostatic field calculation using boundary element method. *IEEE Transactions on Magnetics*, 24(1):23–26, 1988.

[15] F. Moser M. Schanz L. Gaul, M. Kögl. Boundary element methods for the dynamic analysis of elastic, viscoelastic, and piezoelectric solids. *Graz University of Technology*, 2017.

[16] A. Nicolet. Boundary elements and singular integrals in 3d magnetostatics. *Engineering Analysis with Boundary Elements*, 13(2):193–200, January 1994.

[17] R.D. Graglia. On the numerical integration of the linear shape functions times the 3-d green's function or its gradient on a plane triangle. *IEEE Transactions on Antennas and Propagation*, 41(10):1448–1455, 1993.

[18] D. Wilton, S. Rao, A. Glisson, D. Schaubert, O. Al-Bundak, and C. Butler. Potential integrals for uniform and linear source distributions on polygonal and polyhedral domains. *IEEE Transactions on Antennas and Propagation*, 32(3):276–281, March 1984.

[19] J. D. Jackson and Ronald F. Fox. Classical electrodynamics, 3rd ed. *American Journal of Physics*, 67(9):841–842, September 1999.

[20] Alain Bossavit. Computational electromagnetism and geometry: Building a finite-dimensional "maxwell's house. *J Japan Soc Appl Elctromagn Mech*, 7, 01 1999.

[21] G. Paoli, O. Biro, and G. Buchgraber. Complex representation in nonlinear time harmonic eddy current problems. *IEEE Transactions on Magnetics*, 34(5):2625–2628, 1998.

[22] Finite element method magnetics. http://www.femm.info/.

[23] Fabio Freschi, Luca Giaccone, and Maurizio Repetto. Educational value of the algebraic numerical methods in electromagnetism. *COMPEL - The international journal for computation and mathematics in electrical and electronic engineering*, 27(6):1343–1357, November 2008.

[24] M.K. Kim and I. Yun. An efficient implementation of the generalized minimum residual algorithm with a new preconditioner for the boundary element method. *Engineering Analysis with Boundary Elements*, 35(11):1214–1224, November 2011.

[25] Jan Mandel. On block diagonal and schur complement preconditioning. *Numerische Mathematik*, 58(1):79–93, December 1990.

[26] Jonathan Richard Shewchuk. What is a good linear finite element? - interpolation, conditioning, anisotropy, and quality measures. Technical report, In Proc. of the 11th International Meshing Roundtable, 2002.

[27] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 1997.

[28] Sergej Rjasanow and Olaf Steinbach. *The Fast Solution of Boundary Integral Equations*. Springer US, 2007.

[29] Ursula van Rienen, Michael Günther, and Dirk Hecht, editors. *Scientific Computing in Electrical Engineering*. Springer Berlin Heidelberg, 2001.

[30] Mario Bebendorf. *Hierarchical Matrices : A Means to Efficiently Solve Elliptic Boundary Value Problems*. Springer Berlin Heidelberg, 2008.

[31] Steffen Börm and Lars Grasedyck. Hybrid cross approximation of integral operators. *Numerische Mathematik*, 101(2):221–249, June 2005.

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

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

[34] M. Bebendorf and S. Rjasanow. Adaptive low-rank approximation of collocation matrices. *Computing*, 70(1):1–24, February 2003.

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

[36] Stefano Massei, Leonardo Robol, and Daniel Kressner. hm-toolbox: Matlab software for hodlr and hss matrices, 2019.

[37] G.M Del Corso, O. Menchi, and F. Romani. *Technical report : Krylov subspace methods for solving linear systems*. Università di Pisa.

[38] Jocelyne Ehrel. *Course notes : Résolution itérative de systèmes linéaires*. 2014.

[39] Vladimir Ivancevic and Tijana Ivancevic. Undergraduate lecture notes in De Rham-Hodge theory. 07 2008.

Appendix A

Glossary

- BEM: Boundary Element Method
- FEM: Finite Element Method
- SIBC: Surface Impedance Boundary Conditions
- ACA: Adaptive Cross Approximation
- GMRES: Generalized Minimal Residual Method
- SVD: Singular Value Decomposition
- FEMM: Finite Element Method Magnetics

Appendix B

Additional proofs

Lemma B.0.1. *For $k, \in \mathbb{N}, k \leq n$, $e(k,n)$ defined above satisfies $e(k,n) \leq \sqrt{k(n-k)+1}$.*

Proof. Let $k, \in \mathbb{N}, k \leq n$ If $U \in \mathbb{R}^{n\times k}$ is a matrix for which $\frac{1}{\max\limits_{P \text{ submatrix of } B} \sigma_{min}(P)} = e(k,n)$, one may assume without loss of generality that the submatrix $\hat{P}$ with a maximum $\det(P^H P)$ (later denoted as "volume of the submatrix") resides in the first k rows of U. Then, the submatrix of maximal volume in $\tilde{U} = U\hat{P}^{-1} = \begin{pmatrix} I_{k,k} \\ V \end{pmatrix}$ is located in the same k rows. If for any $1 \leq i \leq n, 1 \leq j \leq k$, $|\tilde{u}_{i,j}| > 1$, then by swapping the ith and jth rows, a submatrix whose volume is greater that 1 could be obtained. This contradicts the choice of $\hat{P}$. Thus :

$$\sigma_{min}^{-1}(\hat{P}) = \sigma_{max}(\tilde{U}) \leq \sqrt{\|I\|_2^2 + \|V\|_2^2} \leq \sqrt{k(n-k)+1}.$$

□

Lemma B.0.2 (Exact reproduction of rank k matrices)**.** *Let $n, m \in \mathbb{N}^*$. Let $A \in \mathbb{C}^{n\times m}$ be a matrix of rank exactly k. Then $R_k = \sum_{N=1}^{k} a^N (b^N)^H$ from algorithm 2 equals A.*

Proof. If for $k' \in \{0, .., k\}$, $A'_k = A - \sum_{N=1}^{k} a^N (b^N)^H$ is of rank $k - k'$, then for $k = k'$, A_k is of rank 0 so that $A = \sum_{N=1}^{k} a^N (b^N)^H$.
This first assumption will be proven by induction. First, if $k' = 0$, the result is trivial. Let us suppose now the assumption is true for $k' \in \{0, ..., k-1\}$. Let $V = Im(A_{k'})$, $dim(V) = k - k'$ and $W = V^\perp$ the $n - k + k'$-dimensional complement in $\mathbb{C}^{n\times m}$. Let $\tilde{V}$ and $\tilde{W}$ denote the corresponding spaces for $A_{k'+1} = A_{k'} - a^{k'}(b^{k'})^H$. By construction, $a^{k'+1} = A_{k'}(:, j_{k'+1})$ is a column of $A_{k'}$ thus $a^{k'+1} \in V$ and $V' \subset V$. Similarly, $b^{k'+1}$ and a certain column $i_{k'+1}$ of $A_{k'}$ are co-linear, thus $W' \subset W$. Then there holds, for $\delta = \max_{i,j} |A_{k'+1}(i,j)|$:

$$e_{i_{k'+1}}^H A_{k'} = (b^{k'+1})^H \delta \neq 0$$

$$e_{i_{k'+1}}^H (M_{k'} - a^{k'+1}(b^{k'+1})^H) = (b^{k'+1})^H \delta - e_{i_{k'+1}}^H a^{k'+1}(b^{k'+1})^H = 0$$

so that $e_{i_{k'+1}} \in W' \setminus W$, i.e. $dim(V') \leq dim(V) - 1$. Since $M - k' + 1$ is a rank one perturbation of $M_{k'}$, $rank(M_{k'+1}) = rank(M_{k'}) - 1 = k - k' - 1$. □

Lemma B.0.3 (Interpolation property)**.** *Let $A \in \mathbb{C}^{t\times s}$ be a matrix of rank at least $k \geq 1$ and R_k the cross approximation from algorithm 2. Then, for any row pivot index i^* and any column pivot index j^* there holds :*

$$R_k e_{j*} = A_{t\times\{j^*\}} \quad and \quad e_{i^*}^H R_k = A_{\{i^*\}\times s}$$

i.e., R_k exactly reproduces the pivot columns and rows of A.

Proof. For $k = 1, i, j \in t \times s$,

$$(R_1 e_{j^*})_i = a_i^1 b_{j^*}^1, \; (e_{i^*}^H R_1)_j = a_{i^*}^1 b_j^1 = A_{i^*,j}$$

so that for the remainder A_1 the row i^* and column j^* is zero. One proves by induction over $k' = 1, .., k$ that each $R_{k'}$ fulfills the statement for the first k' pivot elements. For $k' = 1$, the proof is the same as for $k = 1$. Let $k' \in \{1, .., k-1\}$ and suppose the statement true for k'. The matrix $A_{k'} = A - \sum_{N=1}^{k} a^N (b^N)^H$ has zero columns and rows for the first k' pivot columns and rows by construction and lemma B.0.2. For the $k'+1$-st column pivot element, denoted as j^*, there holds :

$$(R_{k'+1}e_{j^*})_i = (R_{k'})_{i,j^*} + a_i^{k'+1} b_{j^*}^{k'+1} (R_{k'})_{i,j^*} + (A_{k'})_{i,j^*} = A_{i,j^*}$$

and analogously for i^*. For any of the first k' pivot elements j^* we conclude
$(R_{k'+1}e_{j^*})_i = (R_{k'})_{i,j^*} + a_i^{k'+1} b_{j^*}^{k'+1} (R_{k'})_{i,j^*} + (A_{k'})_{i,j^*} = (R_{k'})_{i,j^*} + 0 = A_{i,j^*}$. □

Lemma B.0.4. *The cross approximation of a matrix $A \in \mathbb{C}^{t\times s}$ of rank at least $k \geq 0$ by the matrix R_k from 2 is of the form*

$$R_k := \sum_{N=1}^{k} a^N (b^N)^H = A|_{t\times \mathcal{P}_{cols}} (A|_{\mathcal{P}_{rows}\times\mathcal{P}_{cols}})^{-1} A|_{\mathcal{P}_{rows}\times s}$$

Proof. Let $\hat{R}_k = A|_{t\times \mathcal{P}_{cols}} (A|_{\mathcal{P}_{rows}\times\mathcal{P}_{cols}})^{-1} A|_{\mathcal{P}_{rows}\times s}$. By construction, both $R_k := \sum_{N=1}^{k} a^N (b^N)^H$ and $\hat{R}_k$ are of rank k. Let $j_l \in \mathcal{P}_{cols}$ be the l-th pivot column. Then for the j_l-th unit vectore_{j_l} there holds

$$\hat{R}_k e_{j_l} = A|_{t\times \mathcal{P}_{cols}} (A|_{\mathcal{P}_{rows}\times\mathcal{P}_{cols}})^{-1} A|_{\mathcal{P}_{rows}\times j_l} = A|_{t\times \mathcal{P}_{cols}} e_l = A|_{t\times\{j_l\}}$$

According to lemma B.0.3, both R_k and $\hat{R}_k$ are identical on the span of unit vectors $e_{j^*}, j^* \in \mathcal{P}_{cols}$. Since the image of them spans the whole image of R_k and $\hat{R}_k$, both matrices are identical. □

Lemma B.0.5. *The operator T_k maps a matrix $A \in \mathbb{C}^{\mathcal{I}\times\mathcal{J}}$ to a best approximation $\tilde{A} \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)$ with respect to the Frobenius norm :*

$$\left\|A - \tilde{A}\right\|_F = \min_{M' \in \mathcal{H}(\mathcal{T}_{\mathcal{I}\times\mathcal{J}}, k)} \|A - A'\|_F$$

Proof. Let $A \in \mathbb{C}^{\mathcal{I}\times\mathcal{J}}$, $\tilde{A} = T_k(A)$, and $\mathcal{P} \subset \mathcal{T}_{\mathcal{I}\times\mathcal{J}}$ the set of admissible leaves. It follows that

$$\begin{aligned}
\left\|A - \tilde{A}\right\|_F^2 &= \sum_{t\times s\in\mathcal{P}} (|A|_{t\times s} - \tilde{A}|_{t\times s})^2 + \sum_{t\times s\notin\mathcal{P}} (|A|_{t\times s} - \tilde{A}|_{t\times s})^2 \\
&= \sum_{t\times s\in\mathcal{P}} (|A|_{t\times s} - T_k(A|_{t\times s}))^2 + \sum_{t\times s\notin\mathcal{P}} (|A|_{t\times s} - A|_{t\times s})^2 \\
&= \sum_{t\times s\in\mathcal{P}} (|A|_{t\times s} - T_k(A|_{t\times s}))^2
\end{aligned}$$

Since, by definition, for any $t \times s$, $T_k(A|_{t\times s})$ is the best approximation of $A|_{t\times s}$in $\mathbb{C}_k^{t\times s}$, then $\tilde{A}$ is the best approximation of A with respect to the Frobenius norm. □

Appendix C

LU methods

An LU decomposition of a square matrix $A \in \mathbb{C}^{N\times N}$ is composed of a lower triangular matrix L and an upper triangular matrix U such that $A = LU$. An LU decomposition of the matrix A exists iff A is invertible and all its leading principal minors are nonzero. However, all square matrices have an LU decomposition with partial pivoting (PLU, such that $PA = LU$). Computing the LU decomposition is essentially a particular form of Gaussian elimination.

Given $A = (a_{i,j})_{1\leq i,j\leq N}$, let $A^{(0)} = A$. $A^{(1)}, ..., A^{(n)}, ..., A^{(N-1)}$ are then built iteratively, so that $A^{(n)}$ has zeros under the diagonal in the first n columns. Let $A^{(n-1)}$ be built. Considering its n-th column and its i-th row, the n-th column multiplied by $l_{i,n} = -\frac{a^{(n-1)}_{i,n}}{a^{(n-1)}_{n,n}}$ to the i-th row. For $n+1 \leq i \leq N$, this can be doe by multiplying $A^{(n-1)}$ to the left with the lower-triangular matrix

$$L_n = \begin{pmatrix} 1 & & & & 0 \\ & \ddots & & & \\ & & 1 & & \\ & & l_{n+1,n} & \ddots & \\ & & \vdots & & \ddots \\ 0 & & l_{N,n} & & 1 \end{pmatrix}.$$

This creates $A^{(n)} = L_n A^{(n-1)}$. After $N-1$ such steps, $A^{(N-1)}$ is upper triangular, and $L = L_1^{-1}...L_{N-1}^{-1}$ is lower triangular, with $A = L_1^{-1}L_1A^{(0)} = \cdots = L_1^{-1}\dots L_{N-1}^{-1}A^{(N-1)} = LA^{(N-1)}$.

This type of decomposition can be very useful for inverting matrices, or solving matrix systems such as $AX = B$ with$B \in \mathbb{C}^{N\times N}$, using the following steps :

1. Given A, compute its LU decomposition;
2. Now having $LUX = B$, let $Y = UX$ so that $LY = B$. Solve this triangular system for Y;
3. Solve the triangular system $UX = Y$ for X.

Appendix D

Reverse Cuthill Mc-Kee algorithm

Viewing the cluster of points as a graph in which vertices represent electromagnetic interactions, and modeled by a weighted adjacency matrix, the idea behind the Cuthill-McKee algorithm is to reduce the maximum distance between diagonal of the matrix and the non-zero element of the row that is farthest from the diagonal. The steps are the following :

1. Pick a first point, label it 0;
2. Until all points are labeled, find the point with the lowest index that still has unlabeled neighbors. Label its neighbors sequentially starting with smallest still available label.

This method tends to optimize the matrix if the labeling is reversed, which gives rise to the reverse Cuthill-McKee algorithm.

Appendix E

Generalized Minimal Residual Method (GMRES)

GMRES is an iterative Krylow-subspace method for solving nonsymmetric systems of linear equations. Let $A \in \mathbb{C}^{N\times N}$, be a non-singular matrix, $b \in \mathbb{C}^N$, and x_0 an initial guess for solving $Ax = b$.

Definition E.0.1 (Krylow subspace)**.** The Krylow subspace associated with A and r_0 is defined as

$$\mathcal{K}_k(A, r_0) = \text{span}\left\{r_0, Ar_0, A^2r_0, \ldots, A^{k-1}r_0\right\} = \{s_{k-1}(A)r_0, \text{ where } s_{k-1} \text{ is a polynomial of degree at most } k-1\}$$

where $r_0 = b - Ax_0$. Moreover, $\mathcal{K}_1 \subseteq \mathcal{K}_2 \subseteq \mathcal{K}_3...$, , and the dimension increases at most by one in each step, while never exceeding N. In fact, the dimension is bounded by the degree of r_0 with respect to A, i.e. the minimal degree ν of the polynomial s_{k-1} such that $s_{k-1}(A)r_0 = 0$. $\mathcal{K}_\nu$ is invariant and cannot be further enlarged, hence $\dim \mathcal{K}_k(A, r_0) = \min(k, \nu)$ [37].

The solution of a linear system has a natural representation as a member of a Krylov subspace, and if the dimension of this space is small, a well-approximated solution can be found in a few iterations, that is why this type of subspace is suitable for solving linear systems.

In GMRES, it is assumed that b is normalized. The method approximated the exact solution $Ax = b$ by the vector $x_k \in \mathcal{K}_k$ that minimizes the euclidean norm of the residual $r_k = Ax_k - b$. Because the vectors $r_0, Ar_0, ..., A^{k-1}r_0$ might be close to linerarly dependent, the Gram-Schmidt orthogonalization is used to find $w_1, ..., w_k$ which form a basis of $\mathcal{K}_k$. This orthogonalization applied to a Krylow subspace is called the Arnoldi method. These vectors and their coefficients are then stored in an upper Hessenberg matrix (this matrix has zero entries below the first subdiagonal).

The GMRES algorithm can then be described as follows :

1. Choose x_0 and a dimension k of the Krylow subspaces;

2. Arnoldi process :

$$\text{Compute } r_0 = b - Ax_0,\ \beta = \|r_0|_2,\ v-1 = \frac{r_0}{\beta}.$$
$$\text{For } 1 \leq j \leq k :$$
$$w = Av_j,$$
$$\text{For } 1 \leq i \leq j,\ h_{i,j} = (w,\ v_i),\ w = w - h_{i,j}v_i,$$
$$h_{j+1,1} = \|w\|_2,\ v_{j+1} = \frac{w}{h_{j+1,1}}.$$

   Define $V_k = [v_1, , ..., v_k]$, $H_k = (h_{i,j})_{i,j}$ the upper Hessenberg matrix.

3. Form the approximate solution: $x_k = x_0 + V_ky_k$ where $y_k = \text{argmin}_y \|\beta e_1 - H_my\|_2$, $e_1 = \begin{pmatrix} 1 \\ 0 \\ . \\ 0 \end{pmatrix}$.

4. If $\|r_k\|_2 = \|Ax_k - b\|_2$ is lesser than required precision, then stop. Otherwise, set x_0 to x_k and go to step 2.

Theorem E.0.1. *The GMRES method is characterized by*

$$\begin{aligned} x_k &\in x_0 + \mathcal{K}_k(A, r_0), \\ r_k &\perp A\mathcal{K}_k(A, r_0), \end{aligned}$$

which is equivalent to $\|r_k\|_2 = \min_{x \in x_0 + \mathcal{K}_k(A, r_0)} \|b - Ax\|_2$. *The equation* $Ax = b$ *has a unique solution, towards which the method monotonously converges in at most* N *iterations [38].*

Appendix F

Hodge star operator and differential forms

Definition F.0.1 (Differential forms)**.** Let $p \in \mathbb{R}^n$ and let $k \in \mathbb{N}^*$. A k-form at p on $\mathbb{R}^n$ is a function $\omega : \underbrace{\mathbb{R}^n_p \times \cdots \times R^n_p}_{k \text{ times}} \to \mathbb{R}$ which satisfies the following properties:

1. The map ω is multilinear, i.e.

$$w(\overrightarrow{a_1}, \dots, c\overrightarrow{a_i} + c'\overrightarrow{a_{i'}}, \dots, \overrightarrow{a_k}) = cw(\overrightarrow{a_1}, \dots, \overrightarrow{a_i}, \dots, \overrightarrow{a_k}) + c'w(\overrightarrow{a_1}, \dots, \overrightarrow{a_{i'}}, \dots, \overrightarrow{a_k})$$

   for every $a \leq ik$, for all $\overrightarrow{a_1}, \dots, \overrightarrow{a_i}, \overrightarrow{a_{i'}}, \dots, \overrightarrow{a_k} \in \mathbb{R}^n_p$ and all $c, c' \in \mathbb{R}$.

2. The map ω is anti-symmetric, i.e.:

$$(\overrightarrow{a_1}, \dots, \overrightarrow{a_i}, \dots, \overrightarrow{a_j}, \overrightarrow{a_k}) = -(\overrightarrow{a_1}, \dots, \overrightarrow{a_j}, \dots, \overrightarrow{a_i}, \overrightarrow{a_k})$$

   for all $A \leq i < j \leq k$ and all $\overrightarrow{a_1}, \dots, \overrightarrow{a_k} \in \mathbb{R}^n_p$.

A 0-form at p is any number $r \in \mathbb{R}$. For $k \geq 0$, the set of all k-forms at p on $\mathbb{R}^n$ is denoted by $\Omega^k_p\mathbb{R}^n$.

Definition F.0.2 (Coefficients of a k-form)**.** Ley $\omega \in \Omega^k_p\mathbb{R}^n$, where $k \geq 1$. For $A \leq i_1, \dots, i_k leqn$ put

$$w_{i_1,\dots,i_k} = w((e_{i_1})_p, \dots, (e_{i_k})_p).$$

$w_{i_1,\dots,i_k}$ are called the coefficients of ω.

Definition F.0.3 (Wedge-product)**.** Let $r, s \geq 1$, $\alpha \in \Omega^r_p\mathbb{R}^n$ and $\beta \in \Omega^s_p\mathbb{R}^n$. The wedge product is defined by the $(r+s)$ form at p :

$$\alpha \wedge \beta : (\mathbb{R}^n_p)^{r+s} \to \mathbb{R} \tag{F.1}$$

where

$$(\alpha \wedge \beta)(\overrightarrow{a_1}, \dots, \overrightarrow{a_{r+s}}) = \sum \delta^{i_1,\dots,i_{r+s}}_{1,2,\dots,r+s} \alpha(\overrightarrow{a_{i_1}}, \dots, \overrightarrow{a_{i_r}}) \beta(\overrightarrow{a_{i_{r+1}}}, \dots, \overrightarrow{a_{i_{r+s}}}) \tag{F.2}$$

where the summation is taken over all rearrangements $i_1, \dots, i_{r+s}$ of $1, 2 \dots, r+s$.

Definition F.0.4 (Hodge star operator)**.** The Hodge star operator $\star : \Omega_p\mathbb{R}^n \to \Omega_{(n-p)}\mathbb{R}^n$ maps any p-form α into its dual $(n-p)$-form $\star\alpha$ on a smooth n-manifold ($\mathbb{R}^n$ here). It is defined as:

$$\begin{gathered}
\wedge \star \beta = \beta \wedge \star\alpha = < \alpha, \beta > \mu, \text{for } \alpha, \beta \in \Omega_p\mathbb{R}^n, \\
\star \star \alpha = (-1)^{p(n-p)}\alpha, \\
\star(c_1\alpha + c_2\beta) = c_1(\star\alpha) + c_2(\star\beta), c_1, c_2 \in \mathbb{R}, \\
\wedge \star \alpha = 0 \implies \alpha \equiv 0.
\end{gathered}$$

The operator depends on the metric on $\mathbb{R}^n$ and also on the orientation [39].