Université de Liège
Faculté des Sciences Appliquées
Collège de Doctorat en Électricité, Électronique et Informatique

Efficient methods for large-scale time-harmonic wave simulations

Nicolas Marsic
Ingénieur civil électricien

Doctoral dissertation presented
in fulfillment of the requirements for the degree of
Docteur en Sciences de l'Ingénieur

January 2016

Université de Liège
Faculté des Sciences Appliquées
Collège de Doctorat en Électricité, Électronique et Informatique

Efficient methods for large-scale time-harmonic wave simulations

Nicolas Marsic
Ingénieur civil électricien

Doctoral dissertation presented
in fulfillment of the requirements for the degree of
Docteur en Sciences de l'Ingénieur

January 2016

Thesis committee:

Prof. Eric Béchet (Université de Liège)
Prof. Markus Clemens (Bergische Universität Wuppertal)
Prof. Jean-Jacques Embrechts (Université de Liège), Président
Prof. Christophe Geuzaine (Université de Liège), Promoteur
Dr. Koen Hillewaert (Cenaero)
Prof. André Nicolet (Aix-Marseille Université)
Prof. Ruth Sabariego (Katholieke Universiteit Leuven)

Version history

October 2015: – version submitted to the committee

January 2016:
- new appendix A
- new section 5.5
- new example for pollution effect
- minor corrections

Abstract

There is a growing consensus that state of the art low-order finite element technology requires, and will continue to require, too extensive computational resources to provide the necessary resolution for complex simulations, even at the rate of computational power increase. The requirement for precise resolution naturally leads to consider methods, with a higher-order of grid convergence than the classical second-order provided by most industrial grade codes. In particular, for high-frequency time-harmonic wave simulations, high-order schemes allow to efficiently resolve the rapid small scale spatial oscillations of the solution, and allow to alleviate the pollution effect.

Whitney elements are extensively used to model electromagnetic wave problems, and several high-order extensions have been proposed in the literature. As for standard Lagrange elements, though, the computational cost of solving the linear system of equations rapidly becomes overshadowed by the assembly time of the system itself, as the order of the basis functions increases.

The first objective of this thesis is to solve this problem, by reformulating the assembly algorithm into a computationally more efficient procedure. Afterward, this newly developed higher-order approach is tested on different simulations requiring a high precision.

Moreover, the application of the finite element method on these high-frequency problems leads to very large, complex and possibly indefinite linear systems. Unfortunately, direct sparse solvers do not scale well for solving such large systems, and Krylov subspace iterative solvers can exhibit slow convergence or even diverge. Domain decomposition methods provide an elegant alternative to these previous techniques, by iterating between sub-problems of smaller sizes, amenable to sparse direct solvers.

In this thesis, the emphasis is placed on a particular family of domain decomposition method: the optimized Schwarz algorithm. The second objective of this work is to analyze the performances of this technique, when higher-order finite elements are used.

Résumé

Un consensus croissant s'accorde sur le fait que la technologie éléments finis classique nécessite, et nécessitera toujours, des ressources en calcul trop importantes, pour offrir la précision nécessaire à la résolution de simulations complexes, même au rythme auquel la puissance de calcul croît. La nécessité de solutions précises conduit naturellement à considérer des méthodes avec un ordre de convergence plus élevé que le classique second ordre proposé par la plupart des codes industriels. En particulier, dans le cas des simulations d'onde haute fréquence en temps harmonique, les schémas d'ordre supérieur permettent de résoudre efficacement les oscillations rapides des petites échelles, et ils réduisent également la pollution numérique.

Les éléments de Whitney sont largement utilisés dans le cadre des simulations d'onde électromagnétique, et plusieurs extensions d'ordre élevé ont été proposées dans la littérature. Cependant, comme dans le cas classique des éléments de Lagrange, lorsque l'ordre des fonctions de base est augmenté, le temps d'assemblage du système linéaire devient prépondérant par rapport au coût nécessaire à la résolution de ce même système.

Le premier objectif de cette thèse est de résoudre ce problème, en reformulant l'algorithme d'assemblage en une procédure exploitant plus efficacement les ressources de calcul. Ensuite, cette nouvelle approche d'ordre élevé est testée sur différentes simulations nécessitant une haute précision.

En outre, l'utilisation de la méthode des éléments finis sur ces problèmes haute fréquence conduit à des systèmes linéaires de grande taille, complexes, et potentiellement indéfinis. Malheureusement, les solveurs directs ne passent pas à l'échelle lorsqu'il s'agit de résoudre ces systèmes immenses. De plus, les solveurs itératifs à sous espace de Krylov manifestent une convergence lente, voir même une divergence. Les méthodes de décomposition de domaine proposent une alternative élégante aux approches précédentes, en itérant entre des sous problèmes de plus petite taille, traitables par les techniques directes.

Dans cette thèse, l'accent est mis sur une famille particulière de décomposition de domaine: l'algorithme optimisé de Schwarz. Le second objectif de ce travail est d'analyser les performances de cette technique, lorsque des éléments finis d'ordre élevé sont utilisés.

Remerciements

C'est avec un réel plaisir, que je tiens à remercier toutes les personnes sans qui ce travail n'aurait jamais vu le jour. Tout d'abord, je désire remercier mon promoteur, Christophe Geuzaine, pour m'avoir lancé dans cette aventure voici quatre ans. Ton enthousiasme et tes excellents conseils m'ont permis d'arriver au bout de cette thèse!

Je remercie également André Nicolet et Guillaume Demésy, pour m'avoir accueilli quelque temps à Marseille, et pour m'avoir soumis les cas tests de la cavité de Fabry-Pérot et du guide d'onde segmenté. Sans vos contributions, ma thèse aurait sans doute été moins sexy! En plus du sud de la France, ma thèse m'a permis de séjourner quelques mois outre Atlantique et plus précisément à Columbus, Ohio. Je tiens à remercier le professeur Jin-Fa Lee et Caleb Waltz pour m'avoir accueilli et pour leurs conseils.

Je tiens également à remercier les membres de mon jury, les professeurs Eric Béchet, Markus Clemens, Jean-Jacques Embrechts, Koen Hillewaert, André Nicolet et Ruth Sabariego pour avoir lu et commenté mon travail. Vos remarques m'ont permis d'améliorer le manuscrit final.

J'ai eu la chance de bénéficier d'une bourse du Fonds pour la formation à la Recherche dans l'Industrie et dans l'Agriculture (FRIA), et je tiens également à le remercier pour sa confiance. De plus, qui dit calcul haute performance, dit infrastructures de calcul. A cet égard, je remercie le Consortium des Équipements de Calcul Intensif (CÉCI), le Fonds de la Recherche Scientifique (F.R.S–FNRS), la Fédération Wallonie-Bruxelles, ainsi que la Région wallonne pour les moyens de calcul mis à disposition.

Ces remerciements resteraient évidemment incomplets si j'omettais l'excellente ambiance au sein du groupe Applied and Computational Electromagnetics. Je tiens à remercier tous mes collègues, présents et passés. Merci à Alexandre (V. et H.), Amaury, Axel, Bertrand, Christophe, David, Erin, Frédéric, Innocent, Isabel, Jean, Jean de Dieu, Kevin, Maxime (S. et G.), Patrick, Pierre, Ruth, Simon, Steven, Valera, Véronique, Vincent, Vuong et Yannick. Plus particulièrement, je tiens à remercier les habitants du bureau 1.154: Amaury pour l'avoir partagé avec moi pendant quatre ans, ainsi que Vincent pour avoir contribué à mon initiation à awesome! Pour terminer avec les résidents de mon bureau, je remercie mon canard pour son aide au debugging.

Je tiens également à remercier chaleureusement ma famille, ma belle famille ainsi que mes amis pour m'avoir supporté durant toutes ces années. Plus particulièrement, merci à Amandine, Lise et Sylvie pour nos traditionnels temps de midi! Bon courage à vous pour la suite de vos thèses: finalement, c'est possible d'en arriver à bout. Finalement, il me reste sans doute la personne la plus importante à remercier, et je tiens à lui dédier ces dernières lignes. Merci à toi Amandine pour toute l'affection, les encouragements et le soutien que tu m'as apporté. Voilà, l'aventure de la thèse se termine pour moi, et est presque finie pour toi. J'ai hâte de découvrir ce que l'avenir nous réserve. Mais ça c'est une autre histoire...

Nicolas Marsic,
Janvier 2016.

Contents

Introduction

A long time ago in a galaxy far, far away....
— G. Lucas, *Star Wars.*

Since the last few decades, computer simulations have gained popularity in many scientific disciplines, such as, for instance, fluid mechanics [66], solid mechanics [138], electromagnetism [19], chemistry [85] or biology [13]. The design and the analysis of such numerical tools falls into the scope of scientific computing, which is basically the place where physics, mathematics and computer science intersect. Obviously, before designing a new numerical tool, it is first needed to understand the considered physical phenomenon, and to convert it into a mathematical model. Then, this model has to be solved, which means that an appropriate numerical procedure has to be selected. Finally, this procedure has to be implemented and run on an actual computer. However, quite often, a single computer is insufficient: it is then essential to design efficient methods, that distribute the work across many processing units.

In this thesis, we focus particularly on tools to simulate the behavior of waves. For instance, in the field of telecommunications, engineers are interested on how an electromagnetic wave is radiated by an antenna, to optimize its shape under a given set of constrains. Another example can be found in optics: to develop new meta-materials, scientists need to understand, how light propagates through a lattice made of different components. These first two examples involve electromagnetic waves. However, other kinds of waves can be found in nature: the most famous being probably sound, which is nothing but a pressure wave. In this context, for instance, car designers are using computer simulations to understand, how the noise coming from an engine propagates into the car interior.

To accurately predict how a wave propagates, the finite element method can be used [97]. This numerical tool is a technique to solve partial differential equations, arising from the mathematical representation of the laws of physics. This approach allows an easy treatment of complex geometries and non-homogeneous media. Basically, the idea of the method, is to subdivide the physical domain into a collection of simple geometrical shapes (*e.g.*, triangles, cubes, ...), over which the solution is approximated by polynomials. The original partial differential equation is then converted into an algebraic system of equations.

From a numerical point of view, the treatment of waves often leads to very large problems. Indeed, usually, the signal wavelength is many order of magnitude smaller that the whole computation domain. To illustrate this, let us consider a WiFi antenna. It is located in a building, and we want to compute the power of the signal in every room. To accurately solve this apparently simple problem, we have to consider a phenomenon at a scale of a few

centimeters (the WiFi signal wavelength), into a domain of a few meters for instance. This large difference in scales leads to huge algebraic systems to solve.

To better control the size of these simulations, the high-order finite element method can be used [131]. Instead of classically used piecewise linear functions, the solution is approximated by higher-order polynomials, leading to a better approximation of the small-scale oscillations of the solution. For a given accuracy, this approach helps to reduce the size of the resulting algebraic system.

Another strategy is to divide the computational domain into sub-domains, defining thus a set of sub-problems of smaller size. Then, by iterating between these sub-problems, the solution of the original problem can be recovered. Instead of reducing the size of the algebraic system to solve, this approach splits it into smaller ones. Thus, by using more than one computer, large simulations can be carried out. Among the existing domain decomposition methods, optimized Schwarz algorithms are well suited for wave simulations [51].

Scope and goals of this work

This work contributes to the development of an efficient implementation of the high-order finite element method, for the accurate treatment of time-harmonic wave simulations. In addition, to handle large-scale problems, optimized Schwarz algorithms are developed, and coupled to the high-order finite element method. For this purpose, four objectives have been identified.

1. *Efficient assembly of finite element matrices.*
   While higher-order techniques can significantly increase the accuracy of a simulation, they also lead to an important rise in the time needed to assemble the algebraic system: thus the necessity for efficient implementations. A solution was already proposed for discontinuous Galerkin methods based on Lagrange elements [64, 65, 83]. In this work, we propose an extension to vector-valued problems, and to non-nodal methods.

2. *Automatic orientation of non-Lagrange bases.*
   In the finite element method, the functions used to approximate the solution are taken from a basis, that has to be constructed. When handling vector-valued functions, or high-order scalar functions, a notion of orientation must be introduced in this construction. Usually, it is introduced, on-the-fly, during the finite element matrix assembly. However, if the efficient technique is used, this orientation must be taken into account before the assembly. The solution used in [64, 65, 83], for orienting high-order Lagrange basis functions, cannot be extended to more general bases. In this thesis, a general method for orienting non-Lagrange bases is investigated.

3. *Analysis of the optimized Schwarz algorithm with the high-order finite element method.*
   As already mentioned, the algebraic systems, obtained by applying the finite element method on high-frequency wave problems, are usually very large. Because of their size, direct solvers do not scale: the bottleneck being often the memory. On the other hand, iterative methods do not converge well for high-frequency wave problems [48, 49]. In order to handle these huge systems, domain decomposition methods are promising candidates. In this work, we focus on the optimized Schwarz algorithm, and we study its behavior, when coupled with the high-order finite element method.

4. *Tests on large-scale problems.*
   Finally, all the previous developments are tested on various large-scale problems. Cases with and without the domain decomposition are considered.

Outline

This thesis is divided in seven chapters as follows.

In chapter 1, we start by a review of the governing equations of electromagnetic and acoustic time-harmonic waves. We then introduce some classical techniques to solve these problems numerically. Afterward, we motivate the use of the high-order finite element method, as well as the use of a domain decomposition method. Finally, we conclude by an introduction on modern computing architectures.

In chapter 2, the key concepts of the finite element method are reviewed, and weak formulations are introduced for electromagnetic and acoustic waves. We conclude by presenting the classical orientation and finite element assembly approaches.

In chapter 3, an efficient finite element assembly technique, relying on an efficient implementation of numerical quadratures, is presented. Various tests are carried out on scalar and vector cases, to compare the performance of the classical and the new assembly procedures.

In chapter 4, an automatic orientation procedure, exploiting a dictionary structure, is developed. This new procedure is compatible with the efficient assembly technique, and replaces then the classical on-the-fly approach.

In chapter 5, all the above developed tools are tested by simulating an open resonator. The objective is to compute the resonant frequency of a particular mode, as well as its damping time. Since the cavity exhibits a very high quality factor, recovering the damping time requires high precision calculations, which are offered by the high-order finite element method. During the process, an **LU** decomposition must be performed, thus limiting the simulation sizes to roughly 10 million unknowns.

In chapter 6, the optimized Schwarz algorithm is reviewed. After presenting the method, optimized transmission operators are motivated, and presented for both acoustic and electromagnetic wave problems. By using numerical experiments, we also analyze the behavior of the Schwarz algorithm, when coupled with the high-order finite element method. The solution accuracy, as well as the iteration count, are studied.

In chapter 7, our implementation is tested to simulate the propagation of an electromagnetic wave into a large-scale segmented waveguide. Thanks to the domain decomposition, the memory scaling limits, encountered with direct solvers, are alleviated. Problems with up to 50 million unknowns are considered.

Original contributions and communications

In the following, we present the contributions that are, to the best of our knowledge, original.

1. The extension of the efficient numerical quadrature implementation, proposed in [64, 65, 83], to the vector-valued case in chapter 3.
2. The automatic orientation procedure developed in chapter 4.
3. The convergence analysis of the damping time for the cavity considered is chapter 5 (see [30] for an unsuccessfully previous attempt).
4. The implementation of optimized Schwarz algorithms, with high-order finite element discretizations, and the related performance analysis proposed in chapters 6 and 7.

It is worth mentioning that all these developments were implemented from scratch, and available in the form of a C++ library on the subversion tree[1]:

`https://onelab.info/svn/gmsh/trunk`,

in the directory `./projects/small_fem/`. In order to handle the non finite element features, the developed code relies on the following third party libraries:

1. the OpenBLAS[2] [134] (version `0.2.13`) implementation of the BLAS (Basic Linear Algebra Subprograms) interface [18, 38, 39, 84], used to handle the elementary algebraic operations;
2. the direct solver MUMPS[3] [3, 4] (version `5.0.0`), used to compute the solution of the generated finite element linear systems;
3. the GMRES implementation of the PETSc library[4] [11, 12] (version `3.6.0`), used for the Krylov acceleration of the implemented Schwarz algorithm;
4. the SLEPc library[5] [62, 107] (version `3.6.0`), used to solve eigenvalue problems;
5. the Gmsh library[6] [54], used to handle the mesh related features.

Parts of this thesis have been published in the following journal papers:

1. N. Marsic and C. Geuzaine, "Efficient finite element assembly of high order Whitney forms," *IET Science, Measurement & Technology*, vol. 9, pp. 204–210, 2 2015.
2. N. Marsic, C. Waltz, J.-F. Lee, and C. Geuzaine, "Domain decomposition methods for time-harmonic electromagnetic waves with high order Whitney forms," *IEEE Transactions on Magnetics*, in press.
3. B. Thierry, A. Vion, S. Tournier, M. El Bouajaji, D. Colignon, N. Marsic, X. Antoine, and C. Geuzaine, "GetDDM: An open framework for testing optimized Schwarz methods for time-harmonic wave problems," *Computer Physics Communications*, submitted.

[1] Login: gmsh; password: gmsh.
[2] Available at: `http://www.openblas.net`.
[3] Available at: `http://mumps-solver.org`.
[4] Available at: `http://www.mcs.anl.gov/petsc`.
[5] Available at: `http://slepc.upv.es`.
[6] Available at: `http://gmsh.info`.

Moreover, this work has been presented at the following conferences: EMF2013 (Belgium), CEM2014 (United Kingdom), CEFC2014 (France), ACOMEN2014 (Belgium), COMPUMAG2015 (Canada), DD23 (Republic of Korea).

Chapter 1

Time-harmonic wave simulations: description, numerical techniques and difficulties

In this first chapter, time-harmonic wave simulations are introduced. We first start by reviewing the governing equations of electromagnetic and acoustic waves. Afterward, classical numerical techniques for solving such equations are presented and compared. This chapter continues by describing the difficulties encountered when solving high-frequency wave problems. Then, a short motivation of high-order finite element techniques and domain decomposition methods is presented. Finally, we conclude by an introduction on modern computing architectures.

1.1 From Maxwell's equations to an electromagnetic wave

1.1.1 Generalities

It is well known that electromagnetic phenomena are ruled by Maxwell's equations [9]. Using the quantities defined in Table 1.1, these equations can be written in the following form[1]:

$$\begin{cases} \mathbf{curl}\, \boldsymbol{E} = -\dfrac{\partial \boldsymbol{B}}{\partial t}, & \text{(1.1a)} \\ \mathbf{curl}\, \boldsymbol{H} = \dfrac{\partial \boldsymbol{D}}{\partial t} + \boldsymbol{J}, & \text{(1.1b)} \\ \operatorname{div} \boldsymbol{D} = \varrho, & \text{(1.1c)} \\ \operatorname{div} \boldsymbol{B} = \mathbf{0}. & \text{(1.1d)} \end{cases}$$

Equations (1.1a) and (1.1b) are known as Faraday's and Ampère's laws. And equations (1.1c) and (1.1d) are the electric and magnetic Gauß' laws.

The system of equations (1.1) has to be augmented by constitutive laws, describing how matter responds when submitted to an electrical or magnetic field. To begin with, the elec-

[1] The modern formulation of Maxwell's equations, using vector calculus formalism, is attributed to Heaviside [9].

Quantity	Symbol	Unit
Electric field	$\boldsymbol{E}$	V/m
Magnetic field	$\boldsymbol{H}$	A/m
Electric displacement	$\boldsymbol{D}$	C/m^2
Magnetic induction	$\boldsymbol{B}$	T
Electric charge density	ϱ	C/m^3
Electric current density	$\boldsymbol{J}$	A/m^2

Table 1.1: Electromagnetic quantities.

tric displacement and the magnetic induction are related to the electric and magnetic fields through the following relations:

$$\begin{cases} \boldsymbol{D} = \boldsymbol{\varepsilon}\, \boldsymbol{E}, & \text{(1.2a)} \\ \boldsymbol{B} = \boldsymbol{\mu}\, \boldsymbol{H}, & \text{(1.2b)} \end{cases}$$

where $\boldsymbol{\varepsilon}$ is the electric permittivity of the considered material expressed in [F/m], and $\boldsymbol{\mu}$ is the magnetic permeability expressed in [H/m]. Generally speaking, since the material properties can be anisotropic and inhomogeneous, these two quantities are tensor functions of space. Moreover, they can also be functions of time and of the electromagnetic field itself. However, in the scope of this thesis, the assumption is made that these quantities depend only on the space coordinates. In the case of isotropic materials, they become scalar functions: ε and μ. Furthermore, if the material is homogeneous and isotropic, these quantities are then simple scalar constant. Finally, $\boldsymbol{\varepsilon}$ and $\boldsymbol{\mu}$ are traditionally expressed as fractions of the vacuum permittivity ε_0 and vacuum permeability μ_0, which are universal constants:

$$\begin{cases} c_0 &= 299792458 & \text{[m/s]}, \\ \mu_0 &= 4\pi \times 10^{-7} & \text{[H/m]}, \\ \varepsilon_0 &= \dfrac{1}{c_0^2 \mu_0} \approx 8.8542 \times 10^{-12} & \text{[F/m]}, \end{cases}$$

with c_0 being the speed of light in vacuum. The non-dimensional quantities $\boldsymbol{\varepsilon}_r$ (relative electric permittivity) and $\boldsymbol{\mu}_r$ (relative magnetic permeability) are then introduced:

$$\begin{cases} \boldsymbol{\varepsilon} = \varepsilon_0 \boldsymbol{\varepsilon}_r, \\ \boldsymbol{\mu} = \mu_0 \boldsymbol{\mu}_r. \end{cases}$$

Practically, only a few materials exhibit a high relative magnetic permeability. They are referred as ferromagnetic, or ferrimagnetic depending on the microscopic mechanisms involved [98]. Basically, these materials are (pure or alloys of) iron, cobalt and nickel [23]. On the other hand, most materials exhibit a close to one relative permeability. Two categories can be further distinguished:

1. materials with a smaller than one relative permeability, called diamagnetic;
2. materials with a higher than one relative permeability, called paramagnetic.

On the electric side, materials exhibiting a high relative permittivity are referred as dielectric [98].

In the case of conducting materials, according to Ohm's law, the electric field gives rise to an electrical current. We thus have:

$$\boldsymbol{J} = \boldsymbol{\sigma}\, \boldsymbol{E}, \tag{1.3}$$

where $\boldsymbol{\sigma}$ is the material electrical conductivity expressed in $[1/(\Omega\mathrm{m})]$. As for the previous quantities, in the general context of an anisotropic and inhomogeneous material, the conductivity is a tensor function of space. In this work, we also make the assumption that the conductivity is only a function of the space coordinates. For homogeneous isotropic materials, it reduces to a scalar constant σ.

1.1.2 Time-harmonic hypothesis

In a general context, Maxwell's equations are time-dependent. However, in this thesis, this time dependence is assumed to be harmonic. The $e^{-\jmath\omega t}$ convention is used, where ω is the angular frequency of the electromagnetic signals and $\jmath$ the imaginary unit. Thus, the electromagnetic fields can be decomposed into a known time-dependent component and a possibly unknown space-dependent component:

$$\begin{cases} \boldsymbol{E}(\mathbf{x},t) = \Re\Big[e^{-\jmath\omega t}\boldsymbol{e}(\mathbf{x})\Big], \\ \boldsymbol{H}(\mathbf{x},t) = \Re\Big[e^{-\jmath\omega t}\boldsymbol{h}(\mathbf{x})\Big], \\ \boldsymbol{D}(\mathbf{x},t) = \Re\Big[e^{-\jmath\omega t}\boldsymbol{d}(\mathbf{x})\Big], \\ \boldsymbol{B}(\mathbf{x},t) = \Re\Big[e^{-\jmath\omega t}\boldsymbol{b}(\mathbf{x})\Big], \\ \boldsymbol{J}(\mathbf{x},t) = \Re\Big[e^{-\jmath\omega t}\boldsymbol{j}(\mathbf{x})\Big], \\ \varrho(\mathbf{x},t) = \Re\Big[e^{-\jmath\omega t}\rho(\mathbf{x})\Big]. \end{cases}$$

This decomposition is valid by linearity of Maxwell's equations and of the constitutive laws. It is worth recalling, that we imposed $\boldsymbol{\mu}$, $\boldsymbol{\varepsilon}$ and $\boldsymbol{\sigma}$ to be functions of the space coordinates only. Actually, this condition can be slightly relaxed: these values can be functions of ω (which is a fixed value in a time-harmonic context). By substituting this decomposition into (1.1), the so-called time-harmonic Maxwell's equations are obtained:

$$\begin{cases} \mathbf{curl}\, \boldsymbol{e} = \jmath\omega \boldsymbol{b}, & \text{(1.4a)} \\ \mathbf{curl}\, \boldsymbol{h} = -\jmath\omega \boldsymbol{d} + \boldsymbol{j}, & \text{(1.4b)} \\ \operatorname{div} \boldsymbol{d} = \rho, & \text{(1.4c)} \\ \operatorname{div} \boldsymbol{b} = \mathbf{0}. & \text{(1.4d)} \end{cases}$$

In order to obtain an electromagnetic wave equation, the constitutive laws (1.2) and (1.3) are substituted into Ampère's and Faraday's laws:

$$\begin{cases} \mathbf{curl}\, \boldsymbol{e} = \jmath\omega\mu_0 \boldsymbol{\mu}_r \boldsymbol{h}, \\ \mathbf{curl}\, \boldsymbol{h} = -\jmath\omega\varepsilon_0 \boldsymbol{\varepsilon}_r \boldsymbol{e} + \boldsymbol{\sigma}\boldsymbol{e}. \end{cases}$$

Moreover, let us define the following quantities:

- k, the wavenumber in free space, defined as $k = \omega/c_0 = \omega\sqrt{\mu_0\varepsilon_0}$ [rad/m];
- η_0, the intrinsic impedance of free space, defined as $\eta_0 = \sqrt{\mu_0/\varepsilon_0}$ [Ω];
- $\tilde{\boldsymbol{\varepsilon}}_r$, the complex relative electric permittivity, defined as $\tilde{\boldsymbol{\varepsilon}}_r = \boldsymbol{\varepsilon}_r + \frac{\jmath}{k}\eta_0\boldsymbol{\sigma}$ [–].

Then, by taking the curl of Faraday's law, and by eliminating the magnetic field, we have:

$$\begin{aligned}
& \mathbf{curl}\left(\boldsymbol{\mu}_r^{-1}\,\mathbf{curl}\,\boldsymbol{e}\right) = \jmath\omega\mu_0(-\jmath\omega\varepsilon_0\boldsymbol{\varepsilon}_r\boldsymbol{e} + \boldsymbol{\sigma}\boldsymbol{e}),\\
\iff \quad & \mathbf{curl}\left(\boldsymbol{\mu}_r^{-1}\,\mathbf{curl}\,\boldsymbol{e}\right) = \omega^2\mu_0\varepsilon_0\boldsymbol{\varepsilon}_r\boldsymbol{e} + \jmath\omega\mu_0\boldsymbol{\sigma}\boldsymbol{e},\\
\iff \quad & \mathbf{curl}\left(\boldsymbol{\mu}_r^{-1}\,\mathbf{curl}\,\boldsymbol{e}\right) = k^2\boldsymbol{\varepsilon}_r\boldsymbol{e} + \jmath\omega\sqrt{\mu_0^2}\sqrt{\frac{\varepsilon_0}{\varepsilon_0}}\boldsymbol{\sigma}\boldsymbol{e},\\
\iff \quad & \mathbf{curl}\left(\boldsymbol{\mu}_r^{-1}\,\mathbf{curl}\,\boldsymbol{e}\right) = k^2\boldsymbol{\varepsilon}_r\boldsymbol{e} + \jmath k\eta_0\boldsymbol{\sigma}\boldsymbol{e},\\
\iff \quad & \mathbf{curl}\left(\boldsymbol{\mu}_r^{-1}\,\mathbf{curl}\,\boldsymbol{e}\right) - k^2\tilde{\boldsymbol{\varepsilon}}_r\boldsymbol{e} = \mathbf{0},
\end{aligned} \tag{1.5}$$

which is the electric-side[2] time-harmonic electromagnetic wave equation.

1.1.3 Eigenvalue problem

Obviously, equation (1.5) can be used to compute the electric field for a given set of boundary conditions. However, it can be also used in an other way: with the wavenumber, or more precisely k^2, as the unknown. In other words, one may be interested in finding every possible value of k^2, such that (1.5) holds. This is nothing but an eigenvalue problem, which is discussed in more details in chapter 5.

1.1.4 Interface continuity

What happens if a time-harmonic electromagnetic wave crosses two different media, as shown in Figure 1.1?

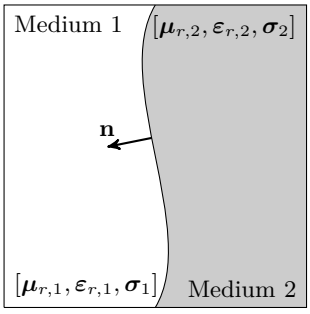


Figure 1.1: Continuity of the electric and magnetic fields across to different media.

[2]It is also possible to take the curl of Ampère's law and to eliminate the electric field, thus leading to a magnetic-side formulation.

By using Maxwell's equations, it can be shown that [119]:

$$\begin{cases} \mathbf{n} \times (\boldsymbol{h}_1 - \boldsymbol{h}_2) = \boldsymbol{j}_s, & \text{(1.6a)} \\ \mathbf{n} \times (\boldsymbol{e}_1 - \boldsymbol{e}_2) = \mathbf{0}, & \text{(1.6b)} \\ \mathbf{n} \cdot (\boldsymbol{b}_1 - \boldsymbol{b}_2) = 0, & \text{(1.6c)} \\ \mathbf{n} \cdot (\boldsymbol{d}_1 - \boldsymbol{d}_2) = \rho_s, & \text{(1.6d)} \end{cases}$$

where $\boldsymbol{j}_s$ is known as the surface current density (in [A/m]), where ρ_s is known as the surface charge density (in [C/m^2]), and where $\mathbf{n}$ is the unit vector normal to the interface. These equations have the following signification:

1. the tangential component of the magnetic field can be discontinuous at the boundary between two media; the discontinuity is equal to the surface current density at the interface;
2. the tangential component of the electric field is continuous at the boundary between two media;
3. the normal component of the magnetic induction field is continuous at the boundary between two media;
4. the normal component of the electric displacement field can be discontinuous at the boundary between two media; the discontinuity is equal to the surface charge density at the interface.

Let us remark that, if both media are not perfect conductors (*i.e.* with a finite conductivity), the surface current density $\boldsymbol{j}_s$ must be zero [119].

If one of the two media is assumed to be a perfect conductor, according to Ohm's law (1.3), an electric field inside the conductor will induce an infinite current density. Thus, in order for this current to remain bounded, we must have $\boldsymbol{e}_2 = \mathbf{0}$. Furthermore, this implies that:

$$\mathbf{n} \times \boldsymbol{e}_1 = \mathbf{0},$$

that is, the electric field in the first medium must be perpendicular to the interface. Moreover, if $\boldsymbol{e}_2 = \mathbf{0}$, then the magnetic field inside the conductor must also be zero, because of Faraday's law (1.4a). Thus, we may also write:

$$\mathbf{n} \times \boldsymbol{h}_1 = \mathbf{j}_s.$$

Let us note that this finite surface current density is not induced by an electrical field, but by a time varying magnetic field.

1.2 From Navier-Stokes equations to an acoustic wave

1.2.1 Generalities

Let us now focus on acoustic waves. These waves propagate through a fluid, and are thus described by the Navier-Stokes equations [6]. In the case of an inviscid flow without body

forces, we can write [105]:

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \displaystyle\sum_{j=1}^{3} \frac{\partial (\rho\, u_j)}{\partial x_j} = 0, & \text{(1.7a)} \\ \rho \dfrac{\partial u_i}{\partial t} + \rho \displaystyle\sum_{j=1}^{3} u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial P}{\partial x_i} \qquad \forall i \in \{1,2,3\}, & \text{(1.7b)} \end{cases}$$

with the quantities described in Table 1.2. Equation (1.7a) is known as the continuity equa-

Quantity	Symbol	Unit
Velocity	$\boldsymbol{u} = [u_1, u_2, u_3]$	m/s
Density	ρ	kg/m^3
Pressure	P	Pa

Table 1.2: Fluid mechanics quantities.

tion, and equation (1.7b) as the momentum equation.

The system of equations (1.7) can then be linearized around the following quiescent point:

$$\begin{cases} P = \bar{P} + \delta P, & \text{(1.8a)} \\ \rho = \bar{\rho} + \delta \rho, & \text{(1.8b)} \\ \boldsymbol{u} = \bar{\boldsymbol{u}} + \delta \boldsymbol{u}, & \text{(1.8c)} \end{cases}$$

where δP, $\delta \rho$ and $\delta \boldsymbol{u}$ are small variations of pressure, density and velocity, around the mean values $\bar{P}$, $\bar{\rho}$ and $\bar{\boldsymbol{u}}$, which are constant in time and space. It is worth noticing that, since the fluid is at rest, we can further impose:

$$\bar{\boldsymbol{u}} = \boldsymbol{0}. \tag{1.9}$$

Inserting equations (1.8) and (1.9) into (1.7), and by neglecting non-linear perturbation terms, we get:

$$\begin{cases} \dfrac{\partial \delta \rho}{\partial t} + \bar{\rho} \displaystyle\sum_{j=1}^{3} \frac{\partial \delta u_j}{\partial x_j} = 0, & \text{(1.10a)} \\ \bar{\rho} \dfrac{\partial \delta u_i}{\partial t} + \delta \rho \dfrac{\partial \delta u_i}{\partial t} + \bar{\rho} \displaystyle\sum_{j=1}^{3} \delta u_j \frac{\partial \delta u_i}{\partial x_j} = -\frac{\partial \delta P}{\partial x_i} \qquad \forall i \in \{1,2,3\}. & \text{(1.10b)} \end{cases}$$

Using the derivative of a product, we can also write:

$$\begin{aligned} \bar{\rho} \sum_{j=1}^{3} \delta u_j \frac{\partial \delta u_i}{\partial x_j} &= -\bar{\rho} \sum_{j=1}^{3} \delta u_i \frac{\partial \delta u_j}{\partial x_j} + \bar{\rho} \sum_{j=1}^{3} \frac{\partial (\delta u_j \delta u_i)}{\partial x_j}, \\ &\simeq -\bar{\rho} \delta u_i \sum_{j=1}^{3} \frac{\partial \delta u_j}{\partial x_j}, \end{aligned} \tag{1.11}$$

the last equality being due to the linearization process. Moreover, by inserting (1.10a) into (1.11), and using once again the derivative of a product, we have:

$$\begin{aligned}\bar{\rho}\sum_{j=1}^{3}\delta u_j\frac{\partial\delta u_i}{\partial x_j} &= \delta u_i\,\frac{\partial\delta\rho}{\partial t},\\ &= -\delta\rho\,\frac{\partial\delta u_i}{\partial t}+\frac{\partial(\delta u_i\,\delta\rho)}{\partial t},\\ &\simeq -\delta\rho\,\frac{\partial\delta u_i}{\partial t}.\end{aligned}\tag{1.12}$$

Finally, inserting (1.12) into (1.10b), the system of equations (1.10) reduces to

$$\begin{cases}\dfrac{\partial\delta\rho}{\partial t}+\bar{\rho}\displaystyle\sum_{j=1}^{3}\frac{\partial\delta u_j}{\partial x_j}=0,\\ \quad\bar{\rho}\dfrac{\partial\delta u_i}{\partial t}+\dfrac{\partial\delta P}{\partial x_i}=0 \qquad\qquad \forall i\in\{1,2,3\},\end{cases}$$

which can be rewritten as:

$$\begin{cases}\dfrac{\partial\delta\rho}{\partial t}+\bar{\rho}\,\mathrm{div}\,\delta\boldsymbol{u}=0, & (1.14a)\\ \bar{\rho}\dfrac{\partial\delta\boldsymbol{u}}{\partial t}+\mathbf{grad}\,\delta P=0. & (1.14b)\end{cases}$$

After this linearization process, let us take the time derivative of (1.14a) and the divergence of (1.14b). Doing so, we end up with:

$$\begin{cases}\dfrac{\partial^2\delta\rho}{\partial t^2}+\bar{\rho}\,\mathrm{div}\,\dfrac{\partial\delta\boldsymbol{u}}{\partial t}=0,\\ \bar{\rho}\,\mathrm{div}\,\dfrac{\partial\delta\boldsymbol{u}}{\partial t}+\mathrm{div}\,\mathbf{grad}\,\delta P=0.\end{cases}$$

Combining these two last equations, we can write:

$$\frac{\partial^2\delta\rho}{\partial t^2}-\mathrm{div}\,\mathbf{grad}\,\delta P=0.$$

Since the problem involves only small perturbations of pressure and density, the phenomenon can be considered as isentropic. With this assumption, the speed of sound writes [7]:

$$c=\sqrt{\left(\frac{\delta P}{\delta\rho}\right)_S}.$$

Then, we finally have:

$$\frac{1}{c^2}\frac{\partial^2\delta P}{\partial t^2}-\mathrm{div}\,\mathbf{grad}\,\delta P=0,\tag{1.16}$$

which is the acoustic wave equation.

1.2.2 Time-harmonic hypothesis

As for electromagnetic waves, let us now consider the particular case of harmonic excitations, that is:

$$\delta P(\mathbf{x}, t) = \Re\Big[e^{-\jmath\omega t} p(\mathbf{x})\Big].$$

By inserting this decomposition into (1.16), and defining the wavenumber as $k = \omega/c$, we finally obtain:

$$\operatorname{div} \mathbf{grad}\, p + k^2 p = 0, \tag{1.17}$$

which is the time-harmonic acoustic wave equation.

1.2.3 Eigenvalue problem

As for the electromagnetic case, equation (1.17) can be used to compute the pressure field for a given set of boundary conditions. However, it also possible to treat this equation as an eigenvalue problem with k^2 as the unknown.

1.2.4 Interface continuity

As studied for electromagnetic waves, what happens if a time-harmonic acoustic wave crosses two different media? By neglecting the surface tension between the two media, it can be shown that the pressure field is continuous across the interface [78]:

$$p_1 = p_2, \tag{1.18}$$

where p_1 is the pressure field on one side of the interface, and p_2 the pressure on the other side.

1.3 Absorbing conditions

When dealing with wave propagation, it is often needed to consider an infinite computational domain Ω. To ensure that the wave is indeed going towards infinity, and not backwards, an additional condition must be imposed on Maxwell's equations or the acoutics equation. For a three-dimensional time-harmonic acoustic wave, this condition writes:

$$\lim_{\|\mathbf{x}\|\to\infty} \|\mathbf{x}\| \left(\frac{\mathbf{x}}{\|\mathbf{x}\|} \cdot \mathbf{grad} - \jmath k \right) p(\mathbf{x}) = 0, \tag{1.19}$$

for any radial direction $\mathbf{x}$, and is known as the Sommerfeld radiation condition [55]. On the other hand, for time-harmonic electromagnetic waves we have:

$$\lim_{\|\mathbf{x}\|\to\infty} \|\mathbf{x}\| \left(\frac{\mathbf{x}}{\|\mathbf{x}\|} \times \mathbf{curl} + \jmath k \right) \boldsymbol{e}(\mathbf{x}) = \mathbf{0}, \tag{1.20}$$

which is known as the Silver-Müller radiation condition [97].

When computing numerical solutions of time-harmonic wave problems, it is not always possible to handle an infinite domain. This latter is usually truncated at some finite distance, as shown in Figure 1.2. Thus, in order to model the same physics as in the infinite situation, an appropriate condition must be imposed on the truncating boundary $\partial\Omega_\infty$.

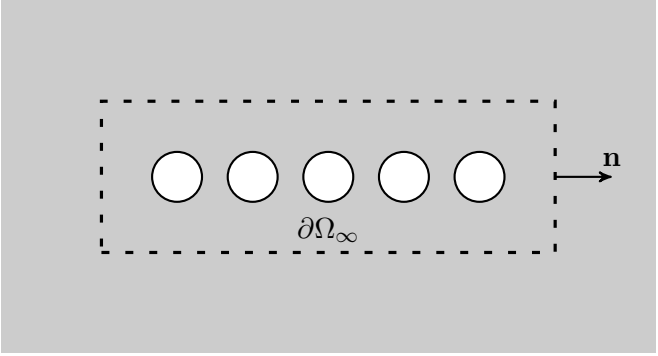


Figure 1.2: Infinite domain truncation.

This condition is usually referred as an absorbing condition. Formally, it links the pressure field with its normal derivative on $\partial\Omega_\infty$, and the electric field with the magnetic field on $\partial\Omega_\infty$:

$$\mathbf{n}\cdot\mathbf{grad}\,p+\mathcal{B}(p)=0 \qquad \text{or} \qquad \mathbf{n}\times\mathbf{curl}\,\boldsymbol{e}+\mathcal{B}(\boldsymbol{e})=0, \tag{1.21}$$

where $\mathcal{B}$ is a well chosen absorbing operator, and $\mathbf{n}$ the unit vector outwardly oriented normal to $\partial\Omega_\infty$.

Based on the Sommerfeld and Silver-Müller radiation conditions, a simple choice for the absorbing operator is:

$$\begin{cases} \mathcal{B}(p)=-\jmath k\,p & \text{for time-harmonic acoustic waves,} \quad (1.22a)\\ \mathcal{B}(\boldsymbol{e})=+\jmath k\,\mathbf{n}\times\boldsymbol{e}\times\mathbf{n} & \text{for time-harmonic electromagnetic waves.} \quad (1.22b) \end{cases}$$

These operators actually neglect the possible curvature of the boundary, or the non-normal incidence of the wave at the boundary. More sophisticated operators can be proposed, that take these effects into account [55].

Another approach for handling truncated infinite domain is the following. Instead of imposing an absorbing condition on $\partial\Omega_\infty$, an additional layer surrounding $\partial\Omega_\infty$ is added as shown in Figure 1.3. Then, the physical properties (*e.g.*, the speed of sound or light) of the layer are chosen so that the wave entering this new domain Ω_{PML} is perfectly absorbed and dissipated. For this reason, the domain Ω_{PML} is called a perfectly matched layer or PML [15, 16, 68, 95].

Only one kind of PML will be considered in this thesis: the so-called Cartesian (or Bérenger's) PML. With this approach, the truncation layer is a rectangle (or a rectangular parallelepiped in three dimensions) aligned with the Cartesian axes. This rectangular shell can be decomposed into three regions in 2D, or seven in 3D, as shown in Figure 1.4 for the two-dimensional case. Each region is associated with the damping of a specific direction of the wave. In a two-dimensional scenario, we have:

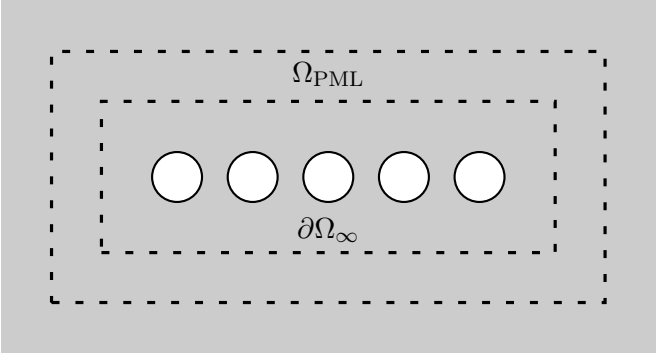


Figure 1.3: Perfectly matched layer.

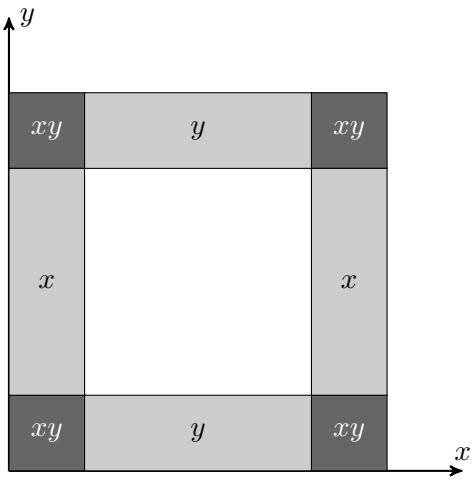


Figure 1.4: The three regions of a two-dimensional Cartesian PML.

1. in the x region, the wave can propagate freely in the y direction, and is damped in the x direction;
2. in the y region, the wave can propagate freely in the x direction, and is damped in the y direction;
3. in the xy region, the wave is damped in both x and y directions.

On the other hand, in a three-dimensional situation, the seven following regions have to be considered: x, y, z, xy, xz, yz and xyz.

For each direction, an absorbing function can be defined: $\sigma_x(x)$, $\sigma_y(y)$, $\sigma_z(z)$. Then, we impose that $\sigma_i = 0$ in the regions of the PML, where the i^{th} direction does not need a damping. With these damping functions in hand, the following quantities can be formed:

$$\begin{cases} \gamma_x = 1 + \jmath\dfrac{\sigma_x}{\omega}, \\ \gamma_y = 1 + \jmath\dfrac{\sigma_y}{\omega}, \\ \gamma_z = 1 + \jmath\dfrac{\sigma_z}{\omega}, \end{cases}$$

where ω is the wave angular frequency. Of course, this definition makes sense only in Ω_{PML}.

For an electromagnetic wave, we may then define the following relative permeability and permittivity tensors:

$$\begin{cases} \boldsymbol{\varepsilon}_r = \operatorname{diag}\left[\dfrac{\gamma_y\gamma_z}{\gamma_x}, \dfrac{\gamma_x\gamma_z}{\gamma_y}, \dfrac{\gamma_x\gamma_y}{\gamma_z}\right], \\ \boldsymbol{\mu}_r = \operatorname{diag}\left[\dfrac{\gamma_y\gamma_z}{\gamma_x}, \dfrac{\gamma_x\gamma_z}{\gamma_y}, \dfrac{\gamma_x\gamma_y}{\gamma_z}\right], \end{cases}$$

that will define the properties of the PML medium. For the acoustic case, we must first rewrite (1.17) in the following form

$$\operatorname{div}(\boldsymbol{\beta}\,\mathbf{grad}\,p) + k^2\,\alpha\,p = 0.$$

Then, we have:

$$\begin{cases} \boldsymbol{\beta} = \operatorname{diag}\left[\dfrac{\gamma_y\gamma_z}{\gamma_x}, \dfrac{\gamma_x\gamma_z}{\gamma_y}, \dfrac{\gamma_x\gamma_y}{\gamma_z}\right], \\ \alpha = \gamma_x\gamma_y\gamma_z. \end{cases}$$

To finalize the construction of a Cartesian PML, one has then to choose a set of damping functions $\sigma_x(x)$, $\sigma_y(y)$, $\sigma_z(z)$. A common choice is to take σ_i as an hyperbolic function [17].

Finally, let us also notice that non-Cartesian PMLs can also be constructed [37, 95, 126, 127].

1.4 Numerical methods for solving partial differential equations

Solving analytically partial differential equations, as the two wave equations derived previously, is unfortunately only possible for very simple cases, exhibiting for instance very simple geometries (*e.g.* circles or rectangles). Thus, in order to solve real-life engineering problems, featuring for example complicated geometries, it becomes necessary to use numerical techniques to approximate the solution. Classically, the partial differential problem is approximated by an algebraic (linear) system of equations. This step is referred to as the discretization of the partial differential equation.

This section presents a brief review of some popular methods. The following acoustic problem will be used as an example:

$$\text{Find } p(\mathbf{x}): \begin{cases} (\operatorname{div}\mathbf{grad} + k^2)p(\mathbf{x}) = 0 & \forall \mathbf{x} \in \Omega, \quad (1.23a) \\ p(\mathbf{x}) = g(\mathbf{x}) & \forall \mathbf{x} \in \partial\Omega, \quad (1.23b) \end{cases}$$

where Ω is a two-dimensional computational domain, $\partial\Omega$ its boundary, $g(\mathbf{x})$ a known scalar function defined on $\partial\Omega$, and $p(\mathbf{x})$ the unknown pressure field to compute in Ω.

1.4.1 Finite difference method

The first method that usually comes to mind, when dealing with numerical solution of partial differential equations, is the finite difference method. Its idea is quite simple. The computational domain Ω is first covered with a *regular* grid, as shown in Figure 1.5.

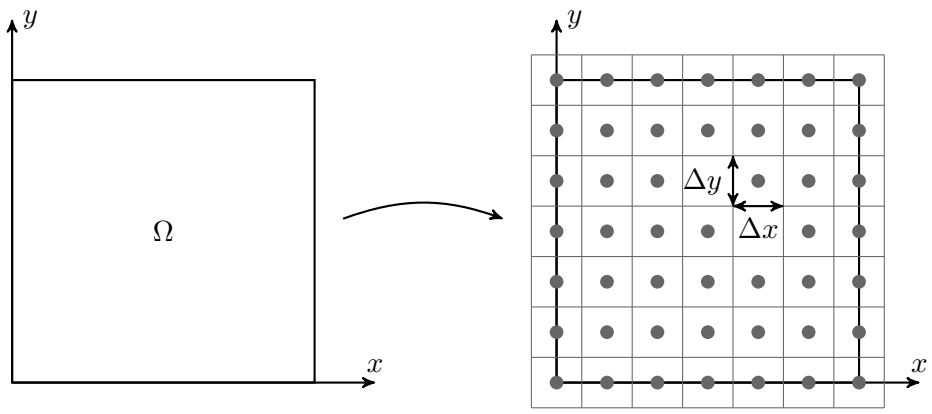


Figure 1.5: Finite difference grid.

This two-dimensional example depicts a grid with cells of size $\Delta x \times \Delta y$. Each cell is usually identified by a set of indices defining its center. For instance, in a two-dimensional context, the indices (i, j) refer to the cell with its center located at:

$$\mathbf{x}_{i,j} = [i\ \Delta x, j\ \Delta y]^T .$$

Then, the derivatives of the partial differential equation are *discretized* on this grid. For instance, the derivative $\frac{\partial^2}{\partial x^2}$ at some point $\mathbf{x}_{i,j}$ can be approximated, using Taylor's expansion, by:

$$\left.\frac{\partial^2 p(\mathbf{x})}{\partial x^2}\right|_{\mathbf{x}=\mathbf{x}_{i,j}} \simeq \frac{p(\mathbf{x}_{i-1,j}) - 2p(\mathbf{x}_{i,j}) + p(\mathbf{x}_{i+1,j})}{\Delta x^2}.$$

Applying this formula to each cell, we can generate a linear system, linking every $p(\mathbf{x}_{i,j})$, were some $p(\mathbf{x}_{i,j})$ may have been fixed with a Dirichlet condition. By solving this linear system, we can recover a discrete approximation of the solution p, represented by a set of values defined at the center of each cell. Intuitively, it can be understood that the finer the grid, the closer to p the approximate solution will be.

One of the advantages of this method is its excellent scaling performances on large simulation clusters. On the other hand, this method also exhibits a significant drawback: the need of regular grid, limiting seriously its field of application.

Finally, among the possible variations of the finite difference method, let us cite the finite difference time domain (FDTD) method [135], which is a popular technique for solving the Maxwell's equations in the time-domain (1.1).

1.4.2 Boundary element method

Let us now consider the boundary element method. It relies on the integral representation theorem [99], which states that the pressure field, solution of (1.23), must satisfy:

$$p(\mathbf{x}) = -\int_{\partial\Omega} G(\mathbf{x},\mathbf{y}) \frac{\partial p(\mathbf{y})}{\partial n_y}\, \mathrm{d}\partial\Omega(\mathbf{y}) + \int_{\partial\Omega} \frac{\partial G(\mathbf{x},\mathbf{y})}{\partial n_y}\, p(\mathbf{y})\, \mathrm{d}\partial\Omega(\mathbf{y}),$$

where $G(\mathbf{x},\mathbf{y})$ is the Green function of the $(\mathrm{div}\,\mathbf{grad} + k^2)$ operator, and $\frac{\partial}{\partial n_y}$ is the normal derivative (with respect to the $\mathbf{y}$ coordinates). The Green function can be thought as the

impulse response of the system. Formally, we have:

$$\begin{cases} \Big(\operatorname{div} \mathbf{grad} + k^2 \Big) \Big[G(\mathbf{x}, \mathbf{y}) \Big] = \delta(\mathbf{y}), \\ \qquad G(\mathbf{x}, \mathbf{y}) \text{ is an outgoing wave,} \end{cases}$$

where $\delta(\mathbf{y})$ is Dirac pulse imposed on $\mathbf{y}$.

Basically, this equation states that, if the Green function of the differential operator is known, and if the pressure field and its normal derivative are known on the *boundary* of the domain Ω, then the solution can be reconstructed anywhere in the domain Ω. Because of the Dirichlet conditions, the pressure field is known on $\partial\Omega$. Thus, only its normal derivative has to be computed.

The boundary element method proposes techniques to link the pressure field and its normal derivative[3], by exploiting the Green function. The major advantage of this approach is that the problem has to be solved only on the boundary of the domain, thus leading to smaller linear systems. However, a major limitation of the boundary element method is that the material properties in the volume have to be (piecewise) homogeneous. If not, the problem cannot be defined using only boundary integrals, and finding the Green function is a real challenge. Furthermore, despite the fact that the linear systems are indeed smaller, they are *dense*, which leads to large memory and computation time requirements. Acceleration techniques have been actively developed for the last few decades to alleviate these issues, for instance the fast multipole method [46, 106] or H-matrix techniques [60].

1.4.3 Finite element method

The last method is the finite element, or FE, method. Basically, this method relies on the following idea: the solution of the problem is assumed to be a linear combination of piecewise polynomial functions:

$$p(\mathbf{x}) = \sum_{j=0}^{N-1} a_j v^j(\mathbf{x}) \qquad v^j(\mathbf{x}) \in V,$$

where the $v^j(\mathbf{x})$ are polynomials, classically of the first-order, spanning the finite-dimensional function space V. Moreover, these polynomial functions are usually constructed such that their support is compact.

Practically, the computational domain Ω is meshed, as shown in Figure 1.6, and the polynomial basis functions are defined on the mesh elements[4]. A classical choice for the polynomials is to require that each $v^j(\mathbf{x})$ is such that:

$$v^j(\mathbf{x}) = \begin{cases} 1 & \text{at the } j^{\text{th}} \text{ vertex of the mesh,} \\ 0 & \text{at the other vertices of the mesh,} \end{cases}$$

thus leading to a finite element basis, of size N, for the function space V.

[3] Or the electric and the magnetic field for Maxwell's equations.

[4] In the example of Figure 1.6, these mesh elements are triangles; it is worth noticing that other elementary geometries can be chosen: quadrangles, tetrahedra, prisms, ...

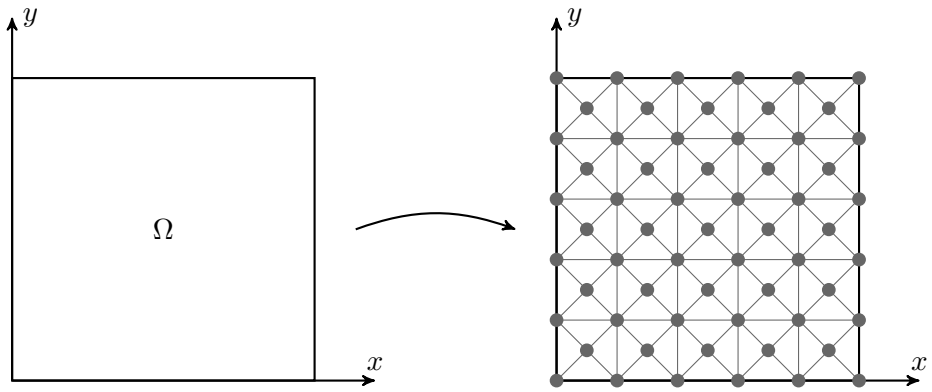


Figure 1.6: Finite element mesh.

The finite element method proposes then a technique to compute the interpolation coefficients a_j, and to choose an appropriate basis for V. Basically, these coefficients are found by solving an N by N linear system. More details are given in chapter 2.

This approach is quite robust, and can naturally handle complicated geometries (since no uniform grids are needed), non-homogeneous problems and non-linear problems. On the other hand, the scaling performances are not exceptional, therefore very large-scale problems are not always treatable.

Among the possible variations of this approach, we have the high-order finite element method. In this case, instead of the first-order polynomial basis presented above, higher-order polynomials are used. They lead to high-accuracy solutions, while keeping the number of unknowns to manageable values. Moreover, they also permit simulations on curved meshes, thus allowing better approximations of the computational domain itself. In this thesis, we will focus on the high-order finite element method.

1.5 Difficulties in solving wave problems in the high-frequency regime

Solving time-harmonic wave problems is known to be a difficult task, especially in the high-frequency regime (*i.e.* with a high value of the wavenumber k), for two major reasons: the pollution effect and the sign-indefiniteness. The pollution effect will impact the numerical discretization strategy, and the sign-indefiniteness the choice of a linear solver.

The remainder of this section will illustrate these two difficulties using the finite element method. However, other methods suffer also from similar problems [24, 58].

1.5.1 The pollution effect

The discretization of wave problems, as the ones defined by equations (1.5) or (1.17), is known to suffer from the so-called pollution, or dispersion, effect, which is described here.

First-order finite element methods

More precisely, when an order 1 FE discretization is used on a uniform mesh, with mesh elements of size h, the relative error (between the exact and the FE solutions) can be written as (in H^1-seminorm) [69]:

$$e \leq C_1 kh + C_2 k^3 h^2, \tag{1.24}$$

where C_1 and C_2 are constants independent of k or h.

This equation implies that asymptotically, *i.e.* when the mesh size becomes sufficiently small, the error is bounded by $C_1 kh$. Let us now consider the pre-asymptotic behavior, *i.e.* when the mesh size is not sufficiently small. In this case, according to (1.24), two situations arise, depending on the wavenumber value:

1. if $k \ll 1$, then the error is bounded by $C_1 kh$;
2. if $k \gg 1$, then the error is bounded by $C_2 k^3 h^2$.

Thus, for the high-frequency regime, the error of the FE discretization suffers from a pollution term that decreases the method accuracy.

From a practical point-of-view, this implies that very fine meshes are needed for solving high-frequency wave problems. More precisely, for low-frequency simulations, it is a rule of thumb to keep the value kh constant (usually $kh = 0.1$). That is, if the frequency is doubled, by doubling the meshing density, the relative error will remain of the same magnitude. On the other hand, in the high-frequency regime, the mesh density has to be increased more than linearly to preserve the error, at least in the pre-asymptotic region.

To illustrate this phenomenon, let us consider an infinite two-dimensional electromagnetic metallic waveguide, truncated with a Silver-Müller radiation condition, and excited with the TM_0 mode. The electromagnetic wave equation (1.5) is solved using the FE method and the solution is compared to the analytical [119] one[5]:

$$\boldsymbol{e}(x, y, z) = E_0 \, e^{\jmath kx} \mathbf{e}_y,$$

where E_0 is a real non-zero value and $\mathbf{e}_y = [0, 1, 0]^T$. Simulations are carried out for different wavenumbers ($k = 5$, 10, 25 and 50 [rad/m]) and mesh densities ($n_\lambda = 5$, 10 and 20 points per wavelength [λ^{-1}]). The relative error, in L^2-norm, between the FE solution and the exact one is plotted in Figure 1.7. Based on these results, we can directly notice that the error grows as we increase the frequency, while keeping the mesh density constant (*i.e.* keeping kh constant). In other words, and as predicted by the theory, if we double the frequency, we have to more than double the mesh density to keep the error constant.

[5] The solution of the TM_0 mode from [119] has been adapted from the $e^{\jmath\omega t}$ time-dependence convention, to the $e^{-\jmath\omega t}$ convention used in this thesis.

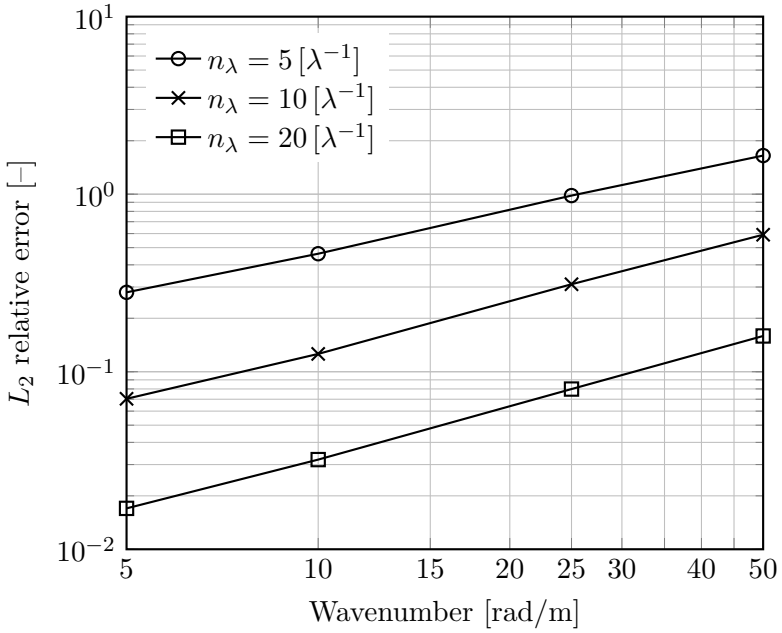


Figure 1.7: Illustration of the pollution effect on a two-dimensional metallic waveguide (mode TM_0).

High-order finite element methods

Let us now consider a FE discretization of order p, with p greater than one. In this case, it can be shown [70] that the relative error between the FE method and the exact solution is bounded by:

$$e \leq C_1(p) \left(\frac{kh}{2p}\right)^p + C_2(p)k \left(\frac{kh}{2p}\right)^{2p}, \tag{1.25}$$

where $C_1(p)$ and $C_2(p)$ are constants independent of k or h, but dependent on p.

Comparing the first-order error estimate (1.24) and the high-order one (1.25), it can be directly noticed that the pollution term in k^3h^2 in the low-order case becomes $k(kh/2p)^{2p}$ when higher-order discretizations are used. Thus, by increasing the FE order (and assuming k and h unchanged), the low-order pollution term is penalized by $(kh/2p)^{2p-2}$, which is a rapidly decreasing function of p. To be complete, according to the theory [70], $C_2(p)$ can increase significantly with p. However, in practice, the pollution effect is still alleviated by high-order FE methods.

Finally, to illustrate this phenomenon, the low-order test case is now extended to higher-order discretizations. For simplicity, only one mesh density is selected: $n_\lambda = 20$ points per wavelength $[\lambda^{-1}]$. Figure 1.8 summarizes the obtained results. From this graph, it can be directly noticed that high-order FE methods are reducing the pollution effect. Indeed, if we take the third-order simulation for instance, we can directly notice that doubling the frequency, while keeping the mesh density constant, leads to results with close errors.

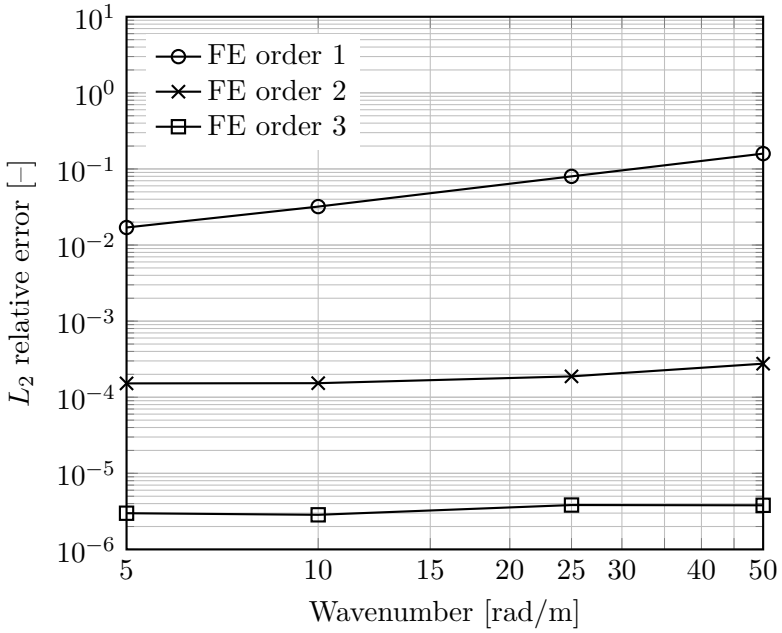


Figure 1.8: Illustration of the pollution effect when high-order FE discretizations are used on a two-dimensional metallic waveguide (mode TM_0 at $n_\lambda = 20\,[\lambda^{-1}]$).

1.5.2 Linear system solvers

One more point has to be discussed regarding the difficulties encountered when solving wave problems: linear solvers. When solving the N by N linear system $\mathbf{Ax} = \mathbf{b}$, arising from a numerical discretization, two strategies are available: direct and iterative.

Direct solvers

Probably the simplest strategy to solve a linear system is to use a Gaussian elimination process. Mathematically, this idea is described by factorizing the system matrix in $\mathbf{A} = \mathbf{LU}$, where $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices. Then, two triangular systems are solved, using forward ($\mathbf{Ly} = \mathbf{b}$) and backward ($\mathbf{Ux} = \mathbf{y}$) substitutions, to recover the original system solution:

$$\mathbf{Ax} = \mathbf{b} \iff \mathbf{LUx} = \mathbf{b} \iff \mathbf{Ly} = \mathbf{b}.$$

More precisely, the system $\mathbf{Ly} = \mathbf{b}$ is solved by noticing that [57]

$$y_i = \frac{\left(b_i - \sum_{j=1}^{i-1} L_{i,j} y_j \right)}{L_{i,i}} \qquad \forall i \in \{1, \dots, N\},$$

and the system $\mathbf{Ux} = \mathbf{y}$ is solved thanks to the formula

$$x_i = \frac{\left(y_i - \sum_{j=i+1}^{N} U_{i,j} x_j \right)}{U_{i,i}} \qquad \forall i \in \{1, \dots, N\}.$$

The simplest **LU** algorithm is given by Algorithm 1.

Many variations and improvements of this last algorithm have been proposed, and nowadays highly robust implementations of this method are available. However they all suffer from the same drawbacks. First, the algorithm is intrinsically sequential, thus its parallelization is a real challenge. The speedup of state-of-the-art solvers saturate thus quickly with the number of computing units. Practically, it is usually useless to use more than 100 cores with this kind of solvers [86]. Moreover, even if the matrix $\mathbf{A}$ is sparse, its **LU** decomposition is usually not. Even with state-of-the-art reordering strategies, the memory requirement of direct solver remains a serious limitation[6] [86]. Finally, the memory scaling of direct solvers is also a significant limitation: that is, increasing the number of computing nodes does not necessarily enable to treat larger linear systems [86]. Indeed, the per node memory consumption is not uniformly distributed across the nodes, since the sparsity pattern of the **LU** decomposition is hard to predict. Thus, practically, increasing the number of cores to more than 100 will not enable the treatment of larger linear systems.

However, despite these drawbacks, direct methods are probably the most *robust* approaches at the time. Let us cite the libraries MUMPS[7] [3, 4], PARDISO[8] [82, 112, 113] and Su-

[6] It is worth noticing that out-of-core strategies are available: that is, a portion of the memory requirement is offloaded on the hard drive. However, this comes with a significant processing time increase, since input/output operations are dramatically slow on hard disks.

[7] Available at: `http://mumps-solver.org`.

[8] Available at: `http://pardiso-project.org`.

```
decomposeLU(A)
    Description
        In-place LU decomposition of the given matrix A.
        After decomposition:
        • the upper diagonal of A, diagonal included, is the matrix U;
        • the lower diagonal of A, diagonal excluded, is the matrix L.
        It is worth noticing that diag L = 1.
    Implementation (OCTAVE)
        // Size of A
        N = size(A, 1);
        // In-place LU
        for(k = 1:N-1)
            A(k+1:N, k) = A(k+1:N, k) / A(k, k);
            for(i = k+1:N)
                for(j = k+1:N)
                    A(i, j) = A(i, j) - A(i, k) * A(k, j);
```

Algorithm 1: Matrix decomposition in **LU** form.

perLU[9] [87] for sparse systems, and LAPACK[10] [5] for dense systems. At the time of this thesis, no records of direct solvers scaling on more than a hundred cores has been found. Because the **LU** decomposition decreases the sparsity of the matrix $\mathbf{A}$ (the so-called *fill-in*), the per node available memory is usually the bottleneck of this approach. In this thesis (see chapter 5), the MUMPS solver limit was found to roughly 10 million unknowns using an order 5 FE method on 120 computing nodes; each one being equipped with 64[GB] of memory.

Iterative solvers

The other family of solvers is of the iterative kind. This means that the solution is found by constructing a series of estimations $\mathbf{x}_m$, which converges to the exact solution. Among those solvers, an important family are Krylov solvers. In this case, the solution is constructed using the Krylov subspace $\mathcal{K}_m$:

$$\mathcal{K}_m = \text{span}\left\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{m-1}\mathbf{r}_0\right\}. \tag{1.26}$$

A famous algorithm following this approach is the Generalized Minimum Residual method or GMRES [108, 110], which relies on the following ideas.

1. An orthonormal basis for the Krylov subspace $\mathcal{K}_m$ is constructed.
2. The norm of the residual in this basis is minimized, that is:

$$\min_{\mathbf{z}_m \in \mathcal{K}_m} \left\|\mathbf{b} - \mathbf{A}(\mathbf{x}_0 + \mathbf{z}_m)\right\| = \min_{\mathbf{z}_m \in \mathcal{K}_m} \left\|\mathbf{r}_0 - \mathbf{A}\mathbf{z}_m\right\|. \tag{1.27}$$

[9] Available at: `http://crd-legacy.lbl.gov/~xiaoye/SuperLU`.
[10] Available at: `http://www.netlib.org/lapack`.

3. The norm of residual at step m is evaluated:
   (a) if it is small enough the algorithm has found the solution;
   (b) otherwise, m is incremented and the algorithm returns to step 1.

The GMRES algorithm proposes a robust and cost-effective way to:

1. construct an orthonormal basis for $\mathcal{K}_m$, using Arnoldi's method [8];
2. minimize (1.27), using the properties of Arnoldi's method and rotation matrices;
3. evaluate the norm of the residual at the m^{th} step without computing $\mathbf{x}_m$ explicitly (actually, $\mathbf{x}_m$ is computed only when the algorithm has converged).

Algorithm 2 gives a more precise description of the method.

$\mathtt{x_m}$ = **gmres**(A, b, $\mathtt{x_0}$, m)

Description

Returns an approximate solution of the linear system $\mathbf{Ax} = \mathbf{b}$ using GMRES with m iterations and an initial guess $\mathbf{x}_0$.

Implementation (OCTAVE)

```
// Size
N = size(A, 1);

// Initialize matrices V_m and H̄_m
V_m = zeros(N, m);
H̄_m = zeros(m+1, m);

// Compute initial residual and the first Krylov vector
r_0 = b - A * x_0;
V_m(:,1) = r_0 / norm(r_0);

// Arnoldi method for generating a basis for K_m
for(j = 1:m)
  for(i = 1:j)
    H̄_m(i,j) = (A * V_m(:,j))' * V_m(:,i);

  v̂ = Av_j − Σ_{i=1}^{j} h_{i,j} v_i;

  H̄_m(j + 1, j) = norm(v̂);
  V_m(:,j + 1) = v̂ / H̄_m(j + 1, j);

// Compute approximate solution x_m
y_m = solve (1.28);
x_m = x_0 + V_m * y_m;
```

Algorithm 2: Generalized Minimum Residual algorithm (GMRES).

Let us start with the first stage of Algorithm 2: Arnoldi's iteration. It populates two matrices: $\mathbf{V}_m$ (N by m) and $\bar{\mathbf{H}}_m$ ($m+1$ by m). It can be shown [108, 110] that $\mathbf{V}_m$ is a orthonormal basis for $\mathcal{K}_m$, and that the minimization problem (1.27) is equivalent to:

$$\min_{\mathbf{z}_m \in \mathcal{K}_m} \left\| \mathbf{b} - \mathbf{A}(\mathbf{x}_0 + \mathbf{z}_m) \right\| = \min_{\mathbf{z}_m \in \mathcal{K}_m} \left\| \mathbf{r}_0 - \mathbf{A}\mathbf{z}_m \right\| = \min_{\mathbf{y}_m \in \mathbf{V}_m} \left\| \|\mathbf{r}_0\| \, \mathbf{e}_1 - \bar{\mathbf{H}}_m \mathbf{y}_m \right\|, \qquad (1.28)$$

where $\mathbf{e}_1$ is the first column vector of the $m+1$ by $m+1$ unit matrix, and $\mathbf{z}_m = \mathbf{V}_m \mathbf{y}_m$. The $\mathbf{x}_m$ solution approximate is now given by:

$$\mathbf{x}_m = \mathbf{x}_0 + \mathbf{V}_m \mathbf{y}_m.$$

The minimization problem of (1.28) is solved by computing a rotation matrix $\mathbf{Q}_m$ with the following properties:

1. $\mathbf{Q}_m$ is unitary;
2. $\mathbf{Q}_m \bar{\mathbf{H}}_m$ leads to an upper triangular matrix $\mathbf{R}_m$ with only zeros on its last row;
3. computing $\mathbf{Q}_{m+1}$ can be done easily by reusing $\mathbf{Q}_m$ (see [108] for more details).

Exploiting the unity of $\mathbf{Q}_m$, (1.28) becomes:

$$\min_{\mathbf{y}_m \in \mathbf{V}_m} \left\| \|\mathbf{r}_0\| \, \mathbf{e}_1 - \bar{\mathbf{H}}_m \mathbf{y}_m \right\| = \min_{\mathbf{y}_m \in \mathbf{V}_m} \left\| \mathbf{Q}_m \left(\|\mathbf{r}_0\| \, \mathbf{e}_1 - \bar{\mathbf{H}}_m \mathbf{y}_m \right) \right\| = \min_{\mathbf{y}_m \in \mathbf{V}_m} \|\mathbf{g}_m - \mathbf{R}_m \mathbf{y}_m\| , \tag{1.29}$$

where $\mathbf{g}_m = \|\mathbf{r}_0\| \, \mathbf{Q}_m \mathbf{e}_1$. Since $\mathbf{R}_m$ has only zeros on its last row, the solution of (1.29) is obtained by solving the m by m system obtained by eliminating the last row. This is a simple procedure, since the problem is triangular. Moreover, it can be shown [108, 110] that the last element of $\mathbf{g}_m$ is the norm of the residual $\mathbf{r}_m$.

The presented algorithm is the basic version of the GMRES, and improvements are of course possible. An important improvement is the following. Due to memory limitations, storing the basis $\mathbf{V}_m$ is not always possible for large values of m[11]. Moreover, because of the finite-precision arithmetic, the last vectors of $\mathbf{V}_m$ are no longer orthogonal if m is high. This problem can be solved by *restarting* the GMRES. That is, the maximum value of m is fixed (usually $m \ll N$). If the residual after m Arnoldi steps if not sufficiently small, a new GMRES is started with the solution $\mathbf{x}_m$ as initial guess.

To be complete, in the special case where $\mathbf{A}$ is hermitian definite-positive, powerful alternatives to GMRES are possible. Among them, the Conjugate Gradient method or CG [63, 108] is probably the most well known. One major advantage of CG compared to GMRES, comes from the fact that it is not required to store all the basis vectors for $\mathcal{K}_m$. Actually, only the current and the last vectors are needed, leading to a highly memory efficient method.

Let us note that the library PETSc[12] [11, 12] offers high-performance implementations of GMRES and other iterative solvers.

The first obvious advantage of Krylov methods over direct methods is that only matrix-vector and vector-vector products are required in the Krylov approach. This leads to two benefits:

1. the sparsity of $\mathbf{A}$ is preserved;
2. matrix-vector and vector-vector operations can be realized efficiently in parallel.

This allows an excellent scaling on high performance parallel platforms, both in term of speedup and memory distribution.

[11] Ultimately, $\mathbf{V}_N$ is an N by N *dense* matrix!

[12] Available at: `http://www.mcs.anl.gov/petsc`.

On the other hand, the question of the GMRES *convergence* has to be raised: can the norm of the residual be decreased up to zero (or the computer precision), and how many iterations does it take? It can be shown [108, 110] that, for a positive definite matrix $\mathbf{A}$, the restarted GMRES converges for any $m > 1$. It is worth noticing that for the non-restarted GMRES, the convergence is always guaranteed. However, as already explained, this is not feasible on large systems because of memory limitations.

Unfortunately, time-harmonic wave problems are sign-indefinite[13], and iterative solvers exhibit poor performances, as depicted in Figure 1.9.

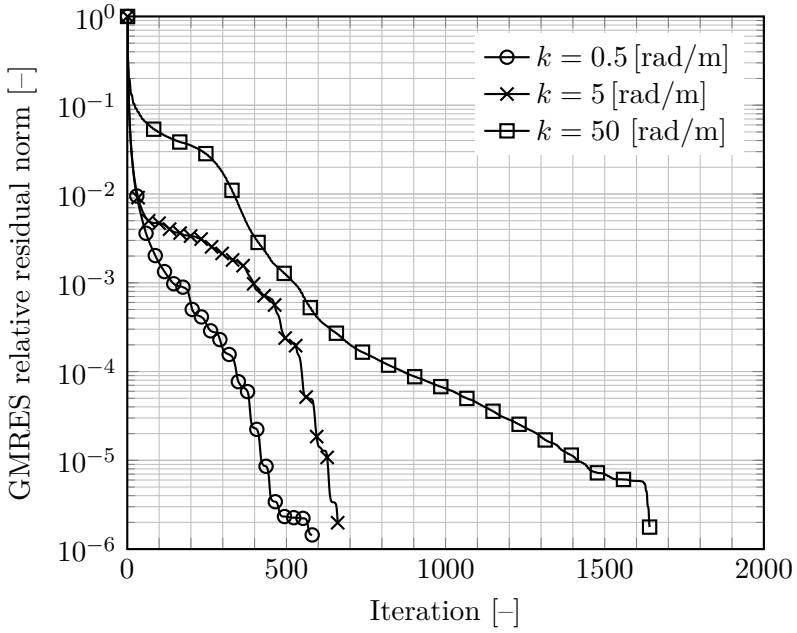


Figure 1.9: Residual history of a non-restarted GMRES for a two-dimensional waveguide, exited with the TM_0 mode at different frequencies.

In this test case, simulations are carried out on a two-dimensional waveguide exited with the TM_0 mode for different frequencies: $k = \{0.5, 5, 50\}$[rad/m]. For the $k = 50$[rad/m] case, the computational domain has been meshed with 8 triangles per wavelength. The same mesh is used in the other cases. The linear system obtained by discretizing the Maxwell's equation (using the finite element method) is solved using a non-restarted GMRES. It can be directly seen that, by increasing the frequency, the convergence rate of the residual decreases. It is worth noticing that in every situation, systems of the same size are solved. Furthermore, the choice of an appropriate size for the Krylov subspace is also complicated (for indefinite matrices): if it is too small, the convergence will stagnate to a given value; if it is too large, it is no longer possible to store it.

The poor convergence of iterative solvers can be improved by *preconditioning* the linear system. That is, instead of solving $\mathbf{Ax} = \mathbf{b}$, the following system is solved:

$$\mathbf{MAx} = \mathbf{Mb},$$

[13] At least, for the traditionally used formulations [96].

where $\mathbf{M}$ is a preconditioner[14] for $\mathbf{A}$. The key idea is to construct matrix $\mathbf{M}$ such that $\mathbf{MA}$ exhibits better properties, with respect to convergence of the iterative solver. Ideally, if $\mathbf{M} = \mathbf{A}^{-1}$, the linear system becomes the identity and solving it becomes trivial. Thus, two properties are required for a good preconditioner:

1. $\mathbf{M}$ should be cheap to construct;
2. $\mathbf{M}$ should be as close as possible to $\mathbf{A}^{-1}$.

At the time of this thesis, finding good preconditioners for time-harmonic problems is still an active research field [49]. While, for instance, shifted-Laplace preconditioner [48] have had some success in preconditioning wave type systems, no robust technique for high-frequency problems is currently known.

Hybrid direct-iterative solvers

Finally, a third family of solvers can be cited: the hybrid ones. The idea is to take the advantages of both direct and iterative solvers by combining them. More precisely, the computational domain is first decomposed into sub-domains, thus defining a set of sub-problems of smaller size. These sub-problems can then be solved independently, and thanks to their reduced size, are amenable to direct solvers. In order to enforce the coherence of the solution at the boundary between two sub-domains, an iterative scheme is set up.

At each iteration, a new set of sub-problems is solved. The differences between two iterations are the *boundary conditions* of the sub-problems. Basically, the solution of one sub-problem will be used, at the next iteration, as the boundary condition of the neighbor sub-problem[15]. Eventually, this iterative process can benefit from Krylov acceleration. This kind of approach falls into the field of domain decomposition methods [51], and will be analyzed deeper in chapter 6.

1.6 Computing architectures

To conclude this chapter, let us now introduce the different computing architectures available nowadays for scientific computing.

1.6.1 Central processing unit and memory

Clearly, all the methods presented previously are implemented using a (or many) computer(s). The heart of this machine is the *processor*, or central processing unit (CPU), and is nothing else but an electronic device, which follows a set of stored instructions, called a program, and capable of processing data thanks to logic and arithmetic operations.

Nowadays, a CPU embeds more than a single processing unit, called in this context a *core*. Thus, a single CPU can *simultaneously* carry out many operations. Furthermore, a modern computer may embed more than one CPU chip, in order to further increase to total number of cores. In this context, a CPU chip is called a *socket*.

[14] Or more precisely, a left-preconditioner.
[15] That is the sub-problem defined on the neighbor sub-domain.

In addition to this operating part, a computer is also composed of memory, where a large amount of data can be stored or fetched by the CPU(s). To simplify the presentation, let us assume that our computer consists in only one socket with only one core. Roughly, the memory can be divided into three levels:

1. the cache memory;
2. the main, or random access, memory (RAM);
3. and the non-volatile memory.

Let us begin by the second level: the main memory. As its name suggests, this is the place where (almost) all the processed data are stored. However, accessing this memory can be slow (compared to the processing speed of the CPU), and thus the cache memory is used.

This cache memory is physically located on the CPU chip, and benefits therefore of a very high access speed. On the other hand, it suffers from a highly limited amount of storage: a few megabytes, compared to the tens or hundreds of gigabytes of the main memory. The idea is that the cache memory is a copy of a restricted portion of the main memory[16]. Thus, if a computer code can reuse as much as possible the data stored in cache, it will benefit from a higher data throughput.

Finally, it is worth noticing that these two memory levels are volatile: *i.e.*, the stored data are lost once the power supply is shut down. Thus, a non-volatile memory is also used, and is usually implemented by an (array of) hard-disk drive(s)[17]. In addition to its non-volatile nature, the hard-disk memory offers also an extremely high amount of storage: above the terabyte. In order to use the data stored in this last memory level, copies with main memory must be done. It is worth noticing that these copies are extremely slow (tens of megabytes per second compared to some gigabytes per second). Thus, most of the computations should be done in main and cache memory.

If we consider now architectures with many sockets and many cores, a few modifications to the previous model must be done. First, on a single socket, the cache memory is further decomposed in many levels. Each core has its own private cache, called level one (or L1). This cache is private, since only its associated core can access it. Then, a shared cache is introduced, called level two (or L2)[18]. As its name suggests, this cache can be accessed by all the cores of the socket. The coherence between the L1 caches and the L2 cache is automatically handled by the hardware. Finally, the whole main memory can be accessed by any core of any socket[19].

The above presentation has been voluntary simplified, and is intended to introduce only fundamental notions. More details on the computer memory organization can be found in [40, 43, 125, 132].

[16] This copy mechanism is directly operated by the hardware: *i.e.*, if the requested part of the main memory is not in cache, it will be automatically fetched.

[17] More recently, solid-state drives are available: they exhibit a significantly higher read/write speed, but at the cost of a reduced amount of storage.

[18] Depending on the hierarchy between the cores, additional cache levels can be introduced.

[19] But possibly at a non-uniform speed, depending on the accessed portion of the memory.

1.6.2 A cluster

When very large simulations have to be carried out, a single computer is not always sufficient: either because it cannot compute the solution in a reasonable amount of time, or because of memory limitations. To further increase the number of cores and the amount of memory, many computers are then interconnect through a network. The set of all these computers is called a *cluster*, and a particular computer in this set is referred to as a *node*.

In this environment, each node has its own memory, and no other node can access it. Thus, if the nodes need to collaborate to solve a common problem, they need to communicate by *exchanging messages*.

1.6.3 Programming paradigms

How can we write a computer software, that can efficiently exploit an architecture of interconnected many cores nodes? To answer this question, let us first forget the hardware to focus only on the following parallel programming paradigms: *process* based and *thread* based.

Process based parallelism

Let us start by introducing the process based parallelism. This notion of *process* comes from the world of operating systems. Without entering the details, out of the scope of this thesis, a process can be thought as an instance of a computer program [125]. Among the properties of a process, the following is of main importance: a process cannot access the memory allowed to another process.

If one wants to use many processes to solve a single problem, the following strategy can be used. First, every process is an instance of the *same* computer program. Thanks to a software mechanism, each process is associated to an identifier. With these identifiers, the processes can alter their behavior to separate the work between them. However, often, the processes have to communicate to handle the given task. Since they cannot access the memory of another process, they need to exchange messages.

Classically, these message exchanges are handled by the message passing interface standard, or MPI[20]. Many implementations of this standard are available: for instance, we may cite OpenMPI[21] and IntelMPI[22].

Let us go back to the hardware. Obviously, this process based paradigm seems appropriate to handle clusters, since the nodes can access only their own memory. However, this approach can also be used when all the cores are on the same node. Of course, this time, the cores can access the whole memory. However, by using a process based approach, the operating system will enforce that each process has access to only its own portion of the memory. This time, the messages are not transmitted on a network, but through the operating system.

Of course, in the situation where many nodes with many cores are available, a full process based strategy can be followed. Inside a given node, the communications are carried out

[20] More information on this standard is available at: `http://mpi-forum.org`.
[21] Available at: `http://www.open-mpi.org`.
[22] Available at: `http://software.intel.com/en-us/intel-mpi-library`.

through the operating system. On the other hand, when two processes located on different nodes need to communicate, the messages are sent through the network.

Because of this private view of the memory (enforced by the hardware or the operating system), the process based parallel programming approach is also referred to as the *distributed memory* paradigm.

Thread based parallelism

Let us now present the thread based parallelism. We saw previously that, on a single node, even if the cores share the same memory, the operating system forbids a process to access the memory allocated by another process. While legitimate from the operating system point of view [125], this prohibition can lead to avoidable complications. This is where the notion of thread comes into place.

Basically, a thread can be seen as a control point of a process [125]. Pragmatically, this means that:

1. any thread can access the memory allocated by its process;
2. different threads can be at different points in the process code.

Thus, by using threads, it is possible to drive many cores to handle a common task. However, this time, the memory restriction does not apply, at least at the process level. That is, a thread can access the memory allocated to its process, but cannot access the memory allocated to another process.

They are many ways to handle threads. However, classically, OpenMP directives are used. More precisely, OpenMP is a specification for a set of compiler directives, library routines, and environment variables, that can be used to specify high-level thread based parallelism in Fortran, C and C++ programs[23].

Since a thread is attached to a process, this paradigm applies only to a single node. That is, by contrast to the process based parallelism, it cannot be used to exploit nodes interconnected by a network. By opposition the distributed memory approach, the thread based parallel programming approach is also referred as the *shared memory* paradigm.

Hybrid process-thread based parallelism

By the nature of a thread, it is of course possible to mix both process and thread based approaches. Classically, on cluster with N nodes and C cores per node, we will create N processes (distributed across the N nodes) with C threads per process.

1.6.4 General-purpose computing on graphics processing units

Driven by a video game industry increasingly demanding, modern graphics processing units, or GPU, have become true platforms for intensive computing. On this basis, manufacturers began to create new architectures, allowing the development and the execution of general

[23] See `http://openmp.org` for more information.

codes on the GPU. This new programming approach has been named General Purpose computing on Graphics Processing Units, or GPGPU.

As already explained, a CPU is a processing unit embedding both control and arithmetic modules. On a GPU, a huge emphasis is given to arithmetic instead of control, at the cost of a control part with a reduced complexity. For instance, a control unit will drive many arithmetic units, thus imposing that all these units perform the exact same operation at a given time. Thanks to this trade-off, a GPU provides a massive arithmetic platform at a relatively low cost.

Two main difficulties arise when programming a graphics processing unit. First, the limited amount of memory: both in term of cache and main memory. Second, the GPU main memory is independent from the CPU main memory. Thus, copies are needed to send the data to the GPU, and to recover the solution on the CPU. These copies are extremely slow compared to the GPU computing throughput. These two factors have to be taken into account, to optimize the performances of the complete processing chain.

More information on GPGPU are available in [76, 92, 111].

1.7 Conclusion

We presented in this chapter the governing equations for time-harmonic electromagnetic and acoustic waves.

Since analytical solutions exist only for a very limited number of cases, numerical techniques are needed to solve these kind of problems. Among the possible approaches, we have the finite element method, which is known to be robust and effective to treat complicated geometries and non-homogeneous material laws.

Unfortunately, it is well known that the convergence rate of the solution, with respect to the mesh refinement, is penalized by a pollution term, that increases with the frequency. This means that, in the high-frequency regime, very large meshes are needed to recover the optimal quadratic convergence of the first-order FE method. However, by using high-order discretizations, the pollution effect can be limited. Thus the increase of the number of unknowns can be controlled, while keeping accurate solutions.

Even with the help of high-order FE methods, high-frequency time-harmonic problems lead to large linear systems to solve. Unfortunately, direct methods do not scale well and iterative methods require a large number of iterations, or do not converge at all. An alternative is then to use a domain decomposition method.

In this thesis, efficient high-order FE algorithms will be studied, as well as domain decomposition methods for solving the resulting systems, for solving the resulting systems on large-scale, distributed memory, computational architectures.

Chapter 2

Key concepts of the finite element method

As explained before, this work relies on the finite element method, because of its robustness and its ability to handle complicated geometries and non-homogeneities. This chapter presents the fundamentals of this method, and is inspired by the reference books [19, 27, 97].

2.1 Variational formulations

Before entering into the finite element method, let us introduce briefly variational formulations. To motivate this approach, let us consider the following model problem on the one-dimensional domain $\Omega = [-1, 1]$:

$$\frac{\partial}{\partial x}\left(\alpha(x)\frac{\partial v(x)}{\partial x}\right) = 0 \qquad \forall x \in [-1, 1], \tag{2.1}$$

where $v(x)$ is the unknown function, and where $\alpha(x)$ is known. Obviously, for this problem to make sense, the solution $v(x)$ has to be at least twice-differentiable on $[-1, 1]$, and $\alpha(x)$ at least differentiable.

When modeling physical phenomena, it is sometimes needed to allow, for instance, a jump in $\alpha(x)$. Classically, this problem is solved by treating two sub-problems: one on each side of the jump and by imposing continuity conditions at the interface. However, a more elegant approach is possible. Let us consider now the following equation:

$$\int_{-1}^{1} \frac{\partial}{\partial x}\left(\alpha(x)\frac{\partial v(x)}{\partial x}\right)\varphi(x)\,\mathrm{d}x = 0 \qquad \forall \varphi \in F,$$

where F is a function space such that

$$F = \Big\{\varphi : \varphi \text{ is continuous and differentiable on } [-1, 1] \text{ and } \varphi \text{ vanishes on } \{-1, 1\}\Big\}.$$

The functions φ are referred to as test functions. Integrating by parts, we have:

$$\begin{aligned}
&\int_{-1}^{1} \frac{\partial}{\partial x}\left(\alpha(x)\frac{\partial v(x)}{\partial x}\right)\varphi(x)\,\mathrm{d}x &&= 0 \qquad \forall \varphi \in F(\Omega),\\
\iff &-\int_{-1}^{1}\left(\alpha(x)\frac{\partial v(x)}{\partial x}\right)\frac{\partial \varphi(x)}{\partial x}\,\mathrm{d}x + \underbrace{\left[\alpha(x)\frac{\partial v(x)}{\partial x}\varphi(x)\right]_{-1}^{1}}_{0} &&= 0 \qquad \forall \varphi \in F(\Omega), \qquad (2.2)
\end{aligned}$$

with the null term due to the definition of F. Proceeding this way has many advantages:

1. the function $\alpha(x)$ can now be discontinuous;
2. it makes sense for the derivative of v to be undefined at some points, since the integral needs to be defined only almost everywhere;
3. by an appropriate choice of test functions, interface conditions can be directly enforced.

This last advantage is easy to verify in our example. Let us assume that the jump in $\alpha(x)$ arises at $x = 0$. Then, we have from (2.2), by splitting the integration interval:

$$\begin{aligned}
&-\int_{-1}^{0^-}\left(\alpha(x)\frac{\partial v(x)}{\partial x}\right)\frac{\partial \varphi(x)}{\partial x}\,\mathrm{d}x - \int_{0^+}^{1}\left(\alpha(x)\frac{\partial v(x)}{\partial x}\right)\frac{\partial \varphi(x)}{\partial x}\,\mathrm{d}x = 0 \qquad \forall \varphi \in F(\Omega),\\
\iff &+\int_{-1}^{0^-}\frac{\partial}{\partial x}\left(\alpha(x)\frac{\partial v(x)}{\partial x}\right)\varphi(x)\,\mathrm{d}x + \left[\alpha(x)\frac{\partial v(x)}{\partial x}\varphi(x)\right]_{-1}^{0^-}\\
&+\int_{0^+}^{1}\frac{\partial}{\partial x}\left(\alpha(x)\frac{\partial v(x)}{\partial x}\right)\varphi(x)\,\mathrm{d}x + \left[\alpha(x)\frac{\partial v(x)}{\partial x}\varphi(x)\right]_{0^+}^{1} &&= 0 \qquad \forall \varphi \in F(\Omega).
\end{aligned}$$

Moreover, since the sum of the two integrals is zero by (2.2), we have:

$$\begin{aligned}
&\alpha(0^-)\frac{\partial v(x)}{\partial x}\Big|_{0^-}\varphi(0^-) - \alpha(0^+)\frac{\partial v(x)}{\partial x}\Big|_{0^+}\varphi(0^+) = 0 && \forall \varphi \in F(\Omega),\\
\iff &\alpha(0^-)\frac{\partial v(x)}{\partial x}\Big|_{0^-} = \alpha(0^+)\frac{\partial v(x)}{\partial x}\Big|_{0^+} && \forall \varphi \in F(\Omega),
\end{aligned}$$

since φ is continuous in $[-1, 1]$. This last equation, is nothing but the continuity condition at the α jump.

Equation (2.1) is called a strong formulation, and equation (2.2) is called a weak or variational formulation. It can be shown that the solutions of (2.1) are also satisfying (2.2). However, equation (2.2) allows more solutions, since it is less restrictive about the continuity and differentiability. Somehow, by integrating our strong form, we have redefined the notion of function and the notion of derivative, making them less restrictive. This idea is the root of the theory of distributions [117].

2.1.1 Function spaces

Let us introduce the function spaces that will be used for modeling electromagnetic and acoustic waves. We have:

$$H^1(\Omega) = \left\{ v \in L_2(\Omega) \middle| \mathbf{grad}\, v \in \left[L_2(\Omega)\right]^3 \right\}, \tag{2.3}$$

$$H(\mathbf{curl}, \Omega) = \left\{ \boldsymbol{v} \in \left[L_2(\Omega)\right]^3 \middle| \mathbf{curl}\, \boldsymbol{v} \in \left[L_2(\Omega)\right]^3 \right\}, \tag{2.4}$$

$$H(\mathrm{div}, \Omega) = \left\{ \boldsymbol{v} \in \left[L_2(\Omega)\right]^3 \middle| \mathrm{div}\, \boldsymbol{v} \in L_2(\Omega) \right\}, \tag{2.5}$$

where $L_2(\Omega)$ is the space of square integrable functions over Ω, and where the differential operators have to be taken in the distributional sense.

By (weakly) imposing continuous gradients in Ω, the $H^1(\Omega)$ space is a natural choice for enforcing a field continuity, as the pressure field for instance. On the other hand, by (weakly) imposing continuous curls in Ω, the $H(\mathbf{curl}, \Omega)$ space is a natural choice for enforcing the tangential continuity of a field, as the electric field for instance. And finally, by (weakly) imposing continuous divergences in Ω, the $H(\mathrm{div}, \Omega)$ space is a natural choice for enforcing the normal continuity of a field, as the magnetic induction field for instance.

Three other function spaces are also important to define. Usually, for a given problem defined over Ω, conditions are imposed on the boundary of Ω, $\partial\Omega$. These conditions are of three types:

1. conditions imposing the value of the unknown field at the boundary; these conditions are called Dirichlet conditions;
2. conditions imposing the derivative of the unknown field at the boundary; these conditions are called Neumann conditions;
3. conditions linking the unknown field and its derivative at the boundary; these conditions are called impedance, or Robin, conditions.

Thus, $\partial\Omega$ can be divided in two non-overlapping sub-domains:

1. $\partial\Omega_D$, where Dirichlet conditions are imposed;
2. $\partial\Omega_N$, where Neumann and impedance conditions are imposed.

With this notation in mind, the following spaces can be defined:

$$H_0^1(\Omega) = \left\{ v \in H^1(\Omega) \middle| v = 0 \text{ on } \partial\Omega_D \right\}, \tag{2.6}$$

$$H_0(\mathbf{curl}, \Omega) = \left\{ \boldsymbol{v} \in H(\mathbf{curl}, \Omega) \middle| \mathbf{n} \times \boldsymbol{v} \times \mathbf{n} = \mathbf{0} \text{ on } \partial\Omega_D \right\}, \tag{2.7}$$

$$H_0(\mathrm{div}, \Omega) = \left\{ \boldsymbol{v} \in H(\mathrm{div}, \Omega) \middle| (\mathbf{n} \cdot \boldsymbol{v})\mathbf{n} = \mathbf{0} \text{ on } \partial\Omega \right\}, \tag{2.8}$$

where $\mathbf{n}$ is the unit vector outwardly oriented normal to $\partial\Omega$. Those spaces correspond to the

above H^1, $H(\mathbf{curl})$ and $H(\mathrm{div})$ spaces, but with homogeneous Dirichlet conditions[1].

Finally, an interesting property of these spaces is the following:

$$\forall v \in H^1(\Omega) : \mathbf{grad}\, v \in H(\mathbf{curl}, \Omega), \tag{2.9}$$

$$\forall \boldsymbol{v} \in H(\mathbf{curl}, \Omega) : \mathbf{curl}\, \boldsymbol{v} \in H(\mathrm{div}, \Omega), \tag{2.10}$$

which also holds with $H_0^1(\Omega)$, $H_0(\mathbf{curl}, \Omega)$ and $H_0(\mathrm{div}, \Omega)$.

2.1.2 Electromagnetic waves

Let us now derive the variational formulation for the time-harmonic electromagnetic wave equation (1.5) defined on Ω. Applying the integration strategy presented at the beginning of this section, we have:

$$\int_\Omega \mathbf{curl}\left(\boldsymbol{\mu}_r^{-1}\,\mathbf{curl}\,\boldsymbol{e}\right)\cdot\boldsymbol{e}'\,\mathrm{d}\Omega - k^2\int_\Omega (\tilde{\boldsymbol{\varepsilon}}_r\,\boldsymbol{e})\cdot\boldsymbol{e}'\,\mathrm{d}\Omega = 0 \quad \forall \boldsymbol{e}' \in H_0(\mathbf{curl}, \Omega),$$

where the test functions $\boldsymbol{e}'$ have been chosen in $H_0(\mathbf{curl}, \Omega)$, to impose only the tangential continuity of the unknown electric field $\boldsymbol{e}$, and to handle Dirichlet conditions[2]. Applying Green's formula to the $\mathbf{curl}\,(\mathbf{curl}\,)$ term, and using the relation $\mathrm{div}\,(\boldsymbol{b}\times\boldsymbol{a}) = \boldsymbol{a}\cdot\mathbf{curl}\,\boldsymbol{b} - \boldsymbol{b}\cdot\mathbf{curl}\,\boldsymbol{a}$, we may write:

$$\begin{aligned}\int_\Omega \left(\boldsymbol{\mu}_r^{-1}\,\mathbf{curl}\,\boldsymbol{e}\right)\cdot\left(\mathbf{curl}\,\boldsymbol{e}'\right)\,\mathrm{d}\Omega - k^2\int_\Omega (\tilde{\boldsymbol{\varepsilon}}_r\,\boldsymbol{e})\cdot\boldsymbol{e}'\,\mathrm{d}\Omega \\ + \int_{\partial\Omega}(\mathbf{n}\times\boldsymbol{\mu}_r^{-1}\,\mathbf{curl}\,\boldsymbol{e})\cdot\boldsymbol{e}'\,\mathrm{d}\partial\Omega = 0 \quad \forall \boldsymbol{e}' \in H_0(\mathbf{curl}, \Omega),\end{aligned} \tag{2.11}$$

which is the variational formulation of the electromagnetic wave equation (1.5).

By looking at the boundary integral in (2.11), we can directly notice that this term involves the curl of electric field at the boundary of Ω. Thus, this term can be used to impose Neumann or impedance conditions, such as (1.21). On the other hand, since we required $\boldsymbol{e}'$ to be null, on the portion of $\partial\Omega$ where Dirichlet conditions are imposed, this boundary integral is zero on $\partial\Omega_D$.

2.1.3 Acoustic waves

Let us now derive the variational formulation for the time-harmonic acoustic wave equation (1.17) defined on Ω. Once again, using the integration strategy, we have:

$$\int_\Omega \mathrm{div}\,(\mathbf{grad}\,p)\,p'\,\mathrm{d}\Omega + k^2\int_\Omega p\,p'\,\mathrm{d}\Omega = 0 \quad \forall p' \in H_0^1(\Omega),$$

where the test functions p' have been chosen in $H_0^1(\Omega)$, to impose the continuity of the unknown pressure field p, and to handle Dirichlet conditions. Applying Green's formula to

[1] Let us remark that the definition of these space is non-standard: classically, it is assumed that Dirichlet conditions are imposed on the whole boundary, and not on a part of it.

[2] Basically, if the value of the electric field is imposed on the boundary of Ω, it is not necessary to test this part of the domain with $\boldsymbol{e}'$, since the value is already known: thus the choice of $H_0(\mathbf{curl})$ instead of $H(\mathbf{curl})$.

the div (**grad**) term, and using the relation $\operatorname{div}(a\boldsymbol{b}) = \boldsymbol{b} \cdot \mathbf{grad}\, a + a \operatorname{div} \boldsymbol{b}$, we may write:

$$\int_\Omega \mathbf{grad}\, p \cdot \mathbf{grad}\, p' \,\mathrm{d}\Omega - k^2 \int_\Omega p\, p' \,\mathrm{d}\Omega - \int_{\partial\Omega} (\mathbf{n} \cdot \mathbf{grad}\, p)\, p' \,\mathrm{d}\partial\Omega = 0 \quad \forall p' \in H_0^1(\Omega), \tag{2.12}$$

which is the variational formulation of the acoustic wave equation (1.17).

As for the electromagnetic case, the boundary integral in (2.12) is nothing but the normal derivative of the pressure field at the boundary of Ω. Thus, this term can be used to impose Neumann or impedance conditions, such as (1.21). On the other hand, since we required p' to be null, on the portion of $\partial\Omega$ where Dirichlet conditions are imposed, this boundary integral is zero on $\partial\Omega_D$.

2.2 Discretization

Now that we have defined the variational formulations for our time-harmonic wave problems, we can enter the core of this chapter: the discretization of a partial differential equation by the finite element method. For simplicity, let us consider the following acoustic problem defined on Ω:

$$\begin{cases} \displaystyle\int_\Omega \mathbf{grad}\, p \cdot \mathbf{grad}\, p' \,\mathrm{d}\Omega + k^2 \int_\Omega p\, p' \,\mathrm{d}\Omega = 0 \quad \forall p' \in H_0^1(\Omega), & \text{(2.13a)} \\ p = 0 \quad \text{on } \partial\Omega. & \text{(2.13b)} \end{cases}$$

Let us note that, extending what follows to a more general context (*e.g.* electromagnetism, non-homogeneous Dirichlet conditions, impedance conditions, ...) is straightforward.

As already introduced in section 1.4.3, the key idea of the finite element method, is to decompose the solution $p(\mathbf{x})$ into a *finite-dimensional* polynomial basis[3] $V_0^1(\Omega) = \{v^0, v^1, \dots, v^{N-1}\}$ of size N:

$$p(\mathbf{x}) = \sum_{j=0}^{N-1} a_j\, v^j(\mathbf{x}) \qquad v^j \in V_0^1(\Omega),\ a_j \in \mathbb{C}. \tag{2.14}$$

In the finite element method, this basis $V_0^1(\Omega)$ is taken as a finite *subspace* of the test functions space $H_0^1(\Omega)$[4]. Thus, the problem consists in finding the coefficients a_j of (2.14), also called degrees of freedom or DoFs.

Let us insert (2.14) into (2.13), and let us take the test functions in V_0^1. We then have:

$$\begin{aligned} & \int_\Omega \mathbf{grad}\, p \cdot \mathbf{grad}\, v^i \,\mathrm{d}\Omega + k^2 \int_\Omega p\, v^i \,\mathrm{d}\Omega = 0 \quad \forall v^i \in V_0^1(\Omega), \\ \iff & \int_\Omega \mathbf{grad} \left(\sum_{j=0}^{N-1} a_j\, v^j \right) \cdot \mathbf{grad}\, v^i \,\mathrm{d}\Omega + k^2 \int_\Omega \left(\sum_{j=0}^{N-1} a_j\, v^j \right) v^i \,\mathrm{d}\Omega = 0 \quad \forall v^i \in V_0^1(\Omega), \\ \iff & \sum_{j=0}^{N-1} a_j \underbrace{\left(\int_\Omega \mathbf{grad}\, v^j \cdot \mathbf{grad}\, v^i \,\mathrm{d}\Omega + k^2 \int_\Omega v^j\, v^i \,\mathrm{d}\Omega \right)}_{\mathcal{I}_{i,j}} = 0 \quad \forall v^i \in V_0^1(\Omega). \end{aligned}$$

[3] Let us remark that v^i does not refer to the i^{th} power of function v, but to the fact that v^i is the i^{th} function of the polynomial basis V.

[4] Or $H_0(\mathbf{curl}, \Omega)$ for the electromagnetic problems treated in this thesis.

This leads to an N by N algebraic system of equations, with the vector $\mathbf{a} = [a_0, \ldots, a_{N-1}]$ as unknowns:

$$\underbrace{\begin{bmatrix} \mathcal{I}_{0,0} & \cdots & \mathcal{I}_{0,N-1} \\ \vdots & \ddots & \vdots \\ \mathcal{I}_{N-1,0} & \cdots & \mathcal{I}_{N-1,N-1} \end{bmatrix}}_{\mathbf{V}} \underbrace{\begin{bmatrix} a_0 \\ \vdots \\ a_{N-1} \end{bmatrix}}_{\mathbf{a}} = \underbrace{\begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}}_{\mathbf{b}}. \tag{2.15}$$

By solving (2.15), the solution $p(\mathbf{x})$ can be reconstructed using (2.14).

It can be easily shown that for the electromagnetic case (with homogeneous Dirichlet conditions), we have:

$$\begin{cases} \mathbf{V}(i,j) = \displaystyle\int_\Omega \left(\boldsymbol{\mu}_r^{-1}\,\mathbf{curl}\,\boldsymbol{v}^j\right)\cdot\left(\mathbf{curl}\,\boldsymbol{v}^i\right)\,\mathrm{d}\Omega - k^2\int_\Omega \left(\tilde{\boldsymbol{\varepsilon}}_r\,\boldsymbol{v}^j\right)\cdot\boldsymbol{v}^i\,\mathrm{d}\Omega, \\ \mathbf{b}(i) = 0, \end{cases}$$

with the vector basis functions $\boldsymbol{v}^i$ taken in the finite-dimensional basis $V_0(\mathbf{curl},\Omega) \subset H_0(\mathbf{curl},\Omega)$ of size N.

A last question must raised: how can we construct this finite-dimensional basis? This construction is characterized by three aspects [27]:

1. a mesh (also called triangulation) of the computational domain Ω;
2. the finite-dimensional function space V is polynomial;
3. the basis functions of V have a support as small as possible.

Regarding the first property, the mesh consists in representing the domain Ω[5] with simpler geometrical elements K^e (forming the set $\mathcal{K}$), such that:

1. the union of all the elements covers the computational domain, *i.e.* $\bigcup_{\mathrm{K}^e\in\mathcal{K}} \mathrm{K}^e = \overline{\Omega}$;
2. each element $\mathrm{K}^e \in \mathcal{K}$ is a closed set with a non-empty interior;
3. the elements are non-overlapping, *i.e.* $\mathrm{K}^i \cap \mathrm{K}^j = \emptyset$ with $i \neq j$;
4. the boundary of any element K^e is piece-wise smooth[6];
5. the intersection of two different elements is either empty, a vertex (in a one-dimensional situation), an edge (in a two-dimensional situation), or a face (in a three-dimensional situation) of both elements.

Therefore, the integrals in equation (2.15), can now be computed by splitting the integral across the mesh elements:

$$\begin{aligned} &\int_\Omega \mathbf{grad}\,v^j\cdot\mathbf{grad}\,v^i\,\mathrm{d}\Omega + k^2\int_\Omega v^j\,v^i\,\mathrm{d}\Omega \\ &= \sum_{e=0}^{E-1}\int_{\mathrm{K}^e}\mathbf{grad}\,v^j\cdot\mathbf{grad}\,v^i\,\mathrm{dK}^e + k^2\int_{\mathrm{K}^e} v^j\,v^i\,\mathrm{dK}^e, \end{aligned} \tag{2.16}$$

[5] It is not always possible to *exactly* represent the geometry of Ω with the mesh; in this situation the mesh represents a geometry Ω_h which approximates Ω.

[6] More exactly, the boundary has to be Lipschitz-continuous; see [27, 97] for more details.

where E is the total number of mesh elements. The same decomposition can be applied to other integrals (*e.g.*, those obtained for electromagnetism).

Concerning the two last properties, the function space V has to be constructed with polynomials defined on a small support: the idea is then to use the mesh elements as support for those polynomials. Following this, in the case of a polynomial basis spanning a finite subset of H^1, four types of functions can be defined [31].

Vertex based: a function v^{V^i} is said to be vertex based, if its value is non-zero at the i^{th} vertex of the mesh, and if its value is zero at the other vertices.

Edge based: a function $v^{\mathbf{e}^i}$ is said to be edge based, if its value is non-zero on the i^{th} edge of the mesh, if it vanishes on the edge end-points, and if its value is zero on the other edges.

Face based: a function $v^{\mathbf{f}^i}$ is said to be face based, if its value is non-zero on the i^{th} face of the mesh, if it vanishes on the face boundary, and if its value is zero on the other faces.

Cell based: a function v^{C^i} is said to be cell based, if its value is non-zero inside the i^{th} cell of the mesh, if it vanishes on the cell boundary, and if its value is zero on the other cells.

Let us note that, if a first-order basis is constructed, only vertex based functions will be used. By increasing the polynomial order, other types will be introduced [31]. To illustrate this, Figure 2.1 gives an example for three[7] scalar cases in a two-dimensional situation.

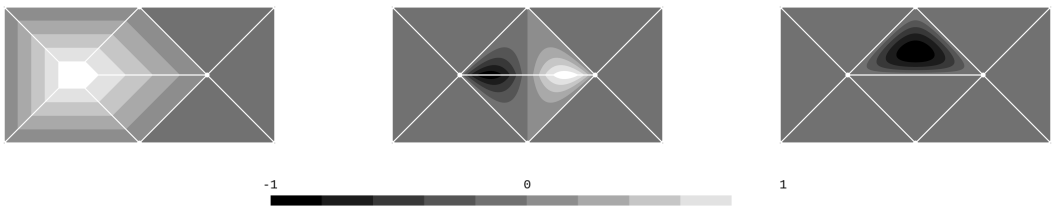


Figure 2.1: Scalar example of (from left to right) vertex, edge and cell based functions.

If we now consider a polynomial basis spanning a finite subset of $H(\mathbf{curl})$, three type of vector functions can be defined [31].

Edge based: a function $\boldsymbol{v}^{\mathbf{e}^i}$ is said to be edge based, if its tangential trace is non-zero on the i^{th} edge of the mesh, and if its tangential trace is zero on the other edges.

Face based: a function $\boldsymbol{v}^{\mathbf{f}^i}$ is said to be face based, if its tangential trace is non-zero on the i^{th} face of the mesh, if its tangential trace vanishes on the face boundary, and if its tangential trace is zero on the other faces.

Cell based: a function $\boldsymbol{v}^{C^i}$ is said to be cell based, if its value is non-zero inside the i^{th} cell of the mesh, if its tangential trace vanishes on the cell boundary, and if its value is zero on the other cells.

[7] Since the examples are in a two-dimensional space, no face can be defined; and thus, no face based function can also be defined.

Again, let us note that, if a first-order basis is constructed, only edge based functions will be used. By increasing the polynomial order, other types will be introduced [31].

By going back to (2.14), each degree of freedom is associated to a basis function. Since each basis function is associated to a vertex, edge, face or cell of the mesh, these DoFs are also associated to a vertex, edge, face or cell. Because of their definition, these basis functions have a small support: that is, the elements touching the corresponding vertex, edge, face; or the interior of an element. This leads to an interesting property of the linear system (2.15): since the basis functions v^i (or $\boldsymbol{v}^i$) are non-zero only on a small portion of the domain, the resulting linear system is *sparse*.

When constructing a finite dimensional space V^1, or $V(\mathbf{curl})$, subset of H^1, or $H(\mathbf{curl})$, one last major property must be specified. These subsets must also verify the inclusions (2.9) or (2.10). In this thesis, we chose to use the basis functions, spanning the finite element spaces V^1 and $V(\mathbf{curl})$, proposed in [136].

2.3 Mappings and reference element

How can we construct a basis for representing our finite-dimensional space V? In the finite element method, this is achieved using geometrical mappings. To simplify the discussion, let us consider a mesh composed of elements with the same geometry (*e.g.*, a mesh only composed of triangles). However, generalizing what follows to more general meshes represents no difficulty.

The key idea is to construct the function space $V(\Omega)$, by using a combination of mappings of functions, taken from a smaller function space $F(\widehat{\mathrm{K}})$ defined on a *reference*[8] element $\widehat{\mathrm{K}}$. In other words, the basis functions defined on this reference element are *mapped* onto the each mesh element, called in this context *physical element* (by opposition to the reference one), thus constructing a basis spanning $V(\Omega)$. Figure 2.2 illustrates this procedure with triangular elements.

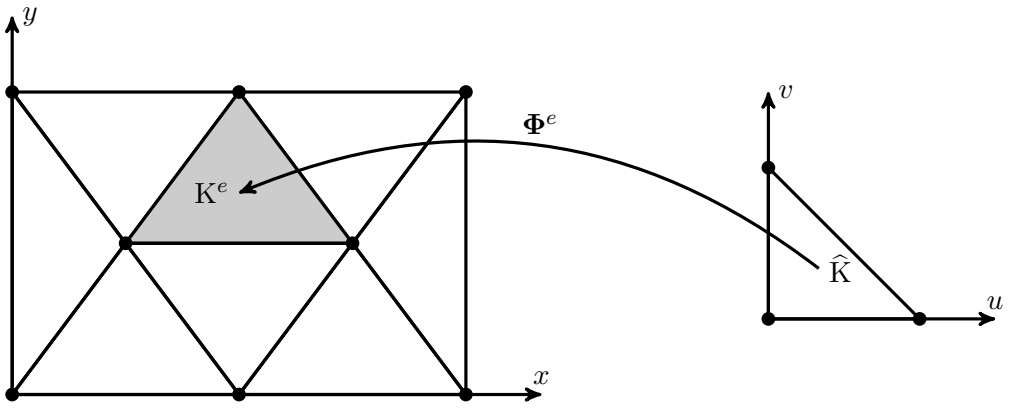


Figure 2.2: Mapping from the triangular reference element to a triangle of the mesh.

[8]Some authors use the term *master element*.

2.3.1 Geometrical mapping

The mapping from $\widehat{\mathrm{K}}$ to K^e

$$\mathbf{\Phi}^e : \widehat{\mathrm{K}} \longrightarrow \mathrm{K}^e, \tag{2.17}$$

has to be a continuously differentiable bijective function [97]. If $\mathbf{u} = [u, v, w]^T$ is a point in the reference element $\widehat{\mathrm{K}}$, then its corresponding point $\mathbf{x} = [x, y, z]^T$ in the physical element K^e is given by:

$$\begin{cases} x = \Phi_x^e(\mathbf{u}), \\ y = \Phi_y^e(\mathbf{u}), \\ z = \Phi_z^e(\mathbf{u}). \end{cases}$$

The Jacobian matrix associated to this transformation is defined as:

$$\boldsymbol{J}^e(\mathbf{u}) = \begin{bmatrix} \dfrac{\partial \Phi_x^e(\mathbf{u})}{\partial u} & \dfrac{\partial \Phi_x^e(\mathbf{u})}{\partial v} & \dfrac{\partial \Phi_x^e(\mathbf{u})}{\partial w} \\ \dfrac{\partial \Phi_y^e(\mathbf{u})}{\partial u} & \dfrac{\partial \Phi_y^e(\mathbf{u})}{\partial v} & \dfrac{\partial \Phi_y^e(\mathbf{u})}{\partial w} \\ \dfrac{\partial \Phi_z^e(\mathbf{u})}{\partial u} & \dfrac{\partial \Phi_z^e(\mathbf{u})}{\partial v} & \dfrac{\partial \Phi_z^e(\mathbf{u})}{\partial w} \end{bmatrix}.$$

Classically, this mapping is implemented by using a first-order Lagrange basis[9], where each Lagrange function is associated with a vertex of the considered element. Formally, we have:

$$\begin{cases} x = \displaystyle\sum_{i=0}^{N-1} x_i \, \varphi_i(\mathbf{u}), \\ y = \displaystyle\sum_{i=0}^{N-1} y_i \, \varphi_i(\mathbf{u}), \\ z = \displaystyle\sum_{i=0}^{N-1} z_i \, \varphi_i(\mathbf{u}), \end{cases}$$

where N is the number of vertices of the element; where $[x_i, y_i, z_i]^T$ are the coordinates, in the physical space, of the i^{th} vertex of the element; and where $\varphi_i(\mathbf{u})$ is the Lagrange function associated to the i^{th} vertex of the element, in the reference space.

This approach can be extended to handle curved physical elements. In this situation, the mapping between the (straight) reference element and the (curved) physical element is implemented by a high-order Lagrange basis. However, by introducing these higher-order Lagrange functions, additional vertices must be introduced in the mesh. Doing so leads to two kinds of vertices: geometrical and topological. The geometrical type refers to all the vertices of the mesh, whether they were introduced to curve the elements or not. On the other hand, the topological type corresponds to the vertices of the associated straight element. Finally, let us note that the reference element is always straight, even if the physical element is curved.

[9] In a Lagrange basis, each function is associated to a vertex; a Lagrange function has then the property to be equal to one at its vertex, and zero at the others.

2.3.2 Field mapping

Now that we know how to construct these mappings, we need to present how a field defined on a physical element is mapped on the reference one. This mapping depends on the nature of the field.

Mapping of function in $H^1(\Omega)$

If the function to be mapped is in $H^1(\Omega)$, then it can be shown [97] that:

$$v(\mathbf{x}) = f \circ \mathbf{\Phi}^{e^{-1}}, \tag{2.18}$$

where $v \in H^1(\Omega)$, $f \in H^1(\widehat{\mathrm{K}})$, and $\mathbf{\Phi}^{e^{-1}}$ is the inverse of the mapping defined in (2.17).

Mapping of a function in $H(\mathbf{curl}, \Omega)$

If the function to be mapped is in $H(\mathbf{curl}, \Omega)$, then it can be shown [97] that:

$$\boldsymbol{v}(\mathbf{x}) = \boldsymbol{J}^{e^{-T}} \boldsymbol{f} \circ \mathbf{\Phi}^{e^{-1}}, \tag{2.19}$$

where $\boldsymbol{v} \in H(\mathbf{curl}, \Omega)$, $\boldsymbol{f} \in H(\mathbf{curl}, \widehat{\mathrm{K}})$, and $\mathbf{\Phi}^{e^{-1}}$ is the inverse of the mapping defined in (2.17).

Mapping of a function in $H(\mathrm{div}, \Omega)$

If the function to be mapped is in $H(\mathrm{div}, \Omega)$, then it can be shown [97] that:

$$\boldsymbol{v}(\mathbf{x}) = \frac{\boldsymbol{J}^e}{\det \boldsymbol{J}^e} \boldsymbol{f} \circ \mathbf{\Phi}^{e^{-1}}, \tag{2.20}$$

where $\boldsymbol{v} \in H(\mathrm{div}, \Omega)$, $\boldsymbol{f} \in H(\mathrm{div}, \widehat{\mathrm{K}})$, and $\mathbf{\Phi}^{e^{-1}}$ is the inverse of the mapping defined in (2.17).

2.3.3 Construction of the global finite element function space

This mapping strategy allows us to construct the basis functions associated to the vertices, edges, faces and/or cells of a given physical mesh element. Thus, by iterating on every element of the mesh, the complete $V(\Omega)$ space is (re)constructed. This approach simplifies drastically the finite element method, since only a few basis functions $\boldsymbol{f}^k \in F(\widehat{\mathrm{K}})$ are needed. Mathematically, an integral on K^e involving the basis functions $\boldsymbol{v}^i \in V(\Omega)$ becomes:

$$\begin{aligned} &\int_{\mathrm{K}^e(\mathbf{x})} \boldsymbol{v}^j(\mathbf{x}) \cdot \boldsymbol{v}^i(\mathbf{x}) \, \mathrm{dK}^e(\mathbf{x}) \\ &= \int_{\widehat{\mathrm{K}}(\mathbf{u})} \left[\boldsymbol{M}^e(\mathbf{u}) \boldsymbol{f}^{l(j)}(\mathbf{u}) \right] \cdot \left[\boldsymbol{M}^e(\mathbf{u}) \boldsymbol{f}^{k(i)}(\mathbf{u}) \right] |\det \boldsymbol{J}^e(\mathbf{u})| \, \mathrm{d}\widehat{\mathrm{K}}(\mathbf{u}), \end{aligned} \tag{2.21}$$

where the function $\boldsymbol{M}^e(\mathbf{u})$ maps the function $\boldsymbol{f}^{l(j)}(\mathbf{u})$ (or $\boldsymbol{f}^{k(i)}(\mathbf{u})$), defined on the reference element $\widehat{\mathrm{K}}(\mathbf{u})$, onto the physical element $\mathrm{K}^e(\mathbf{x})$. This integral on the reference space is called an *elementary* finite element integral. Each integral can be identified by the triplet $\{e, k, l\}$.

As it can be seen in equation (2.21), the basis functions in the reference element have a special indexing: $k(i)$ or $l(j)$. Indeed, depending if we are referring to the physical or reference space, two different sets of indices are needed. In this context, the letters k and l are referring to the indices of the basis functions $\boldsymbol{f}^k \in F(\widehat{\mathrm{K}})$. The letters i and j are still referring to the indices of the basis functions $\boldsymbol{v}^i \in V(\Omega)$. Thus, the notations $k(i)$ or $l(j)$ means that the function $\boldsymbol{f}^k \in F(\widehat{\mathrm{K}})$ will be mapped onto the function $\boldsymbol{v}^i \in V(\Omega)$. Practically, this means that an index mapping between the two spaces will be needed: see section 2.7 for more details. At this stage, let us just note that:

- the indices of the basis functions spanning V are called *global* indices;
- the indices of the basis functions spanning F are called *local* indices.

2.3.4 Elementary integrals

Let us write the elementary integrals associated to the weak formulations (2.11) and (2.12).

Time-harmonic electromagnetic wave formulation

We start by writing the variational formulation of electromagnetic waves (2.11), for a mesh element K^e inside Ω (thus excluding its boundary $\partial\Omega$):

$$\underbrace{\int_{\mathrm{K}^e} \left(\boldsymbol{\mu}_r^{-1}\,\mathbf{curl}\,\boldsymbol{e}^j\right)\cdot\left(\mathbf{curl}\,\boldsymbol{e}^i\right)\,\mathrm{d}\mathrm{K}^e}_{\mathcal{I}^e_{i,j}} - k^2 \underbrace{\int_{\mathrm{K}^e} \left(\tilde{\boldsymbol{\varepsilon}}_r\,\boldsymbol{e}^j\right)\cdot\boldsymbol{e}^i\,\mathrm{d}\mathrm{K}^e}_{\mathcal{J}^e_{i,j}} = 0 \qquad \forall \boldsymbol{e}^i \in V_0(\mathbf{curl},\mathrm{K}^e),$$

where the finite-dimensional space $V_0(\mathbf{curl},\mathrm{K}^e)$ is a subset of $H_0(\mathbf{curl},\mathrm{K}^e)$. In this equation, two terms can be identified:

1. a $\mathbf{curl}\,\boldsymbol{e}^j \cdot \mathbf{curl}\,\boldsymbol{e}^i$ term, that will be mapped in the reference element $\widehat{\mathrm{K}}$ according to (2.20), since the inclusion (2.10);
2. a $\boldsymbol{e}^j \cdot \boldsymbol{e}^i$ term, that will be mapped in the reference element $\widehat{\mathrm{K}}$ according to (2.19).

Applying these mappings leads to the integrals:

$$\begin{cases} \mathcal{I}^e_{i,j} = \displaystyle\int_{\widehat{\mathrm{K}}} \left(\boldsymbol{\mu}_r^{-1}\frac{\boldsymbol{J}^e}{\det\boldsymbol{J}^e}\,\mathbf{curl}\,\boldsymbol{f}^{l(j)}\right)\cdot\left(\frac{\boldsymbol{J}^e}{\det\boldsymbol{J}^e}\,\mathbf{curl}\,\boldsymbol{f}^{k(i)}\right)|\det\boldsymbol{J}^e|\,\mathrm{d}\widehat{\mathrm{K}}, & \text{(2.22a)} \\ \mathcal{J}^e_{i,j} = \displaystyle\int_{\widehat{\mathrm{K}}} \left(\tilde{\boldsymbol{\varepsilon}}_r\,\boldsymbol{J}^{e^{-T}}\boldsymbol{f}^{l(j)}\right)\cdot\left(\boldsymbol{J}^{e^{-T}}\boldsymbol{f}^{k(i)}\right)|\det\boldsymbol{J}^e|\,\mathrm{d}\widehat{\mathrm{K}}, & \text{(2.22b)} \end{cases}$$

which are the elementary integrals of the finite element formulations for time-harmonic electromagnetic waves. The basis functions $\boldsymbol{f}^{l(j)}$ and $\boldsymbol{f}^{k(i)}$ are taken from a finite-dimensional subset of $H_0(\mathbf{curl},\widehat{\mathrm{K}})$, where $\widehat{\mathrm{K}}$ is the reference element, on which K^e is mapped.

Time-harmonic acoustic wave formulation

Now, let us consider the variational formulation of acoustic waves (2.12), for a mesh element K^e inside Ω (thus excluding its boundary $\partial\Omega$):

$$\int_{\mathrm{K}^e} \mathbf{grad}\,p^j\cdot\mathbf{grad}\,p^i\,\mathrm{d}\mathrm{K}^e + k^2\int_{\mathrm{K}^e} p^j\,p^i\,\mathrm{d}\mathrm{K}^e = 0 \quad \forall p^i \in V_0^1(\mathrm{K}^e),$$

where the finite-dimensional space $V_0^1(\mathrm{K}^e)$ is a subset of $H_0^1(\mathrm{K}^e)$. In this equation, two terms can be identified:

1. a $\mathbf{grad}\, p^j \cdot \mathbf{grad}\, p^i$ term, that will be mapped in the reference element $\widehat{\mathrm{K}}$ according to (2.19), since the inclusion (2.9);
2. a $p^j\, p^i$ term, that will be mapped in the reference element $\widehat{\mathrm{K}}$ according to (2.18).

Applying these mappings leads to the integrals:

$$\begin{cases} \displaystyle\int_{\mathrm{K}^e} \mathbf{grad}\, p^j \cdot \mathbf{grad}\, p^i \,\mathrm{dK}^e = \int_{\widehat{\mathrm{K}}} \left(\boldsymbol{J}^{e^{-T}}\, \mathbf{grad}\, f^{l(j)}\right) \cdot \left(\boldsymbol{J}^{e^{-T}}\, \mathbf{grad}\, f^{k(i)}\right) |\det \boldsymbol{J}^e| \,\mathrm{d}\widehat{\mathrm{K}}, & \text{(2.23a)} \\ \displaystyle\int_{\mathrm{K}^e} p^j\, p^i \,\mathrm{dK}^e = \int_{\widehat{\mathrm{K}}} f^{l(j)}\, f^{k(i)}\, |\det \boldsymbol{J}^e| \,\mathrm{d}\widehat{\mathrm{K}}, & \text{(2.23b)} \end{cases}$$

which are the elementary integrals of the finite element formulations for time-harmonic acoustic waves. The basis functions $f^{l(j)}$ and $f^{k(i)}$ are taken from a finite-dimensional subset of $H_0(\widehat{\mathrm{K}})$, where $\widehat{\mathrm{K}}$ is the reference element, on which K^e is mapped.

2.4 Orientation of the reference space

To simplify the presentation of the above mapping strategy, a technical point was left uncovered: the so-called orientation problem. Before going any further, let us introduce a set of new notations and concepts.

2.4.1 Vertices and elements of the mesh

Let us assume that we have access to an enumeration of the mesh elements and vertices of the whole mesh. We denote by V^a the *topological* vertex a of the mesh; and K^e is, as previously, the e^{th} elements of the mesh. For an element composed of M vertices, we also assume that an enumeration of its vertices is available. The notation:

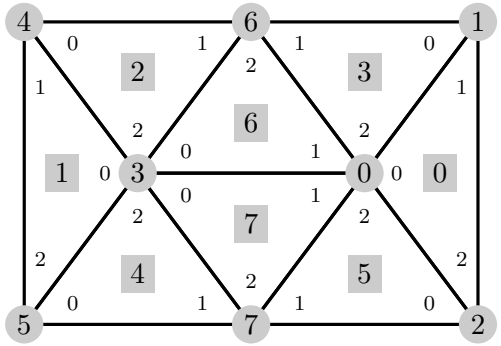
$$\mathbf{K}^e = \underbrace{[a, b, \ldots, q]}_{M},$$

means that element K^e is composed of the vertices $\{V^a, V^b, \ldots, V^q\}$ of the mesh. It is worth stressing that in what follows, only the *topological* vertices have to be considered, even when curved physical elements are treated.

Moreover, we assume that the vertices in an element are given a local index. So, the notation $\mathbf{K}^e(i) = a$, means that the vertex i of element K^e is the vertex a of the mesh, that is the vertex V^a. We call a the global index of V^a, and i its local index with respect to K^e. Finally, Figure 2.3 illustrates all the above notations on a triangular mesh.

2.4.2 Edges of the mesh

An edge of a mesh element is defined by the local indices of the vertices composing it. We denote by $\mathbf{e}^j$ the edge j of a mesh element. The notation e_i^j refers to the vertex i, *in the local vertex indexing of the element*, of edge j.



a is the vertex a of the mesh, that is V^a;
e is the element e of the mesh, that is K^e;

Inside an element,
i is the local index of the nearest vertex;

$\mathbf{K}^0 = [0, 1, 2]$; $\mathbf{K}^1 = [3, 4, 5]$;
$\mathbf{K}^2 = [4, 6, 3]$; $\mathbf{K}^3 = [1, 6, 2]$;
$\mathbf{K}^4 = [5, 7, 3]$; $\mathbf{K}^5 = [2, 7, 0]$;
$\mathbf{K}^6 = [3, 0, 6]$; $\mathbf{K}^7 = [3, 0, 7]$.

Figure 2.3: A triangular mesh and the associated notations.

We consider that the edges are oriented: $\mathbf{e}^j$ starts at e_0^j and ends and e_1^j. For example, in the case of a triangle, we may have: $\mathbf{e}^1 = [e_0^1, e_1^1] = [1, 2]$. This means that edge 1 is composed by the vertices 1 and 2 of the triangle. This edge starts at vertex 1 and ends at vertex 2.

The notation $\mathbf{K}^e(\mathbf{e}^j) = [a, b]$ means that edge j of element K^e is composed by the vertices V^a and V^b. It starts at vertex V^a and ends at vertex V^b.

In this thesis, an edge is oriented thanks to the global indices of its vertices. An edge starts at the vertex with the smallest index, and ends at the vertex with the biggest index. Formally, for edge j of element e we have:

$$\mathbf{e}^j = [e_0^j, e_1^j] \iff \mathbf{K}^e(e_0^j) < \mathbf{K}^e(e_1^j). \tag{2.24}$$

If the orientation of an edge is unknown, or does not matter, the following notation is used:

$$\widetilde{\mathbf{e}}^j = [\widetilde{e}_0^j, \widetilde{e}_1^j]$$

which means that edge j is composed by the vertices with local indices e_0^j and e_1^j, but *the orientation rule* (2.24) *is not necessarily satisfied.*

Finally, let us illustrate these notations using the mesh of Figure 2.3. Since only triangles are considered, we have:

$$\begin{cases} \widetilde{\mathbf{e}}^0 = [\widetilde{0}, \widetilde{1}], \\ \widetilde{\mathbf{e}}^1 = [\widetilde{1}, \widetilde{2}], \\ \widetilde{\mathbf{e}}^2 = [\widetilde{2}, \widetilde{0}], \end{cases}$$

as *local* edge indexing. Now, if we consider element 6, we have:

$$\begin{cases} \mathbf{e}^0 = [1, 0], \\ \mathbf{e}^1 = [1, 2], \\ \mathbf{e}^2 = [0, 2], \end{cases}$$

and

$$\begin{cases} \mathbf{K}^6(\mathbf{e}^0) = [0, 3], \\ \mathbf{K}^6(\mathbf{e}^1) = [0, 6], \\ \mathbf{K}^6(\mathbf{e}^2) = [3, 6]. \end{cases}$$

2.4.3 Faces of the mesh

A face of a mesh element is denoted in the same way than an edge: $\mathbf{f}^j$ is the face j of a given mesh element, and f_i^j is the *local* vertex i of face j.

A face is also oriented. For a triangular face, the notation $[f_0^j, f_1^j, f_2^j]$ means that $\mathbf{f}^j$ starts at vertex f_0^j, goes through vertex f_1^j and ends at vertex f_2^j. For a quadrangular face, the notation $[f_0^j, f_1^j, f_2^j, f_3^j]$ means that $\mathbf{f}^j$ starts at vertex f_0^j and ends at vertex f_3^j. It first goes through vertex f_1^j, and then vertex f_2^j.

As an edge, a face is oriented thanks to the global index of its vertices. If face j of an element K^e is triangular, we have:

$$\mathbf{f}^j = [f_0^j, f_1^j, f_2^j] \iff \mathbf{K}^e(f_0^j) < \mathbf{K}^e(f_1^j) < \mathbf{K}^e(f_2^j). \tag{2.25}$$

If this face is quadrangular, we have:

$$\mathbf{f}^j = [f_0^j, f_1^j, f_2^j, f_3^j] \iff \begin{cases} \mathbf{K}^e(f_0^j) \text{ is the smallest among } \left\{\mathbf{K}^e(f_1^j), \mathbf{K}^e(f_2^j), \mathbf{K}^e(f_3^j)\right\}, & \text{(2.26a)} \\ \mathbf{K}^e(f_1^j) < \mathbf{K}^e(f_3^j), & \text{(2.26b)} \\ f_2^j \text{ is the vertex opposite to } f_0^j. & \text{(2.26c)} \end{cases}$$

If the orientation of a face is unknown, or does not matter, the following notation is used:

$$\widetilde{\mathbf{f}}^j = [\widetilde{f}_0^j, \widetilde{f}_1^j, \widetilde{f}_2^j]$$

for triangular faces, or

$$\widetilde{\mathbf{f}}^j = [\widetilde{f}_0^j, \widetilde{f}_1^j, \widetilde{f}_2^j, \widetilde{f}_3^j] \iff \text{(2.26c)},$$

for quadrangular faces. This means that face j is composed by vertices with local indices f_0^j, f_1^j, f_2^j (and f_3^j), but *the orientation rules* (2.25) *or* (2.26) *are not necessarily satisfied.* However, *in the case of a quadrangular face, we assume that the sub-condition* (2.26c) *is always satisfied.*

Finally, let us illustrate these notations using the mesh of Figure 2.3. Since only triangles are considered, we have:

$$\widetilde{\mathbf{f}}^0 = [\widetilde{0}, \widetilde{1}, \widetilde{2}],$$

as *local* face indexing. Now, if we consider element 6, we have:

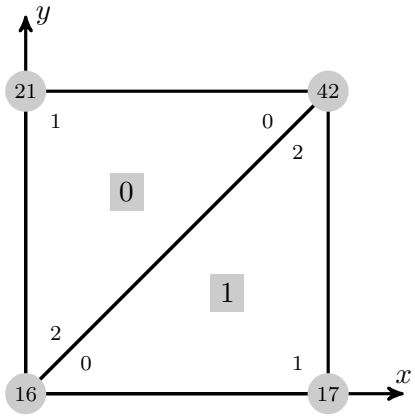
$$\mathbf{f}^0 = [1, 0, 2],$$

and

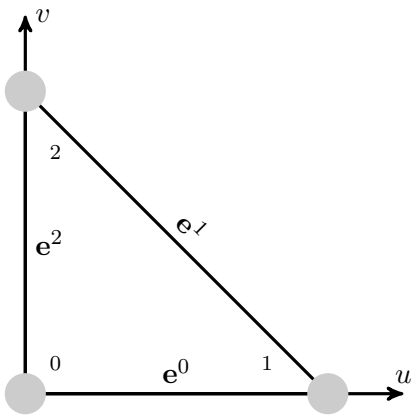
$$\mathbf{K}^6(\mathbf{f}^0) = [0, 3, 6].$$

2.4.4 Illustrations

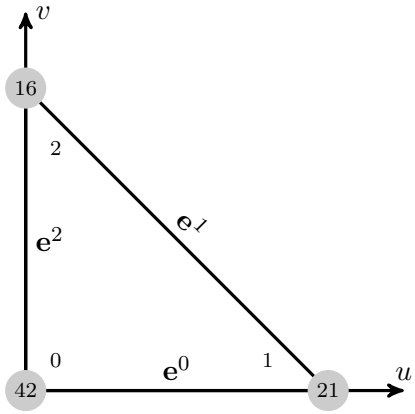
Now that we have introduced the notions of oriented edges and faces, let us illustrate the orientation problem. We consider a square domain $\Omega(x, y)$ meshed by two triangles, as illustrated in Figure 2.4a.



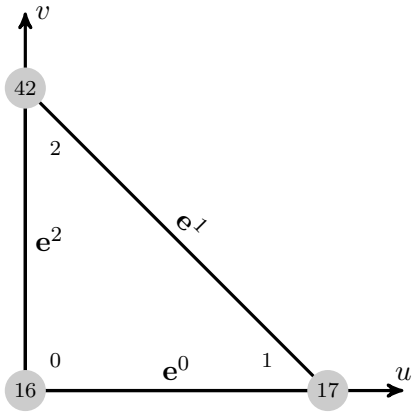
(a) Triangular mesh of a domain $\Omega(x, y)$.



(b) Reference triangle $\widehat{\mathrm{K}}(u, v)$.



(c) Element $\mathrm{K}^0(x, y)$ mapped on $\widehat{\mathrm{K}}(u, v)$.



(d) Element $\mathrm{K}^1(x, y)$ mapped on $\widehat{\mathrm{K}}(u, v)$.

Figure 2.4: Mesh, reference element and mappings.

Our objective is to generate a basis for $V^1(\Omega) \subset H^1(\Omega)$ by using one (or more) basis (bases) defined on the reference triangle $\widehat{\mathrm{K}}(u, v)$ of Figure 2.4b. We require that the resulting function space is continuous across edge $[V^{16}, V^{42}]$. The two triangles of $\Omega(x, y)$ are mapped on $\widehat{\mathrm{K}}(u, v)$, as illustrated at Figures 2.4c and 2.4c.
This problem will be illustrated with a Lagrange basis and a non-Lagrange H^1 basis.

Illustration with a Lagrange basis: an order 3 case

Let us use the following two-dimensional Lagrange basis of order 3, defined on the reference triangle $\widehat{\mathrm{K}}$ of Figure 2.4b.

$$V^1(\widehat{\mathrm{K}}) = \Big[\underbrace{f^{V^0}, f^{V^1}, f^{V^2}}_{\text{Vertex based functions}}, \underbrace{f_0^{\mathbf{e}^0}, f_1^{\mathbf{e}^0}, f_0^{\mathbf{e}^1}, f_1^{\mathbf{e}^1}, f_0^{\mathbf{e}^2}, f_1^{\mathbf{e}^2}}_{\text{Edge based functions}}, \underbrace{f_0^C}_{\text{Cell based function}} \Big]. \tag{2.27}$$

In order to stress the orientation problem, let us consider only the edge contribution of this basis. Since this basis is of order 3, two functions must be defined on each edge. Below are

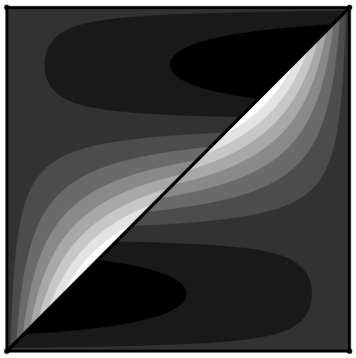
the six edge based functions [137]:

$$\left\{\begin{array}{l}
\left.\begin{array}{l} f_0^{\mathbf{e}^0}(\lambda_0,\lambda_1,\lambda_2) = 4.5\,\lambda_0\,\lambda_1\,(3\,\lambda_0-1), \\ f_1^{\mathbf{e}^0}(\lambda_0,\lambda_1,\lambda_2) = 4.5\,\lambda_0\,\lambda_1\,(3\,\lambda_1-1), \end{array}\right\} \text{ functions associated to edge 0 of } \widehat{\mathrm{K}}, \\[2ex]
\left.\begin{array}{l} f_0^{\mathbf{e}^1}(\lambda_0,\lambda_1,\lambda_2) = 4.5\,\lambda_1\,\lambda_2\,(3\,\lambda_1-1), \\ f_1^{\mathbf{e}^1}(\lambda_0,\lambda_1,\lambda_2) = 4.5\,\lambda_1\,\lambda_2\,(3\,\lambda_2-1), \end{array}\right\} \text{ functions associated to edge 1 of } \widehat{\mathrm{K}}, \\[2ex]
\left.\begin{array}{l} f_0^{\mathbf{e}^2}(\lambda_0,\lambda_1,\lambda_2) = 4.5\,\lambda_2\,\lambda_0\,(3\,\lambda_2-1), \\ f_1^{\mathbf{e}^2}(\lambda_0,\lambda_1,\lambda_2) = 4.5\,\lambda_2\,\lambda_0\,(3\,\lambda_0-1), \end{array}\right\} \text{ functions associated to edge 2 of } \widehat{\mathrm{K}},
\end{array}\right.$$

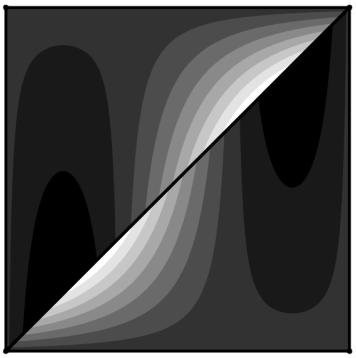
with

$$\left\{\begin{array}{lcl} \lambda_0(u,v) &=& 1-u-v, \\ \lambda_1(u,v) &=& u, \\ \lambda_2(u,v) &=& v. \end{array}\right.$$

To begin with, let us use this basis $V^1(\Omega)$ for both element K^0 and K^1 of Figure 2.4a. Figure 2.5 illustrates the contribution of each element to edge $[V^{16}, V^{42}]$. It can be clearly seen, that the resulting function space is not continuous across $[V^{16}, V^{42}]$.



(a) Contributions of the first functions associated to edge $[V^{16}, V^{42}]$.



(b) Contributions of the second functions associated to edge $[V^{16}, V^{42}]$.

Figure 2.5: Contribution of K^0 and K^1 on edge $[V^{16}, V^{42}]$ when using basis (2.27) for both elements.

Since Lagrange functions have the property to be equal to one at one point, an zero at another set of points, it is worth looking at which point the functions of $V^1(\Omega)$ are associated, as shown in Figure 2.6. By looking at this figure, it is possible to understand the problem. In the reference element $\widehat{\mathrm{K}}$, when traveling from vertex V^{16} to vertex V^{42}, the functions $f_0^{\mathbf{e}_2}$ and $f_1^{\mathbf{e}_2}$ are not taken in the same order in K^0 and K^1. This problem appears, because the definition of the basis functions does not take into account the orientation of the edges of K^0 and K^1, as defined in (2.24).

Let us orient the mapped triangle in Figures 2.4c and 2.4d according to (2.24). Figure 2.7 illustrates this orientation process. By looking at Figure 2.7, it is clear that the two following

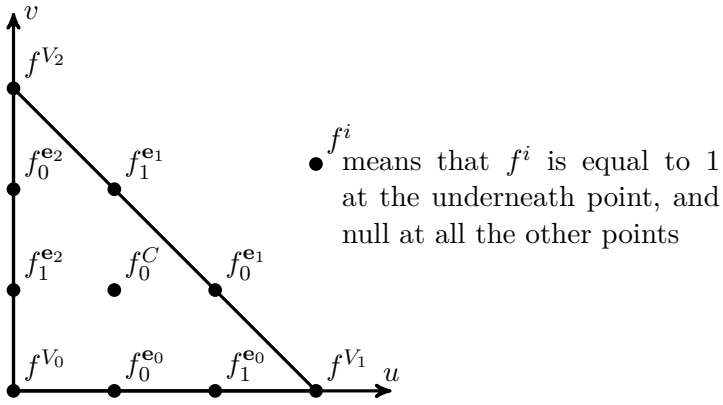
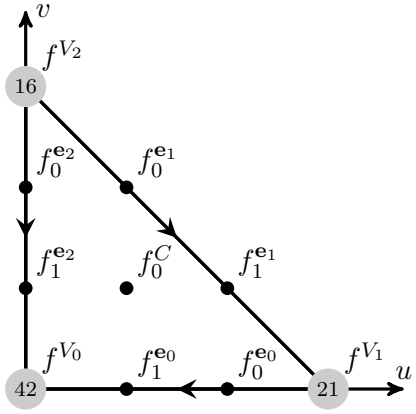
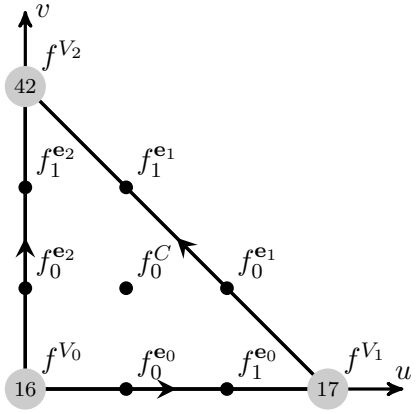


Figure 2.6: Lagrange points and the associated functions in a reference triangle $\widehat{\mathrm{K}}$.



(a) Oriented element K^0 in the reference element $\widehat{\mathrm{K}}$.



(b) Oriented element K^1 in the reference element $\widehat{\mathrm{K}}$.

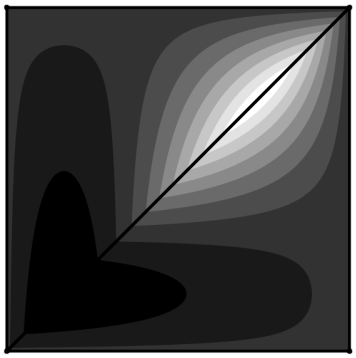
Figure 2.7: Orientation of the triangle K^0 and K^1 in the reference triangle $\widehat{\mathrm{K}}$, with associated Lagrange functions.

bases must be used to generate a continuous function space across $[V^{16}, V^{42}]$:

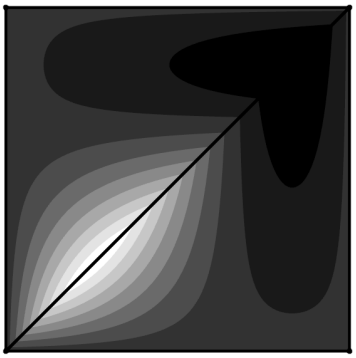
$$\begin{cases} V^1(\widehat{\mathrm{K}})\Big|^0 = \left[f^{V_0}, f^{V_1}, f^{V_2}, f_1^{\mathbf{e}_0}, f_0^{\mathbf{e}_0}, f_1^{\mathbf{e}_1}, f_0^{\mathbf{e}_1}, f_0^{\mathbf{e}_2}, f_1^{\mathbf{e}_2}, f_0^C\right], & \text{(2.28a)} \\ V^1(\widehat{\mathrm{K}})\Big|^1 = \left[f^{V_0}, f^{V_1}, f^{V_2}, f_0^{\mathbf{e}_0}, f_1^{\mathbf{e}_0}, f_0^{\mathbf{e}_1}, f_1^{\mathbf{e}_1}, f_1^{\mathbf{e}_2}, f_0^{\mathbf{e}_2}, f_0^C\right], & \text{(2.28b)} \end{cases}$$

where basis $V^1(\widehat{\mathrm{K}})\Big|^0$ is used on element K^0, and basis $V^1(\widehat{\mathrm{K}})\Big|^1$ on element K^1. The contribution of each element on edge $[V^{16}, V^{42}]$ is shown in Figure 2.8. The function space is now continuous across $[V^{16}, V^{42}]$.

Before concluding this section, it is worth mentioning one last point. The bases, as defined in (2.27) and (2.28), share the same functions. They only differ by the *order* in which these functions are taken. As we will see later, this property will unfortunately not be met in other



(a) Contributions of the first functions associated to edge $[V^{16}, V^{42}]$.



(b) Contributions of the second functions associated to edge $[V^{16}, V^{42}]$.

Figure 2.8: Contribution of K^0 and K^1 on edge $[V^{16}, V^{42}]$ when using two bases (2.28).

kinds of bases.

Illustration with a Lagrange basis: order 1 and 2 cases

Let us now consider an easier case: a second-order one. This time, we use the following basis for both elements K^0 and K^1:

$$V^1(\widehat{\mathrm{K}}) = \left[f^{V_0}, f^{V_1}, f^{V_2}, f_0^{\mathbf{e}_0}, f_0^{\mathbf{e}_1}, f_0^{\mathbf{e}_2} \right],$$

Since we have only one function per edge, which is even, it makes no sense to permute the functions of an edge according to its orientation.

This orientation procedure has also no meaning in the first order case. Indeed, these bases have no edge based or face based functions. Actually, for the order 1 and 2 cases, applying the orientation rules (2.24), (2.25) and (2.26) is useless.

Illustration with a Lagrange basis: conclusion

As we saw above, the orientation of Lagrange bases is quite an easy procedure. Only one set of functions needs to be generated. Depending on the actual orientation of the edges and faces of an element, the functions are just swapped. In order to enforce continuity, the orientation rules given in (2.24),(2.25) and (2.26) can be used. We also saw that the orders 1 and 2 do not need a such procedure. We illustrated the orientation problem in the case of a triangular mesh, but this procedure can be extended to tetrahedra, quadrangles, hexahedra, and so on.

Lagrange functions are widely used in finite element simulations. However, these bases cannot be used in every situation. For example, $H(\mathbf{curl})$ or $H(\mathrm{div})$ spaces can be more suited for some finite element formulations [19], because of their continuity properties. Unfortunately, these spaces cannot be spanned by a Lagrange basis.

Illustration with a non-Lagrange basis: an H^1 basis of order 3

Let us now consider a Legendre H^1 basis [136]. This basis is quite advantageous compared to a Lagrange basis. When increasing the polynomial order from p to $p+1$, the older polynomials of order p are still valid, and one needs to compute only the polynomials of order $p+1$. This is not possible for a Lagrange basis, since new Lagrange points are introduced thus invalidating the polynomials of order p. For this reason, the Legendre basis is called hierarchical. Let us first define the two following functions:

$$\begin{cases} \ell_n^S(x,t) &= t^n\,\ell_n\left(\dfrac{x}{t}\right) & \text{for } x\in[-t,t] \text{ and } t\in\,]0,1], \\ L_n^s(x,t) &= \displaystyle\int_{-t}^{x} \ell_{n-1}^s(\tau,t)\,\mathrm{d}\tau & \text{for } n\geq 2, \end{cases}$$

where $\ell_n(x)$ is the Legendre function of order n defined over $[-1,1]$.

As in the Lagrange case, let us try to generate a continuous function space across edge $[V^{16},V^{42}]$ of Figure 2.4a. Before choosing the basis functions for K^0 and K^1, let us look on how the edge based functions are defined in this Legendre basis:

$$f_j^{\mathbf{e}^i}(u,v) = L_{j+1}^S\Big(\lambda_{e_1^i}(u,v)-\lambda_{e_0^i}(u,v),\lambda_{e_1^i}(u,v)+\lambda_{e_0^i}(u,v)\Big) \quad \forall i\in\{0,1,2\} \text{ and } \forall j\in\{0,1\},$$

where

$$\begin{cases} \lambda_0(u,v) &= 1-u-v, \\ \lambda_1(u,v) &= u, \\ \lambda_2(u,v) &= v. \end{cases}$$

As we can see, for this kind of basis, the orientation is directly taken into account in the definition of the edge based functions. Indeed, a function $f^{\mathbf{e}^i}$, associated to edge i, is parametrized by the local indices of the vertices composing this edge. So, two functions can by generated for a given $f_j^{\mathbf{e}^i}$, and the right one can be chosen by using (2.24).

It is clear that the swapping strategy for orienting Lagrange basis cannot be used any more. Indeed, depending on the orientation, $f_j^{\mathbf{e}^i}$ will have different expressions. Thus, unlike Lagrange bases, more than one set of basis functions needs to be generated: one per possible orientation.

This lack of symmetry of non-Lagrange bases can be easily shown on a one-dimensional case of order 3. Table 2.1 compares a Lagrange and a Legendre basis. It can be directly seen that Lagrange functions can be swapped, while it is impossible for Legendre functions.

Illustration with a non-Lagrange basis: $H(\mathbf{curl})$ and $H(\mathrm{div})$ spaces

For the case of $H(\mathbf{curl})$ and $H(\mathrm{div})$ spaces, it is also possible to find parametrized basis functions, as proposed by [136]. So, as for the previous case, multiple sets of functions need to be generated: one per possible orientation. It is worth noticing, that the orientation problem appears even at the first-order in $H(\mathbf{curl})$ and $H(\mathrm{div})$ bases.

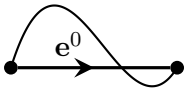
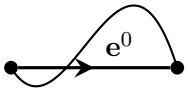
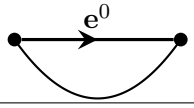
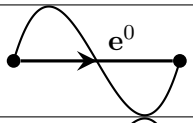
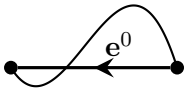
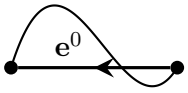
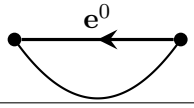
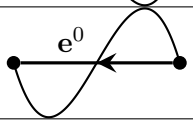
Orientation	Lagrange		Legendre	
	$f_0^{\mathbf{e}^0}$	$f_1^{\mathbf{e}^0}$	$f_0^{\mathbf{e}^0}$	$f_1^{\mathbf{e}_0}$
Left – Right				
Right – Left				

Table 2.1: Comparison between the edge contributions of a Lagrange basis and a Legendre basis on an order 3 line element.

Illustration with a non-Lagrange basis: the sign flip strategy

In the example presented above, another orienting strategy could have been used: the sign flip. Indeed, all the problems we had, can be solved by flipping the sign of a function, associated to an edge that has the wrong orientation.

Unfortunately, this approach works only for orienting the edges, since an edge can only go in two directions. In three-dimensional cases, where face based functions need to be used, it becomes impossible to solve the orientation problem with only a sign flip, as pointed out in [1]. In those cases, it becomes mandatory to use more than one set of functions.

2.5 Basic algorithms: orienting an edge or a face

Now that we understand the orientation problem, let us propose some fundamental implementations, that will be used in the remaining of this thesis.

The first step to determine a correctly oriented basis, is to correctly orient a given edge or face. That is, for a given edge $\widetilde{\mathbf{e}}^j = [\widetilde{e}_0^j, \widetilde{e}_1^j]$, finding whether $\mathbf{e}^j = [e_0^j, e_1^j]$ or $\mathbf{e}^j = [e_1^j, e_0^j]$ satisfy (2.24). Or, for a given face $\widetilde{\mathbf{f}}^j = [\widetilde{f}_0^j, \widetilde{f}_1^j, \widetilde{f}_2^j, (\widetilde{f}_3^j)]$, which *permutation* of $\{f_0^j, f_1^j, f_2^j, (f_3^j)\}$ satisfies (2.25) or (2.26).

Let $\mathbf{a}$ be a vector, and $\mathbf{b}$ a sorted version of vector $\mathbf{a}$. We call permutation vector, the vector $\mathbf{p}$ such that: $\mathbf{a}(\mathbf{p}) = \mathbf{b}$. The function `sort` implements this procedure. It takes a vector $\mathbf{a}$ as input, sorts this vector in-place (that is $\mathbf{a}$ becomes $\mathbf{b}$), and returns the corresponding permutation vector $\mathbf{p}$.

2.5.1 Edge and face orientation

With this permutation vector in hand, orienting an edge or a triangular face is an easy task:

1. we form the vector $\mathbf{K}^e(\widetilde{\mathbf{e}}^j)$ for an edge, or the vector $\mathbf{K}^e(\widetilde{\mathbf{f}}^j)$ for a face;
2. we sort this vector in ascending order, and we keep the permutations vector $\mathbf{p}$;
3. the oriented edge is given by $\mathbf{e}^j = \widetilde{\mathbf{e}}^j(\mathbf{p})$, and the oriented triangular face is given by $\mathbf{f}^j = \widetilde{\mathbf{f}}^j(\mathbf{p})$.

Because of the sorting procedure, condition (2.24), or (2.25), is directly satisfied.

In the case of a quadrangular face, a *correction* needs to be applied to $\widetilde{\mathbf{f}}^j(\mathbf{p})$, so that we can guarantee that condition (2.26c) remains satisfied (the sorting procedure may have altered it).

Let us define $\widehat{\mathbf{f}}^j$ as $\widehat{\mathbf{f}}^j = \widetilde{\mathbf{f}}^j(\mathbf{p})$. Because of the sorting procedure, it is clear that $\widehat{\mathbf{f}}^j$ verifies both (2.26a) and (2.26b). To impose (2.26c), we need to swap the element $\widehat{\mathbf{f}}^j(2)$ with the element associated to the vertex opposite to $\widehat{\mathbf{f}}^j(0)$. This element must be in the sub-vector $\widehat{\mathbf{f}}^j(1:3)$[10]. The vector resulting from this swapping is actually $\mathbf{f}^j$, the oriented quadrangular face. Indeed, doing so will impose that:

$$\begin{cases} \mathbf{f}^j(2) \text{ is the vertex opposite to } \mathbf{f}^j(0), & (2.29) \\ \mathbf{f}^j(3) \text{ is the vertex opposite to } \mathbf{f}^j(1), & (2.30) \end{cases}$$

thus verifying (2.26c). Moreover, this swapping did not alter the validity of condition (2.26a), since the swapping occurred in the sub-vector $\widehat{\mathbf{f}}^j(1:3)$, which is such that:

$$\mathbf{K}^e\Big[\widehat{\mathbf{f}}^j(0)\Big] < \mathbf{K}^e\Big[\widehat{\mathbf{f}}^j(1:3)\Big].$$

Furthermore, condition (2.26b) remains also valid. Indeed:

1. if the swapping occurred between $\widehat{\mathbf{f}}^j(1)$ and $\widehat{\mathbf{f}}^j(2)$, we still have
$$\mathbf{K}^e\Big[\widehat{\mathbf{f}}^j(1:2)\Big] < \mathbf{K}^e\Big[\widehat{\mathbf{f}}^j(3)\Big];$$
2. if the swapping occurred between $\widehat{\mathbf{f}}^j(2)$ and $\widehat{\mathbf{f}}^j(3)$, we still have
$$\mathbf{K}^e\Big[\widehat{\mathbf{f}}^j(1)\Big] < \mathbf{K}^e\Big[\widehat{\mathbf{f}}^j(2:3)\Big].$$

Algorithms 3 and 4 illustrate the above procedures.

[10] If the element is $\widehat{\mathbf{f}}^j(2)$, it means that $\widehat{\mathbf{f}}^j = \mathbf{f}^j$.

```
Vector<int> orientEntity(Vector<int> vIndex, Vector<int> entity)
  Description
    Let vIndex[] be the global vertex indices of a mesh element.
    And let entity[] be the non-oriented local vertex indices of an edge
    or face of this element, that is ẽ^j or f̃^j.

    This function returns a vector v[], which is a reordered copy of
    entity[], that satisfies the orientation rules (2.24), (2.25)
    or (2.26).
  Implementation (C++)
    // Build a vector with the vertex global indices of the given entity
    Vector<int> entityVertexGlobalIndex;
    for(int v = 0; v < entity.size(); v++)
      entityVertexGlobalIndex[v] = vIndex[entity[v]];

    // Sort it from smaller to bigger and keep permutation
    Vector<int> vertexPermutation = sort(entityVertexGlobalIndex);

    // Swap entity local indices
    Vector<int> orientedEntity;
    for(int v = 0; v < entity.size(); v++)
      orientedEntity[v] = entity[vertexPermutation[v]];

    // Apply quadrangular face correction if needed
    if(entity.size() == 4)
      quadFaceCorrection(orientedEntity, entity);       // See algorithm 4

    // Done
    return orientedEntity;
```

Algorithm 3: Entity (edge or face) orientation algorithm.

```
void quadFaceCorrection(Vector<int> faceToCorrect, Vector<int>
faceReference)
    Description
        Let faceReference[] be the non-oriented local vertex indices f̃^j of a
        quadrangular face of an element.
        Let faceToCorrect[] be the non-oriented local vertex indices f̂^j of
        the same quadrangular face, that is satisfying (2.26a) and (2.26b).

        This procedure modifies faceToCorrect[] in order to satisfy (2.26).
    Implementation (C++)
        // First, find the local index in faceReference[] equaling faceToCorrect[0]
        int indexZero = 0;
        while(faceReference[indexZero] != faceToCorrect[0])
            indexZero++;

        /* Then go 2 indices further in faceReference[] (modulo 4):          */
        /* this is the index of the vertex that should be at the opposite    */
        /* of the vertex index faceToCorrect[0]                              */
        int oppositeVertex = faceReference[(indexZero + 2) % 4];

        // Check if vertex 2 is opposite to vertex 0 in faceToCorrect[]...
        if(faceToCorrect[2] != oppositeVertex)
            /* ...  if not, find oppositeVertex in faceToCorrect[],          */
            /* and swap its position with position 2                         */
            int position = 0;
            while(faceToCorrect[position] != oppositeVertex)
                position++;

            faceToCorrect.swap(2, position);
```

Algorithm 4: Quadrangular face orientation correction algorithm.

2.6 Constructing an oriented basis for a given mesh element

Now that we know how to orient a given edge or face, we can orient a whole element and construct a basis for it. To do so, we first need to iterate on all the edges and faces of the given element. For each entity (edge or face), an orientation can be computed using Algorithm 3. Since this algorithm takes as input the vectors $\widetilde{\mathbf{e}}^j$ or $\widetilde{\mathbf{f}}^j$, we need a way to access them.

In this thesis we assume that an element is represented by a object of the **Element** class. An **Element** implements the following methods:

1. **getIndex** returning the index of this element in the mesh;
2. **getVertexIndex** returning a **Vector<int>** v, such that v[i] is the i$^{\text{th}}$ global vertex index of the **Element**;
3. **getEdgeIndex** returning a **Vector<Vector<int>>** e, such that e[j][] is the $\widetilde{\mathbf{e}}^j$ of the **Element**;
4. **getFaceIndex** returning a **Vector<Vector<int>>** f, such that f[j][] is the $\widetilde{\mathbf{f}}^j$ of the **Element**.

With these methods in hand, implementing a function constructing an oriented basis for a given element is straightforward, as shown in Algorithm 5.

```
Basis getOrientedBasis(Element element)
  Description
    This function returns the Basis associated to the given Element.
  Implementation (C++)
    // Get global vertex indices of the given Element
    Vector<int> vIndex = element.getVertexIndex();

    // Get vectors ẽ^j and f̃^j for every edge and face
    Vector<Vector<int>> unorientedEdge = element.getEdgeIndex();
    Vector<Vector<int>> unorientedFace = element.getFaceIndex();

    // Orient every edge
    Vector<Vector<int>> orientedEdge;
    for(int e = 0; e < unorientedEdge.size(); e++)
      orientedEdge[e] = orientEntity(vIndex, unorientedEdge[e]);

    // Orient every face
    Vector<Vector<int>> orientedFace;
    for(int f = 0; f < unorientedFace.size(); f++)
      orientedFace[f] = orientEntity(vIndex, unorientedFace[f]);

    /* Generate a basis using the oriented edges and faces ...          */
    /* For instance, rules derived in [136] can be used.                */

    // Once done, return the constructed basis
    return orientedBasis;
```

Algorithm 5: Orientation of a basis for a given element.

2.7 Assembly procedure

We now have everything in hand to construct the finite element linear system (2.15). Algorithm 7 presents the classical approach to construct the linear system. It takes three arguments:

1. a vector of elements;
2. an N by N zero matrix;
3. an object **FeTerm**, that can compute the elementary integrals (2.22) or (2.23), through a method **FeTerm::get**, for a given triplet $\{e, k, l\}$;

and populates the given matrix with the finite element integrals. This procedure is often called the *assembly* of the FE linear system. A possible implementation of the **FeTerm::get** method, handling oriented basis, is given by Algorithm 6.

```
scalar FeTerm::get(Element e, int k, int l)
  Description
    This method computes the integrals (2.22) or (2.23) for a given
    triplet {e, k, l}.
  Implementation (C++)
    // Get oriented basis for Element e
    Basis F(K̂) = getOrientedBasis(e);
    // Compute elementary integral associated to the triplet {e, k, l}
    return ∫_K̂ [M^e f^l] · [M^e f^k] |det J^e| dK̂    f^k ∈ F(K̂) ;
```

Algorithm 6: And example of **FeTerm::get** method.

As explained before, it is not necessary to evaluate every possible entry of the FE matrix. Indeed, since the basis functions have a limited non-zero domain, only a few integrals in (2.15) are non-zero. Using the mapping strategy proposed in the last section, the assembly algorithm is extremely simple.

We first start by iterating on the mesh elements, and for every element, we take all the degrees of freedom associated to its vertices, edges, faces and cell. This is done by a **takeDof** function taking a mesh element and returning a vector of degrees of freedom. These degrees of freedom are represented algorithmicaly by a **Dof** structure, enabling the *unique* identification of a DoF. It is the algorithmic counterpart of the i or j *global* index in the integrals (2.21). The conversion from a **Dof** to a global index is accomplished by the **getGlobalId** function. This last takes a **Dof** as input and returns its corresponding global index. It is worth mentioning that the vector returned by **takeDof** has also an important property. The order in which the degrees of freedom are stored correspond to the *local* indexing: that is the k or l index used in (2.21). In other words, the k^{th} entry of this vector corresponds to the DoF associated to the k^{th} basis function in the reference space $F(\widehat{\mathrm{K}})$.

We now have access to all the $\{e, k, l\}$ triplets for computing the elementary integrals (2.21), and we have also a procedure to map a *local* index k or l onto a global index i or k. So, finally, we just need to compute all these integrals, and to sum them in the FE matrix at the

appropriate entry, as shown in Algorithm 7.

```
void assemble(Vector<Element> element, Matrix A, FeTerm feTerm)
  Description
    This function assembles the given matrix A related to the finite
    element formulation feTerm.  This formulation is defined on the mesh
    given by the Vector of Element element.
  Implementation (C++)
    // Iterate on all mesh elements
    for(int e = 0; e < element.size(); e++)
      // DoFs associated to element e (with local indexing of e)
      Vector<Dof> dof = takeDof(element[e]);

      // Iterate on all pairs of DoFs
      for(int k = 0; k < dof.size(); k++)
        for(int l = 0; l < dof.size(); l++)
          // Get global indexes of DoFs k and l
          int globalI = getGlobalId(dof[k]);
          int globalJ = getGlobalId(dof[l]);

          // Assemble the local contribution of DoFs k and l of element e
          A[globalI, globalJ] += feTerm.get(element[e], k, l);
```

Algorithm 7: Finite element assembly procedure.

Finally, let us note that this algorithm can be parallelized with a thread-based paradigm: each thread is assigned a portion of the elements to assemble. It is worth mentioning that mutual exclusion is required, when more than one thread is accessing a given entry in the matrix.

Chapter 3

Finite element assembly through efficient numerical quadratures

In this chapter, a finite element assembly strategy, relying on efficient quadratures, will be proposed. In the classical assembly algorithm, presented in chapter 2, the elementary integrals were computed on-the-fly through the `FeTerm::get` method. In the proposed improvement, these integrals will be *pre-computed* using computationally efficient matrix-matrix products. The `FeTerm::get` method will then just fetch the requested elementary terms.

3.1 Quadrature rules

Before entering the core of this chapter, let us recall some fundamentals on quadrature rules. Let $p(\mathbf{x})$ be a polynomial function of order n defined over $\widehat{\mathrm{K}} \in \mathbb{R}^3$. It is well known [131], that the integral

$$I = \int_{\widehat{\mathrm{K}}} p(\mathbf{x}) \, \mathrm{d}\widehat{\mathrm{K}},$$

can be computed *exactly* with the following weighted sum, called a *quadrature*:

$$I = \sum_{g=1}^{G} w_g \, p(\mathbf{x}_g) = \sum_{g=1}^{G} w_g \, p\Big|_g, \tag{3.1}$$

where the $w_g \in \mathbb{R}$ are well chosen weights, where the $\mathbf{x}_g \in \widehat{\mathrm{K}}$ are well chosen evaluation points, and where $G \geq 1$ is sufficiently large. Regarding the last equality, to simplify the notations, we consider that $p(\mathbf{x}_g) = p\Big|_g$. Multiple strategies, called *quadrature rules*, can be used to derive w_g, $\mathbf{x}_g$ and G. In this thesis, the quadratures are taken from [131].

Finally, let us notice that this approach is well-suited for the finite element method, since the solution is decomposed into a *polynomial* basis. Thus, *exactly* computing the elementary integrals (2.21) is possible and easy.

3.2 Performance of the classical assembly algorithm when using high-order finite elements

Let us now test the performances of the assembly Algorithm 7, with the elementary integrals (2.21) computed on-the-fly, by using Algorithm 6 and an appropriate quadrature rule.

As test case, we consider a three-dimensional metallic waveguide, truncated with a Silver-Müller radiation condition, and excited at $k = 25\,[\mathrm{rad/m}]$ with a $\mathrm{TE}_{1,1}$ mode. The waveguide is 2λ long, λ being the wavelength, and has a rectangular cross section of $\lambda \times \lambda$. The geometry is meshed with 8 tetrahedra per wavelength.

The Maxwell's equation (1.5) is solved using the finite element method. The assembly Algorithm 7 is used and is parallelized using OpenMP directives. The resulting linear system is solved using the MUMPS direct solver. Simulations are carried out on a Intel Xeon `E5645` using six threads, and timings are available in Figure 3.1.

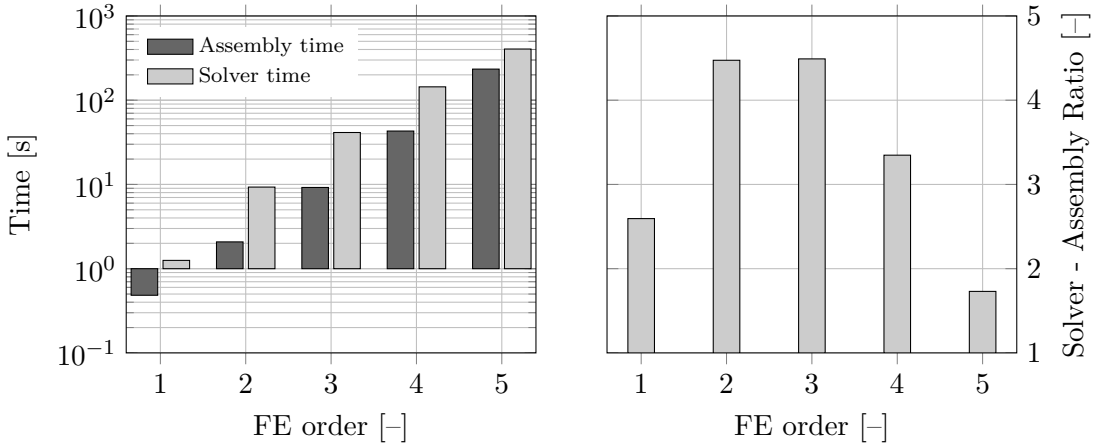


Figure 3.1: High-order FE simulations of a waveguide: timings and ratios.

Based on these results, two conclusions can be drawn. First, the assembly and solver times increase as the FE order is increased, which is expected since the number of unknowns is increased, as shown in Figure 3.2. Secondly, and probably less expected, the ratio between the solver time and the assembly time decreases with the FE order. In other words, the assembly time becomes a larger fraction of the overall time, as the FE order increases.

By analyzing deeper the assembly procedure, we can notice that increasing the FE basis order has a double impact on the computation time:

1. each element will have more associated DoFs, increasing thus the number of elementary integrals to compute;
2. the numerical quadrature will require more points, since higher-order polynomials are used, slowing thus the evaluation of each elementary integral.

To give some order of magnitude, Table 3.1 shows the number of elementary integrals (2.22b) needed on a reference tetrahedron, and the required number of quadrature points.

To mitigate the assembly time increase, with respect to an augmentation of the FE order,

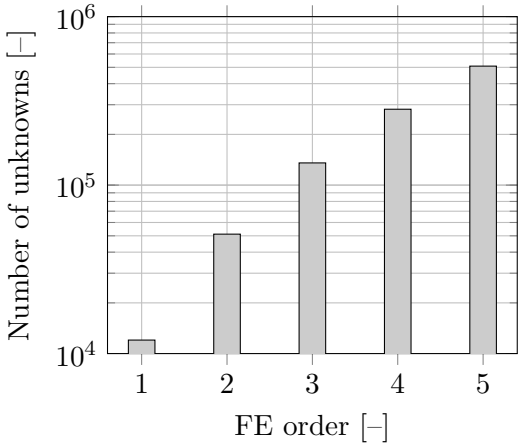


Figure 3.2: High-order FE simulations of a waveguide: evolution of the unknowns number.

Curl-conforming basis order	Integration points	Elementary integrals
1	4	144
2	11	900
3	24	3600
4	43	11025

Table 3.1: Number of elementary integrals (2.22b), and required integration points, for a tetrahedral reference element.

a computationally efficient variation of Algorithm 6 is needed.

3.3 Efficient quadrature using matrix operations

The key idea of a fast quadrature procedure, is to compute all the elementary integrals, associated to each physical mesh element, using matrix-matrix products, as proposed in [64, 65, 83] for discontinuous Galerkin schemes, with high-order nodal Lagrange elements.

Indeed, matrix operations exhibit an excellent cache reuse [74], and are standardized in the Basic Linear Algebra Subprograms (BLAS) interface [18, 38, 39, 84]. Moreover, highly optimized and parallel implementations are available for modern multi-core architectures (*e.g.* MKL[1] [71], OpenBLAS[2] [134] or ATLAS[3] [133]). As already stated in the introduction of this thesis, the OpenBLAS implementation will be used in this work.

In order to use this approach, the elementary integrals of (2.22) or (2.23) have to be rewritten. It is worth noticing that these integrals can be classified as follow:

1. product of H^1 functions, as in the integral (2.23b);
2. product of $H(\mathbf{curl})$ functions, as in the integrals (2.23a) and (2.22b);

[1]Available at: `https://software.intel.com/en-us/intel-mkl`.
[2]Available at: `http://www.openblas.net`.
[3]Available at: `http://math-atlas.sourceforge.net`.

3. product of $H(\mathrm{div})$ functions, as in the integral (2.22a);
4. product of a known function by an $H(\mathbf{curl})$ function, which can, for instance, be used to introduce an imposed current density in the electromagnetic wave equation (1.5).
5. product of a known function by an H^1 function, which can, for instance, be used to introduce volume forces in the acoustics equation (1.17).

In the following subsections, each type of integral will be reformulated as a matrix-matrix product.

3.3.1 Products of H^1 functions

Let us start with the H^1 function space case. Let $\mathcal{I}^e_{k,l}$ be the elementary integral associated to the element e and the *local* degrees of freedom k and l:

$$\mathcal{I}^e_{k,l} = \int_{\widehat{\mathrm{K}}} \left(\alpha f^l\right)\left(f^k\right)|\det \boldsymbol{J}^e| \, \mathrm{d}\widehat{\mathrm{K}},$$

where α is function leading to a more general version of the integral (2.23b). This function can be used, for instance, to take into account a non-uniform distribution of the speed of sound in (1.17). This integral can be evaluated thanks to a numerical quadrature:

$$\begin{aligned}
\mathcal{I}^e_{k,l} &= \sum_{g=1}^{G} w_g \, \alpha\Big|_g f^l\Big|_g f^k\Big|_g |\det \boldsymbol{J}^e|\Big|_g, \\
&= \sum_{g=1}^{G} \left(\alpha\Big|_g |\det \boldsymbol{J}^e|\Big|_g\right)\left(w_g \, f^l\Big|_g f^k\Big|_g\right), \\
&= \sum_{g=1}^{G} \mathbf{B}\Big[e,g\Big]\mathbf{C}\Big[g,\{k,l\}\Big], \qquad (3.2a) \\
&:= \mathbf{D}\Big[e,\{k,l\}\Big], \qquad (3.2b)
\end{aligned}$$

where the operation $\{\cdot,\cdot\}$ takes two indices and combine them into a new unique one.

From equations (3.2), we can notice that each $\mathcal{I}^e_{k,l}$ elementary integral is an element of a matrix $\mathbf{D}$ at line e and column $\{k,l\}$. Moreover, this matrix $\mathbf{D}$ is constructed by the product of two other matrices, $\mathbf{B}$ and $\mathbf{C}$, defined as follow:

$$\begin{cases}
\mathbf{B}\Big[e,g\Big] = \alpha\Big|_g |\det \boldsymbol{J}^e|\Big|_g, & (3.3a) \\
\mathbf{C}\Big[g,\{k,l\}\Big] = w_g \, f^l\Big|_g f^k\Big|_g. & (3.3b)
\end{cases}$$

Figure 3.3 illustrates the structure of these $\mathbf{B}$ and $\mathbf{C}$ matrices. It is worth noticing that matrix $\mathbf{B}$ stores only mesh (*i.e.* metric) dependent data, while matrix $\mathbf{C}$ does not. The latter is defined only on the reference space $\widehat{\mathrm{K}}$.

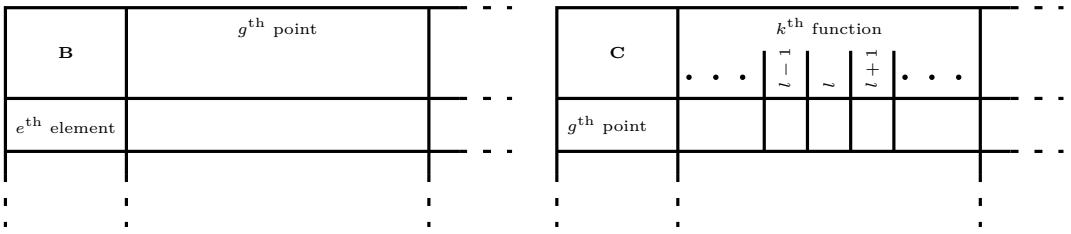


Figure 3.3: Structure of matrices **B** and **C** for an elementary integral composed by the product of H^1 functions.

3.3.2 Products of $H(\mathbf{curl})$ functions

Now, let us consider the $H(\mathbf{curl})$ function space case. Let $\mathcal{I}^e_{k,l}$ be the elementary integral associated to the element e and the *local* degrees of freedom k and l:

$$\mathcal{I}^e_{k,l} = \int_{\widehat{\mathrm{K}}} \left(\underbrace{\tilde{\boldsymbol{\varepsilon}}_r \, \boldsymbol{J}^{e^{-T}}}_{\mathbf{L}^{e^{-T}}} \boldsymbol{f}^l \right) \cdot \left(\boldsymbol{J}^{e^{-T}} \boldsymbol{f}^k \right) |\det \boldsymbol{J}^e| \, \mathrm{d}\widehat{\mathrm{K}},$$

where, for convenience, the product of the permittivity tensor $\tilde{\boldsymbol{\varepsilon}}_r$ and the Jacobian matrix $\boldsymbol{J}^{e^{-T}}$ is defined as the matrix $\mathbf{L}^{e^{-T}}$. Now, let us express the matrix-vector products in a per element fashion:

$$\begin{aligned}
\mathcal{I}^e_{k,l} &= \int_{\widehat{\mathrm{K}}} \sum_{a=1}^{3} \left(\sum_{b=1}^{3} L^{e^{-T}}_{a,b} \, f^l_b \right) \left(\sum_{c=1}^{3} J^{e^{-T}}_{a,c} \, f^k_c \right) |\det \boldsymbol{J}^e| \, \mathrm{d}\widehat{\mathrm{K}}, \\
&= \int_{\widehat{\mathrm{K}}} \sum_{a=1}^{3} \sum_{b=1}^{3} \sum_{c=1}^{3} L^{e^{-T}}_{a,b} \, J^{e^{-T}}_{a,c} \, f^l_b \, f^k_c \, |\det \boldsymbol{J}^e| \, \mathrm{d}\widehat{\mathrm{K}}.
\end{aligned}$$

Then, this integral can be computed using a numerical quadrature:

$$\begin{aligned}
\mathcal{I}^e_{k,l} &= \sum_{g=1}^{G} w_g \sum_{a=1}^{3} \sum_{b=1}^{3} \sum_{c=1}^{3} L^{e^{-T}}_{a,b} \Big|_g \, J^{e^{-T}}_{a,c} \Big|_g \, f^l_b \Big|_g \, f^k_c \Big|_g \, |\det \boldsymbol{J}^e| \Big|_g, \\
&= \sum_{g=1}^{G} \sum_{b=1}^{3} \sum_{c=1}^{3} \left(|\det \boldsymbol{J}^e| \Big|_g \sum_{a=1}^{3} L^{e^{-T}}_{a,b} \Big|_g \, J^{e^{-T}}_{a,c} \Big|_g \right) \left(w_g \, f^l_b \Big|_g \, f^k_c \Big|_g \right), \\
&= \sum_{g=1}^{G} \sum_{b=1}^{3} \sum_{c=1}^{3} \mathbf{B}\Big[e, \{g,b,c\} \Big] \mathbf{C}\Big[\{g,b,c\}, \{k,l\} \Big], \qquad (3.4a) \\
&:= \mathbf{D}\Big[e, \{k,l\} \Big], \qquad (3.4b)
\end{aligned}$$

where the operation $\{\cdot,\cdot,\cdot\}$ takes three indices and combine them into a new unique one.

From equations (3.4), we can once again notice, that each $\mathcal{I}^e_{k,l}$ elementary integral is an element of a matrix **D** at line e and column $\{k,l\}$. Moreover, this matrix **D** is constructed

by the product of two other matrices, $\mathbf{B}$ and $\mathbf{C}$, defined as follow:

$$\begin{cases} \mathbf{B}\Big[e, \{g, b, c\}\Big] = |\det \boldsymbol{J}^e| \Big|_g \sum_{a=1}^{3} L_{a,b}^{e^{-T}} \Big|_g J_{a,c}^{e^{-T}} \Big|_g , & \text{(3.5a)} \\ \mathbf{C}\Big[\{g, b, c\}, \{k, l\}\Big] = w_g \, f_b^l \Big|_g f_c^k \Big|_g . & \text{(3.5b)} \end{cases}$$

Figure 3.4 illustrates the structure of these $\mathbf{B}$ and $\mathbf{C}$ matrices. As in the previous case, matrix $\mathbf{B}$ stores only mesh (*i.e.* metric) dependent data, while matrix $\mathbf{C}$ does not. The latter is defined only on the reference space $\widehat{\mathrm{K}}$.

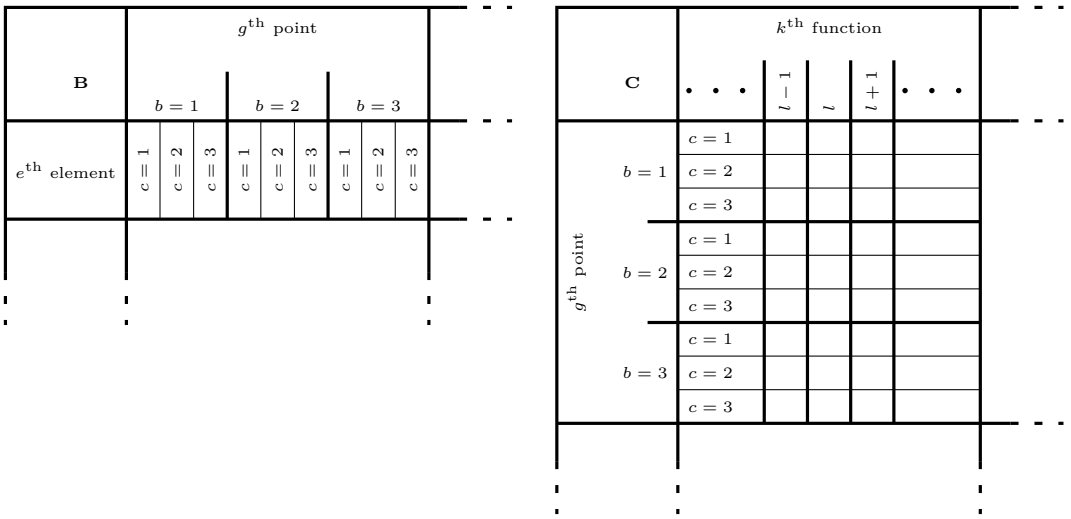


Figure 3.4: Structure of matrices $\mathbf{B}$ and $\mathbf{C}$ for an elementary integral composed by the product of $H(\mathbf{curl})$ or $H(\mathrm{div})$ functions.

3.3.3 Products of $H(\mathrm{div})$ functions

Now, let us consider the $H(\mathrm{div})$ function space case. Let $\mathcal{I}_{k,l}^e$ be the elementary integral associated to the element e and the *local* degrees of freedom k and l:

$$\begin{aligned} \mathcal{I}_{k,l}^e &= \int_{\widehat{\mathrm{K}}} \left(\boldsymbol{\mu}_r^{-1} \frac{\boldsymbol{J}^e}{\det \boldsymbol{J}^e} \, \mathbf{curl}\, \boldsymbol{f}^l \right) \cdot \left(\frac{\boldsymbol{J}^e}{\det \boldsymbol{J}^e} \, \mathbf{curl}\, \boldsymbol{f}^k \right) |\det \boldsymbol{J}^e| \, \mathrm{d}\widehat{\mathrm{K}}, \\ &= \int_{\widehat{\mathrm{K}}} \left(\boldsymbol{\mu}_r^{-1} \boldsymbol{J}^e \, \mathbf{curl}\, \boldsymbol{f}^l \right) \cdot \left(\boldsymbol{J}^e \, \mathbf{curl}\, \boldsymbol{f}^k \right) \frac{|\det \boldsymbol{J}^e|}{(\det \boldsymbol{J}^e)^2} \, \mathrm{d}\widehat{\mathrm{K}}, \\ &= \int_{\widehat{\mathrm{K}}} \frac{1}{|\det \boldsymbol{J}^e|} \Big(\underbrace{\boldsymbol{\mu}_r^{-1} \boldsymbol{J}^e}_{\mathbf{M}^e} \, \mathbf{curl}\, \boldsymbol{f}^l \Big) \cdot \Big(\boldsymbol{J}^e \, \mathbf{curl}\, \mathbf{f}^k \Big) \mathrm{d}\widehat{\mathrm{K}}. \end{aligned}$$

Once again, for convenience, the product of the inverse permeability tensor $\boldsymbol{\mu}_r^{-1}$ and the Jacobian matrix $\boldsymbol{J}^e$ has been replaced by matrix $\mathbf{M}^e$. By rewriting this integral in a per

element way, we obtain:

$$\begin{aligned}\mathcal{I}^e_{k,l} &= \int_{\widehat{\mathrm{K}}} \frac{1}{|\det \boldsymbol{J}^e|} \sum_{a=1}^{3} \left(\sum_{b=1}^{3} M^e_{a,b}\ \mathrm{curl}_b\, \boldsymbol{f}^l \right) \left(\sum_{c=1}^{3} J^e_{a,c}\ \mathrm{curl}_c\, \boldsymbol{f}^k \right) \mathrm{d}\widehat{\mathrm{K}}, \\ &= \int_{\widehat{\mathrm{K}}} \sum_{a=1}^{3}\sum_{b=1}^{3}\sum_{c=1}^{3} \frac{1}{|\det \boldsymbol{J}^e|} M^e_{a,b}\, J^e_{a,c}\ \mathrm{curl}_b\, \boldsymbol{f}^l\ \mathrm{curl}_c\, \boldsymbol{f}^k\, \mathrm{d}\widehat{\mathrm{K}}.\end{aligned}$$

As done previously, this integral is evaluated thanks to a numerical quadrature:

$$\begin{aligned}\mathcal{I}^e_{k,l} &= \sum_{g=1}^{G} w_g \sum_{a=1}^{3}\sum_{b=1}^{3}\sum_{c=1}^{3} \frac{1}{|\det \boldsymbol{J}^e|\Big|_g} M^e_{a,b}\Big|_g J^e_{a,c}\Big|_g\ \mathrm{curl}_b\, \boldsymbol{f}^l\Big|_g\ \mathrm{curl}_c\, \boldsymbol{f}^k\Big|_g, \\ &= \sum_{g=1}^{G}\sum_{b=1}^{3}\sum_{c=1}^{3} \left(\frac{1}{|\det \boldsymbol{J}^e|\Big|_g} \sum_{a=1}^{3} M^e_{a,b}\Big|_g J^e_{a,c}\Big|_g \right) \left(w_g\ \mathrm{curl}_b\, \boldsymbol{f}^l\Big|_g\ \mathrm{curl}_c\, \boldsymbol{f}^k\Big|_g \right), \\ &= \sum_{g=1}^{G}\sum_{b=1}^{3}\sum_{c=1}^{3} \mathbf{B}\Big[e, \{g,b,c\}\Big] \mathbf{C}\Big[\{g,b,c\}, \{k,l\}\Big], \qquad (3.6a) \\ &:= \mathbf{D}\Big[e, \{k,l\}\Big]. \qquad (3.6b)\end{aligned}$$

Similarly to the $H(\mathbf{curl})$ case, each $\mathcal{I}^e_{k,l}$ elementary integral is an element of a matrix $\mathbf{D}$ at line e and column $\{k,l\}$. This matrix is constructed by the product of two other matrices, $\mathbf{B}$ and $\mathbf{C}$:

$$\begin{cases} \mathbf{B}\Big[e, \{g,b,c\}\Big] = \dfrac{1}{|\det \boldsymbol{J}^e|\Big|_g} \displaystyle\sum_{a=1}^{3} M^e_{a,b}\Big|_g J^e_{a,c}\Big|_g, & (3.7a) \\ \mathbf{C}\Big[\{g,b,c\}, \{k,l\}\Big] = w_g\ \mathrm{curl}_b\, \boldsymbol{f}^l\Big|_g\ \mathrm{curl}_c\, \boldsymbol{f}^k\Big|_g. & (3.7b) \end{cases}$$

Again, matrix $\mathbf{B}$ stores only mesh (*i.e.* metric) dependent data, while matrix $\mathbf{C}$ does not. The latter is defined only on the reference space $\widehat{\mathrm{K}}$. The structure of the matrices $\mathbf{B}$ and $\mathbf{C}$ in (3.7) is similar the matrices $\mathbf{B}$ and $\mathbf{C}$ of the $H(\mathbf{curl})$ case (3.5). Thus, Figure 3.4 is also valid for the $H(\mathrm{div})$ scenario.

3.3.4 Product of a known function by an $H(\mathbf{curl})$ function

We now consider the product of a known function, $\boldsymbol{j}(\mathbf{x})$, by an $H(\mathbf{curl})$ function. Let $\mathcal{I}^e_k$ be the elementary integral associated to the element e and the *local* degree of freedom k:

$$\mathcal{I}^e_k = \int_{\widehat{\mathrm{K}}} \boldsymbol{j} \cdot \left(\boldsymbol{J}^{e^{-T}} \boldsymbol{f}^k \right) |\det \boldsymbol{J}^e|\ \mathrm{d}\widehat{\mathrm{K}}.$$

Using the same strategy as in the previous subsections, this integral is first rewritten in a per element fashion:

$$\begin{aligned}\mathcal{I}_k^e &= \int_{\widehat{\mathrm{K}}} \sum_{a=1}^{3} j_a \left(\sum_{c=1}^{3} J_{a,c}^{e^{-T}} f_c^k \right) |\det \boldsymbol{J}^e| \, \mathrm{d}\widehat{\mathrm{K}}, \\ &= \int_{\widehat{\mathrm{K}}} \sum_{a=1}^{3} \sum_{c=1}^{3} j_a J_{a,c}^{e^{-T}} f_c^k \, |\det \boldsymbol{J}^e| \, \mathrm{d}\widehat{\mathrm{K}}.\end{aligned}$$

Then, we integrate using a numerical quadrature:

$$\begin{aligned}\mathcal{I}_k^e &= \sum_{g=1}^{G} w_g \sum_{a=1}^{3} \sum_{c=1}^{3} j_a \Big|_g J_{a,c}^{e^{-T}} \Big|_g f_c^k \Big|_g |\det \boldsymbol{J}^e| \Big|_g, \\ &= \sum_{g=1}^{G} \sum_{c=1}^{3} \left(|\det \boldsymbol{J}^e| \Big|_g \sum_{a=1}^{3} j_a \Big|_g J_{a,c}^{e^{-T}} \Big|_g \right) \left(w_g \, f_c^k \Big|_g \right), \\ &= \sum_{g=1}^{G} \sum_{c=1}^{3} \mathbf{B}\Big[e, \{g,c\}\Big] \mathbf{C}\Big[\{g,c\}, k\Big], \qquad (3.8a) \\ &:= \mathbf{D}\Big[e, k\Big]. \qquad (3.8b)\end{aligned}$$

Again, from equations (3.8), we can notice that each $\mathcal{I}_k^e$ elementary integral is an element of a matrix $\mathbf{D}$ at line e and column k. By identification, this matrix is constructed by the product of the two following matrices:

$$\begin{cases} \mathbf{B}\Big[e, \{g,c\}\Big] = |\det \boldsymbol{J}^e| \Big|_g \sum_{a=1}^{3} j_a \Big|_g J_{a,c}^{e^{-T}} \Big|_g, & (3.9a) \\ \mathbf{C}\Big[\{g,c\}, k\Big] = w_g \, f_c^k \Big|_g. & (3.9b) \end{cases}$$

As in the previous case, matrix $\mathbf{B}$ stores only mesh (*i.e.* metric) dependent data, while matrix $\mathbf{C}$ does not. The latter is defined only on the reference space $\widehat{\mathrm{K}}$. The structure of the matrices $\mathbf{B}$ and $\mathbf{C}$ is available in Figure 3.5.

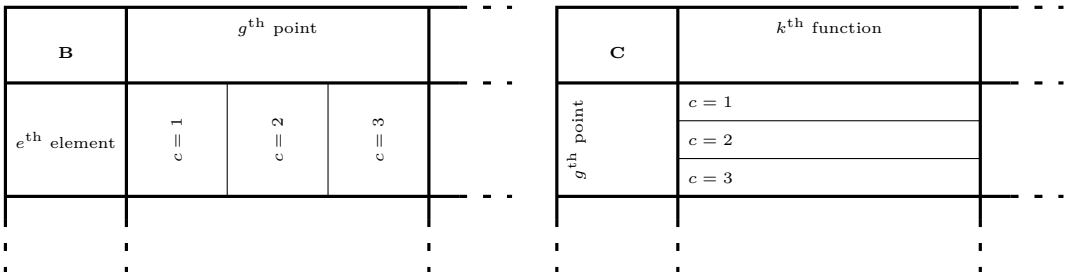


Figure 3.5: Structure of matrices $\mathbf{B}$ and $\mathbf{C}$ for an elementary integral composed by the product of a known function by an $H(\mathbf{curl})$ function.

3.3.5 Product of a known function by an H^1 function

Finally, let us consider our last scenario: the product of a known function, $j(\mathbf{x})$, by an H^1 function. Let $\mathcal{I}_k^e$ be the elementary integral associated to the element e and the *local* degree of freedom k:

$$\mathcal{I}_k^e = \int_{\widehat{\mathrm{K}}} j\, f^k\, |\det \boldsymbol{J}^e|\, \mathrm{d}\widehat{\mathrm{K}}.$$

Using a numerical quadrature to compute this integral, we obtain:

$$\begin{aligned}
\mathcal{I}_k^e &= \sum_{g=1}^{G} w_g\, j\Big|_g\, f^k\Big|_g\, |\det \boldsymbol{J}^e|\Big|_g, \\
&= \sum_{g=1}^{G} \left(j\Big|_g\, |\det \boldsymbol{J}^e|\Big|_g \right) \left(w_g\, f^k\Big|_g \right), \\
&= \sum_{g=1}^{G} \mathbf{B}\Big[e,g\Big]\, \mathbf{C}\Big[g,k\Big], \qquad (3.10a) \\
&:= \mathbf{D}\Big[e,k\Big]. \qquad (3.10b)
\end{aligned}$$

Again, from equations (3.10), we can notice that each $\mathcal{I}_k^e$ elementary integral is an element of a matrix $\mathbf{D}$ at line e and column k. By identification, this matrix is constructed by the product of two other matrices:

$$\begin{cases}
\mathbf{B}\Big[e,g\Big] = j\Big|_g\, |\det \boldsymbol{J}^e|\Big|_g, & (3.11a) \\
\mathbf{C}\Big[g,k\Big] = w_g\, f^k\Big|_g. & (3.11b)
\end{cases}$$

As in the previous case, matrix $\mathbf{B}$ stores only mesh (*i.e.* metric) dependent data, while matrix $\mathbf{C}$ does not. The latter is defined only on the reference space $\widehat{\mathrm{K}}$. The structure of the matrices $\mathbf{B}$ and $\mathbf{C}$ is available in Figure 3.6.

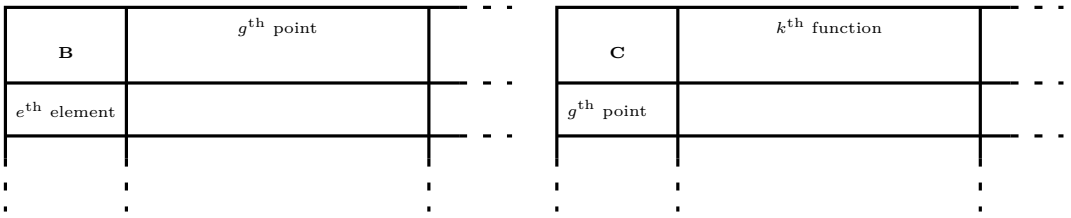


Figure 3.6: Structure of matrices $\mathbf{B}$ and $\mathbf{C}$ for an elementary integral composed by the product of a known function by an H^1 function.

3.4 Efficient assembly

With the matrix computation of the elementary integrals, we can now propose a new assembly procedure. The assembly loops of Algorithm 7 remain unchanged, while the `FeTerm` class is

revisited. In the classical approach, it computes the elementary integrals on the fly, as shown in Algorithm 6. In the new approach it fetches the precomputed integrals, as proposed in Algorithm 8. For simplicity and clarity, the example below is restricted to the Maxwell's elementary integrals (2.22).

```
FeTerm::FeTerm(Vector<Element> e)
  Description
    Construct a new FeTerm for the Maxwell's elementary integrals (2.22),
    defined on the given Vector of Element.
  Implementation (C++)
    /* Precomputation of the integrals (2.22b),                          */
    /* using the product of H(curl) function procedure.                  */
    DCurl = precomputeHCurl(e);       // DCurl is a member matrix of FeTerm
    /* Precomputation of the integrals (2.22a),                          */
    /* using the product of H(div) function procedure.                   */
    DDiv = precomputeHDiv(e);          // DDiv is a member matrix of FeTerm
---------------------------------------------------------------------------
scalar FeTerm::get(Element e, int k, int l)
  Description
    This method computes the integral (2.21) for a given triplet {e,k,l}.
    This is a faster reimplementation of algorithm 6.
  Implementation (C++)
    int eIdx = e.getIndex();
    return  DDiv[eIdx][combineIndex(k, l)] -
            k^2 * DCurl[eIdx][combineIndex(k, l)];
```

Algorithm 8: An efficient quadrature for the finite element assembly (electromagnetic example).

Two final remarks are worth mentioning. First, the construction of the metric-dependent matrices requires a loop on every mesh element. Secondly, the object **FeTerm** has to be instantiated, *before* the assembly begins.

3.5 Orientation problem

Comparing Algorithms 6 and 8, it is clear that something has been forgotten: the orientation process. Previously, in Algorithm 6, the orientation of the physical elements was taken into account on-the-fly: that is, a new basis was generated for each element before computing the requested integral. Unfortunately, this on-the-fly approach cannot be used any more in the newly proposed quadrature algorithm. Indeed, in this version, we want to reuse the product of every possible basis function in the product $\mathbf{BC}$. Therefore, we cannot recompute a new basis for every physical element.

In order to handle the orientation of the physical elements, while keeping the matrix product approach, the following procedure can be used:

1. for a given element type[4], generate every possible orientation;
2. give each possible orientation a tag;
3. sort the physical element vector with respect to their orientation tag;
4. propagate the new indices in `Element::getIndex`;
5. for each sub-vector,
   (a) compute the associated oriented basis and generate matrix $\mathbf{C}$;
   (b) compute matrix $\mathbf{B}$ with elements of the sub-vector;
   (c) compute $\mathbf{D}_i = \mathbf{BC}$;
6. append every computed matrix $\mathbf{D}_i$ to matrix $\mathbf{D}$.

This last matrix $\mathbf{D}$ contains every possible elementary integral, with a correctly oriented basis functions. With this approach, Algorithm 8 remains valid.

Chapter 4 will discuss on the different solutions to generate every possible orientation for a given element type. At this stage, we just assume that the physical element vector has been correctly sorted with respect to the elements orientations.

3.6 Benchmarks

In this last section, the performances of the quadrature procedure will be analyzed. Two scenarios are considered: vectorial electromagnetic simulations and scalar acoustic simulations. Each scenario is then divided into two sub-cases: a complex arithmetic waveguide case, and a real arithmetic cavity case.

3.6.1 Electromagnetic simulations

Waveguide

Let us go back to the waveguide benchmark of section 3.2. This time, two implementations of the `FeTerm` class are considered: the on-the-fly version of Algorithm 6 (the one used in section 3.2), and the matrix version of Algorithm 8. The assembly times, as well as the speedup of the fast quadrature with respect to the old on-the-fly quadrature, are reported in Figure 3.7. It can be directly seen that the newly proposed assembly procedure outperforms the classical one, and that the speedup increases with the FE order. For instance, for a 5^{th} order basis functions (508410 unknowns), the speedup reaches 4.6.

Another interesting comparison is the ratio between the time spent solving the FE linear system, and the time spent assembling the FE matrix: results are available in Figure 3.8. It can be directly observed that using the fast quadrature approach, the solver time becomes a higher fraction of the overall computation time.

[4]We mean by element type, the shape of the element: that is, whether the element is a line, a triangle, an hexahedron, ...

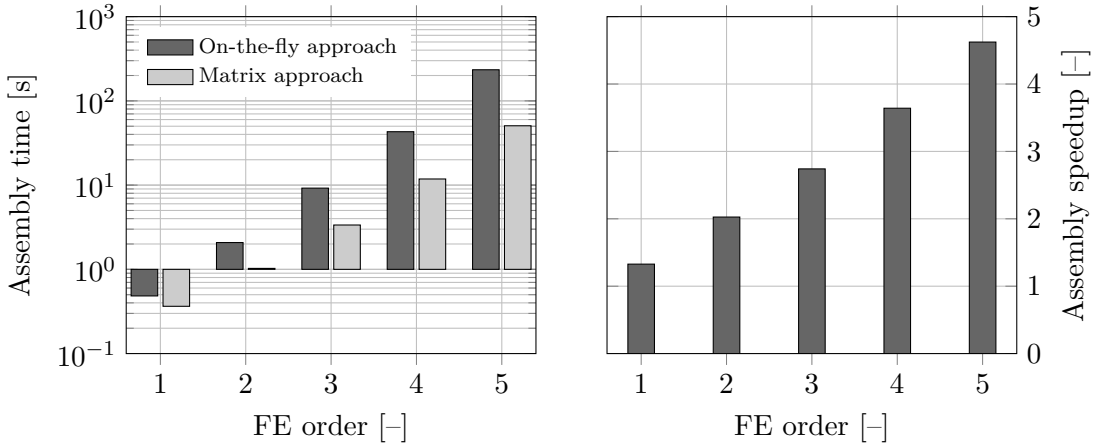


Figure 3.7: High-order FE simulations of a waveguide: assembly time for the on-the-fly and matrix quadratures, electromagnetic case.

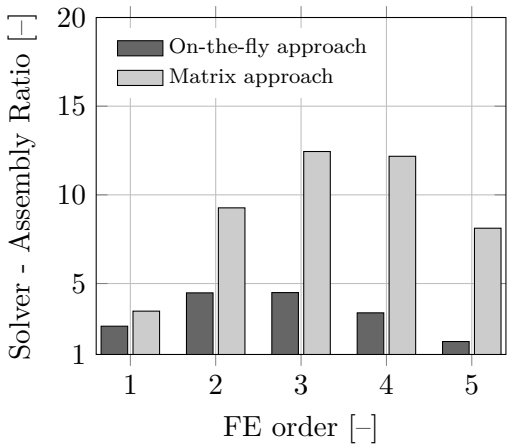


Figure 3.8: Ratio between the time spent solving the FE linear system and assembling the FE matrix for a waveguide simulations, electromagnetic case.

Cavity

In the previous benchmark, a infinite waveguide was considered, thus leading to a complex problem. However, complex arithmetic is significantly slower than real arithmetic, even on modern floating-point units[5]. Thus, the latter benchmark is actually the worst case scenario.

Let us now consider a real arithmetic case. To do so, the waveguide is truncated using a perfect reflector (and not a Silver-Müller condition), transforming the metallic waveguide into a perfect cavity. The assembly time for the on-the-fly and the matrix quadrature are displayed in Figure 3.9.

As expected, the matrix assembly procedure outperforms the classical one, with a speedup of 22.6 for the 6^{th} order case (825993 unknowns). Moreover, the speedups are significantly

[5]Hardware, usually integrated into the central processing unit, responsible of floating-point operations.

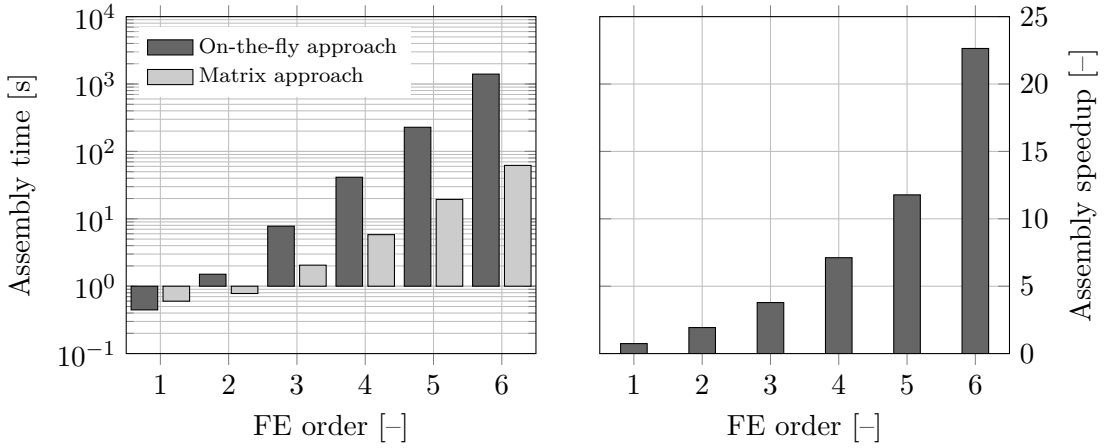


Figure 3.9: High-order FE simulations of a cavity: assembly time for the on-the-fly and matrix quadratures, electromagnetic case.

higher than in the complex arithmetic situation.

For the ratio between the solver and the assembly times, an interesting behavior can be observed in Figure 3.10. For the 5^{th} and 6^{th} FE order simulation, this ratio drops below 1, when the classical on-the-fly quadrature approach is used. This means that, for these simulations, the time spent assembling the FE matrix was higher than the time spent solving it! With the matrix quadrature version, this ratio remains above 1.

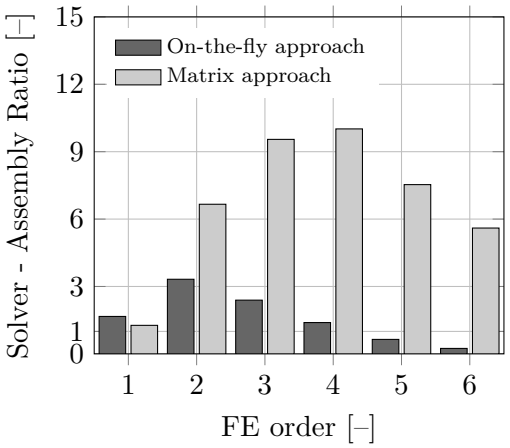


Figure 3.10: Ratio between the time spent solving the FE linear system and assembling the FE matrix for a cavity simulations, electromagnetic case.

3.6.2 Acoustic simulations

Finally, let us test the acoustic case. As done previously, we first start by the infinite waveguide and then the cavity case.

Waveguide

Let us compare the assembly time, and the ratio between the time spent solving and assembling the linear system. Results are reported in Figures 3.11 and 3.12. We can directly notice the same behavior as the electromagnetic case. First, the newly proposed assembly approach outperforms the classical one, as the finite element order is increased. Second, with the fast method, a high fraction of the overall computation time is spent solving the linear system, rather than assembling it. This is not the case with the classical approach, as we increase the discretization order.

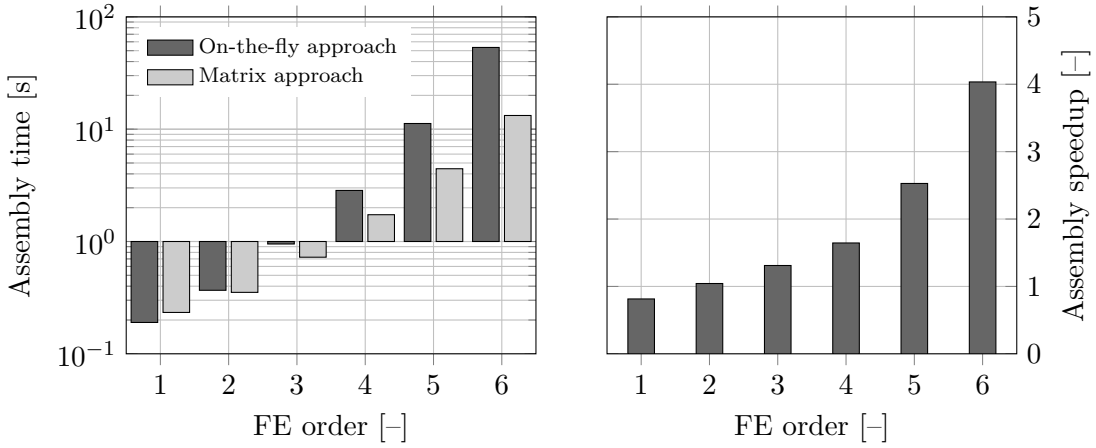


Figure 3.11: High-order FE simulations of a waveguide: assembly time for the on-the-fly and matrix quadratures, acoustic case.

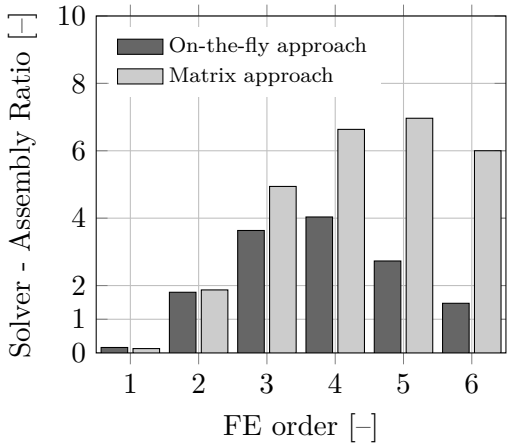


Figure 3.12: Ratio between the time spent solving the FE linear system and assembling the FE matrix for a waveguide simulations, acoustic case.

Cavity

Let us now consider the acoustic cavity case. Timing results for the assembly are presented in Figure 3.13, and the ratios between the solver and assembler timings are reported in Figure 3.14. Again, the same behavior, as in the electromagnetic case, is observed.

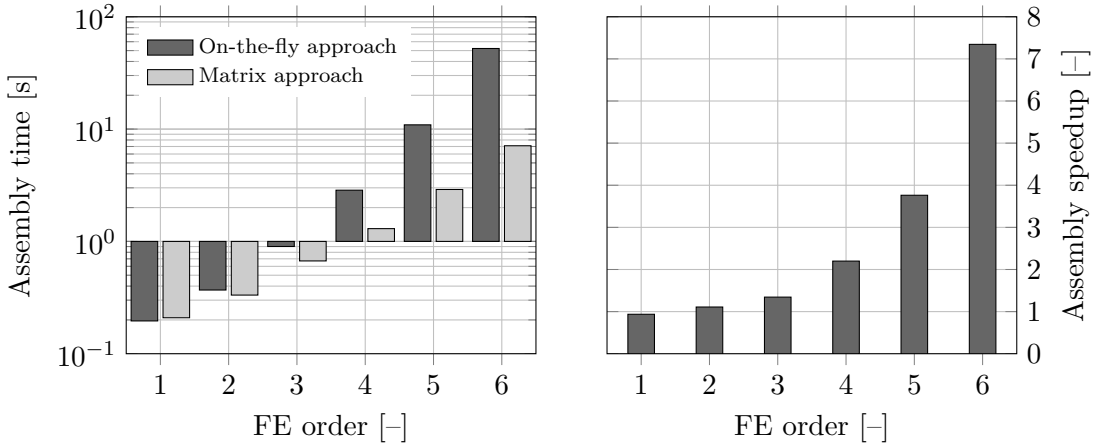


Figure 3.13: High-order FE simulations of a cavity: assembly time for the on-the-fly and matrix quadratures, acoustic case.

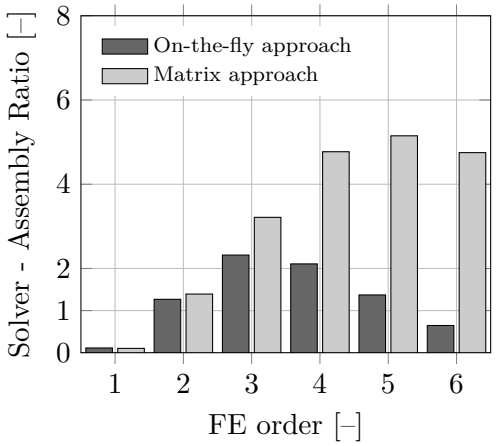


Figure 3.14: Ratio between the time spent solving the FE linear system and assembling the FE matrix for a cavity simulations, acoustic case.

Let us note, that the speedups are significantly lower than in the electromagnetic case. This phenomenon is explained by the fact, that less unknowns were assembled in the acoustic case, since the degrees of freedom are associated to nodes and not to edges. For instance, at order 6, we have 196601 unknwons, compared to the 825993 of the electromagnetic counterpart.

3.7 Conclusion

In this chapter, we presented a strategy to improve the finite element assembly time. To achieve this purpose, we rewrote our finite element terms, so that they can be computed in a matrix-matrix product. Because of the excellent cache reuse of this operation, the assembly time can be decreased, compared to the classical on-the-fly approach of Algorithm 6.

We tested our implementation on various setups. While the assembly time is not improved for lower-order discretizations, we found that it can be significantly reduced, with the proposed matrix-matrix approach, on higher-order simulations. This speedup is achieved at a cost: the addition storage, during the assembly phase, of temporary data. For the high-frequency time-harmonic problems treated in this thesis, this additional cost poses no issue:

1. the temporary data can be freed once the matrix is assembled;
2. the memory required by the factorization of the matrix is significantly larger.

In this chapter, an important aspect of the method was not treated: the orientation of the basis functions. Indeed, with this new approach, the classical on-the-fly orientation procedure cannot be used anymore. To solve this problem, we proposed to sort the physical elements with respect to some orientation criterion. Then, for each type of orientation, a matrix-matrix product is generated. The practical aspects of this orientation technique are presented in chapter 4.

Chapter 4

Orientation dictionary

In the previous chapter, an efficient quadrature procedure was introduced, enabling a fast assembly of the finite element linear system. However, it was required to generate every possible orientation for a given element type. This chapter addresses this topic, using orientation dictionaries.

4.1 The full permutation tree approach

4.1.1 A simple case

Let us start by assuming an element defined by the V following vertices: $\{V^0, V^1, \ldots, V^{V-1}\}$. In other words, we assume an element composed by vertices with global indices in the range $[0, \ldots, V-1]$.

By *permuting* these vertices, we can generate $V!$ elements. These elements will only differ by the *order* in which their vertices are taken. Moreover, these $V!$ permutations will lead to different edges and faces orientations, that is $V!$ different element orientations.

Thus, in order to generate every possible basis for every possible orientation, we can generate the $V!$ permutations of the set $\mathcal{V} = \{0, 1, \ldots, V-1\}$, defining the global vertex indices of an element. Then we can apply Algorithm 5 to generate the basis functions associated to each one of the $V!$ possible elements. A tree structure can be used to represent those permutations. This permutation tree has the following properties:

1. the tree has a depth of V;
2. a node at depth d has $V - d$ children[1];
3. each node has a label between 0 and $V-1$, with the exception of the root, that stays unlabeled;
4. sibling nodes have strictly different labels;
5. a node cannot be given a label already taken by one of its ancestors;
6. the descendants of a node are sorted with increasing labels;
7. a leaf can be tagged.

[1]The depth of the root node is equal to 0.

With this structure, the ordered sequence of labels on a path from the root to a leaf is a possible permutation of the set $\mathcal{V}$. Moreover, this structure has exactly $V!$ possible paths that leads to different label sequences. Thus, this tree spans all the possible permutations of the set $\mathcal{V}$.

This tree structure can be also used as a permutation *dictionary*: a given permuted sequence can be matched to a path in the tree, and thus to a leaf tag. Indeed, a possible permutation $\mathbf{v}$ of the set $\mathcal{V}$ corresponds to the path, traveling (in order) through the nodes labeled $\mathbf{v}(0)$, $\mathbf{v}(1), \ldots, \mathbf{v}(V-1)$. Moreover, using property 7, the ending node of the path can be associated to a tag. Thus, this structure is a dictionary, where a permuted sequence is a key that maps to a user-defined value. The matching time scales in $V \log_2 V$. Indeed, we need to go through the V nodes of a path. And at each level, the next node can be found by a dichotomic search that scales in $\log_2 V$. This dichotomic approach is allowed thanks to property number 6.

This permutation dictionary can be advantageously transformed into an *orientation dictionary*. Since the tree is spanning every possible permutation of $\mathcal{V}$, it must also span the possible orientations of an element with global vertex indices taken in $\mathcal{V}$. Indeed, the permutation tree is generating every possible local vertex indices ordering, for a given set of global vertex indices[2]. Furthermore, the orientation rules (2.24), (2.25) and (2.26) are based on both the local and global vertex indices of an element. Thus, the permutation tree is both:

1. a *generator* of every possible orientation of an element;
2. an orientation *dictionary*.

As first guess, let us assume that each leaf of the tree is a possible orientation of an element. By setting each leaf tag with a unique number, every possible orientation can then be uniquely identified.

For instance, by using the notation introduced in section 2.4, we can generate the following six triangles[3], by permuting the set $\mathcal{V} = \{0, 1, 2\}$:

$$\begin{aligned} &\mathbf{K}^e = [0, 1, 2], \quad \mathbf{K}^e = [1, 2, 0], \quad \mathbf{K}^e = [2, 0, 1], \\ &\mathbf{K}^e = [0, 2, 1], \quad \mathbf{K}^e = [1, 0, 2], \quad \mathbf{K}^e = [2, 1, 0]. \end{aligned}$$

Each one of these triangles has its own edges and face orientation, as shown in Figure 4.1. Figure 4.2 depicts the permutation tree associated to $\mathcal{V} = \{0, 1, 2\}$. On this tree, each leaf has a unique tag, meaning that each permutation of $\mathcal{V}$ is considered as a possible orientation of the triangle composed by the vertices $\{V^0, V^1, V^2\}$. If the element $\mathbf{K}^e = [1, 0, 2]$ is encountered in the mesh, its ordered global vertex indices can be introduces in the tree, and a unique orientation tag is associated with the element: in this example, the tag 2.

[2] With, up to now, the simplification that the global vertex indices are ranging from 0 to $V - 1$.

[3] Indeed, a triangle has three vertices: so $V = 3$ and $V! = 6$.

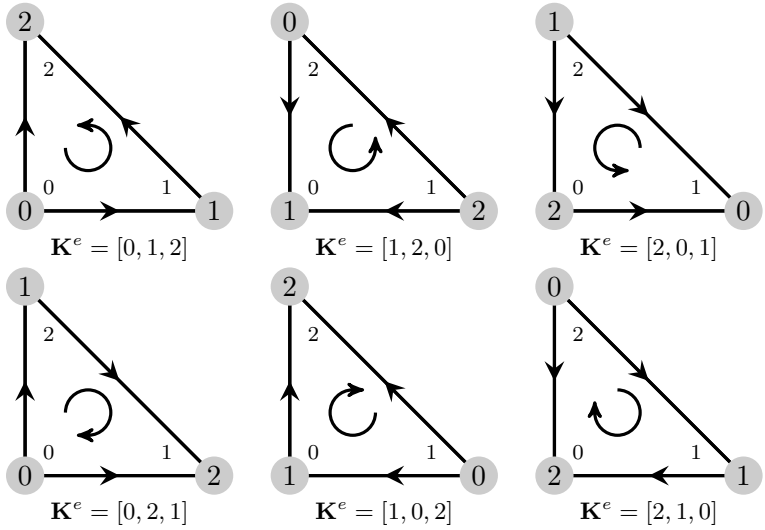


Figure 4.1: Six possible triangles with their edges and face orientations.

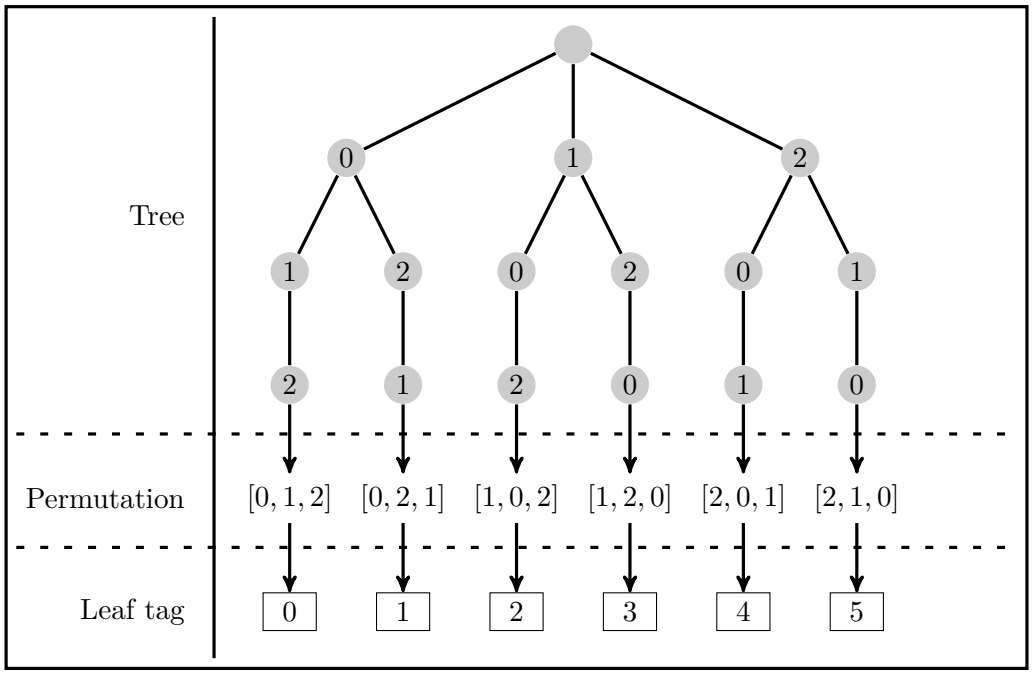


Figure 4.2: Permutation tree of the set $\mathcal{V} = \{0, 1, 2\}$.

4.1.2 The general case: global index reduction

Let us now consider the general situation of an element defined by the following V global vertex indices $\{a, b, \ldots, q\}$. It is worth mentioning that there is no assumptions on this vertex numbering. Is it possible to reuse the permutation tree structure developed previously, in order to implement an orientation generator/dictionary? Fortunately the answer is yes. Indeed, even if the permutation tree relies on the assumption that an element is composed by the vertex indices $\{0, 1, \ldots, V-1\}$, the orientation rules rely only on *inequalities*, which are *similar* to the inequalities present in the permutation tree.

Practically, for a given element $\mathbf{K}^e = [a, b, \ldots, q]$, we need to find an element, defined by the set $\mathcal{V} = \{0, 1, \ldots, V-1\}$, leading the the same inequalities (2.24), (2.25) and (2.26). To do so, we need to construct a vector $\mathbf{K}^\star$, with elements taken from the set $\mathcal{V}$, such that the smaller/bigger relations between the elements of $\mathbf{K}^\star$ are the same than the relations in $\mathbf{K}^e$. We call this operation a *global index reduction*. Algorithm 9 proposes a possible implementation of this procedure.

```
Vector<int> reduceGlobalIndex(Element element)
  Description
    This function returns the reduced global indices of the given
    Element.
  Implementation (C++)

    // Take the global vertex indices of the given element
    Vector<int> vIndex = element.getVertexIndex();

    // Sort vIndex in ascending order, and keep sorting permutations
    Vector<int> vIndexPermutation = sort(vIndex);

    /* The first entry of vIndexPermutation is the position in vIndex       */
    /* of the smallest vertex index.                                        */
    /* The second entry the position of the second smallest.                */
    /* And so on...                                                         */

    /* We want the smallest vertex index to have the reduced vertex 0.      */
    /* The second smallest to have the reduced vertex 1.                    */
    /* And so on...                                                         */

    // Fill a vector with the reduced index
    Vector<int> reducedIndex(vIndex.size());
    for(int i = 0; i < vIndex.size(); i++)
      reducedIndex[vIndexPermutation[i]] = i;

    // Done
    return(reducedIndex);
```

Algorithm 9: Implementation of the global index reduction.

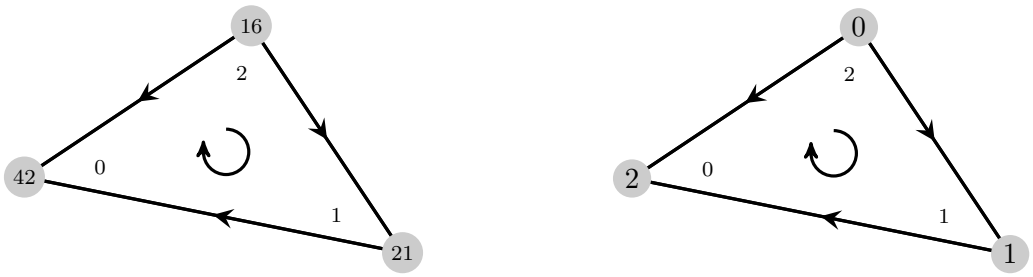
Let us illustrate this by the following example. $\mathbf{K}^e = [42, 21, 16]$ is the triangle illustrated

at Figure 4.3a. A triangle has its edges and face defined in the following way:

$$\begin{cases} \widetilde{\mathbf{e}}^0 = \{0, 1\}, \\ \widetilde{\mathbf{e}}^1 = \{1, 2\}, \\ \widetilde{\mathbf{e}}^2 = \{2, 0\}, \\ \widetilde{\mathbf{f}}^0 = \{0, 1, 2\}. \end{cases}$$

By applying the inequalities (2.24) and (2.25), we end up with the following oriented edges and face:

$$\begin{cases} \mathbf{e}^0 = [1, 0] & \text{since} & \mathbf{K}^e(1) < \mathbf{K}^e(0) & \iff & 21 < 42, & (4.1a) \\ \mathbf{e}^1 = [2, 1] & \text{since} & \mathbf{K}^e(2) < \mathbf{K}^e(1) & \iff & 16 < 21, & (4.1b) \\ \mathbf{e}^2 = [2, 0] & \text{since} & \mathbf{K}^e(2) < \mathbf{K}^e(0) & \iff & 16 < 42, & (4.1c) \\ \mathbf{f}^0 = [2, 1, 0] & \text{since} & \mathbf{K}^e(2) < \mathbf{K}^e(1) < \mathbf{K}^e(0) & \iff & 16 < 21 < 42. & (4.1d) \end{cases}$$



(a) The triangle $\mathbf{K}^e = [42, 21, 16]$.

(b) The triangle $\mathbf{K}^e = [2, 1, 0]$.

Figure 4.3: Two triangles with the same orientation: one defined by the vertex global indices $\{16, 21, 42\}$ and the other by $\{0, 1, 2\}$.

Using Algorithm 9, element $\mathbf{K}^e = [42, 21, 16]$ is transformed into the equivalent (from the orientation point of view) element $\mathbf{K}^\star = [2, 1, 0]$ of Figure 4.3b. Indeed, by applying the orientation rules on $\mathbf{K}^\star$, we have:

$$\begin{cases} \mathbf{e}^0 = [1, 0] & \text{since} & \mathbf{K}^\star(1) < \mathbf{K}^\star(0) & \iff & 1 < 2, & (4.2a) \\ \mathbf{e}^1 = [2, 1] & \text{since} & \mathbf{K}^\star(2) < \mathbf{K}^\star(1) & \iff & 0 < 1, & (4.2b) \\ \mathbf{e}^2 = [2, 0] & \text{since} & \mathbf{K}^\star(2) < \mathbf{K}^\star(0) & \iff & 0 < 2, & (4.2c) \\ \mathbf{f}^0 = [2, 1, 0] & \text{since} & \mathbf{K}^\star(2) < \mathbf{K}^\star(1) < \mathbf{K}^\star(0) & \iff & 0 < 1 < 2. & (4.2d) \end{cases}$$

We can directly see from (4.1) and (4.2), that the two triangles are equivalent from the orientation point of view.

4.1.3 Integration with the fast quadrature procedure

With the orientation tree in hand, we can now feed the fast quadrature procedure with correctly oriented basis. Indeed, with the tree, we can generate every possible orientation for

a given element type. With these orientations, we can then generate every possible oriented basis. Moreover, using the tree as an orientation dictionary, we can *sort* all the mesh elements with respect to their orientation. Thus, we can apply the fast quadrature procedure of chapter 3 to each group of element sharing the same orientation.

4.1.4 Analysis of the permutation tree approach

The method presented in this section allows the generation of every possible basis for every possible orientation, thus enabling the use of the fast quadrature of chapter 3. However, it exhibits a major weakness: for a given element type, the number of possible orientations, *and thus the number of possible bases*, is the *factorial* of the number of vertices.

In the cases of lines, triangles, quadrangle, tetrahedra, pyramids and prisms, this behavior can be tolerated. But in the case of hexahedra, we cannot accept a such behavior. Table 4.1 summarizes the number of bases generated by the naive approach.

Element type	Number of vertices	Number of bases
Line	2	2
Triangle	3	6
Quadrangle	4	24
Tetrahedron	4	24
Pyramid	5	120
Prism	6	720
Hexahedron	8	40 320

Table 4.1: Number of possible bases generated by the naive approach.

In the next section a solution to decrease the factorial impact of this method is proposed.

4.2 The connectivity approach

Let us look at the 4! possible vertex permutations of a quadrangle, as displayed in Figure 4.4. It can be directly seen that these quadrangles can be split in 3 groups, by considering the vertices *connectivity*. In other words, the quadrangles of a group are sharing the same edges.

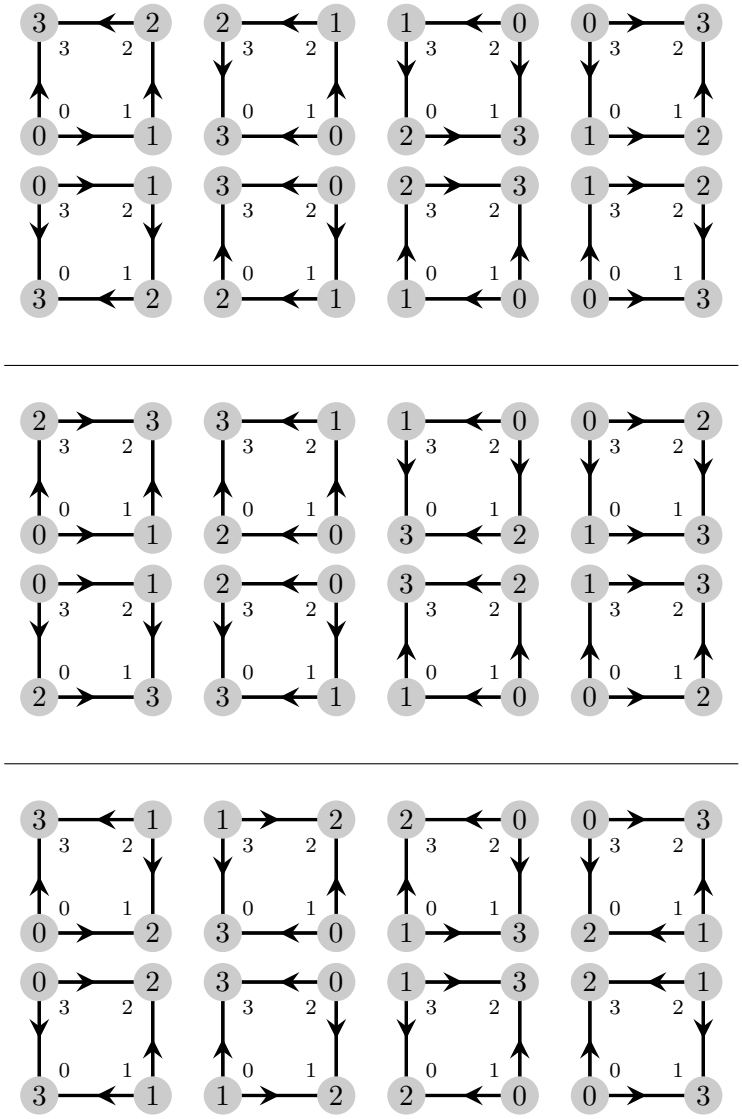


Figure 4.4: The 24 possible vertices permutations of a quadrangle.

Thus, for a given group, it makes sense to compute a *geometrical mapping* that maps one quadrangle on another *of the same group*, as shown in Figure 4.5. On the other hand, it is impossible to find a mapping between a quadrangle of one group and another quadrangle of another group. Indeed, in this case, the resulting mapped quadrangle will not keep its convexity, as shown in Figure 4.6.

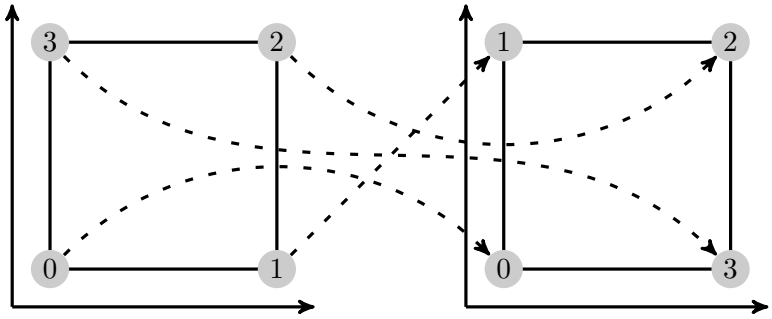


Figure 4.5: Mapping between two quadrangles of the same group: convexity kept.

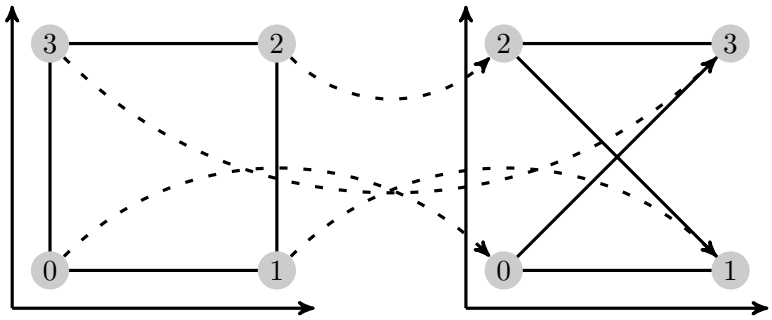


Figure 4.6: Mapping between two quadrangles of the different groups: convexity lost.

Using geometrical mappings, we no longer need to define a basis for each possible permuted quadrangle. Instead, *we can define a basis on only one quadrangle of each group*. The other quadrangles of a group can then be taken into account by mapping them on the reference quadrangle. Thus, for the quadrangular case, we only need three finite element bases. They can be defined, for instance, on the reference quadrangles of Figure 4.7.

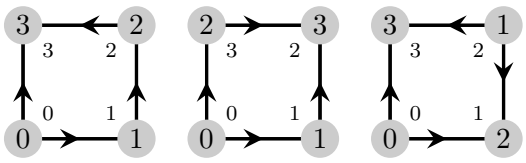


Figure 4.7: The 3 quadrangles on which a set of bases can be defined.

4.2.1 The general case: connectivity groups and reference orientations

As showed above, for the quadrangular case, only three quadrangles are actually needed to define the finite element bases. We managed to reduce the number of bases, by *grouping* the quadrangles by vertex *connectivity*. Indeed, in a same group, the quadrangles can be mapped on each other. Thus, only one basis is needed for a given group. For a given mesh element, this approach leads us to consider two different kinds of orientation:

1. the orientation of the mesh element, called *natural* orientation;
2. the orientation of an element, on which the mesh element can be mapped, called

reference orientation; or reference element, since this oriented element will be chosen to define an FE basis.

The grouping idea of the quadrangle can be generalized for any kind of element type. Doing so, we end up with the following procedure:

1. generate the permutation tree associated to an element type, as in the previous section;
2. tag every leaf with a pair; the first entry of this pair is a unique number defining the leaf; the second entry will be set latter;
3. sort all these permutations into groups with the same vertex connectivity; those groups are called *connectivity groups*;
4. select one permutation (*i.e.* one leaf of the permutation tree) in every connectivity group as a reference;
   (a) choose the permutation with the smallest leaf number;
   (b) this number becomes the tag of this connectivity group;
5. the second entry of each leaf tag is the tag of its connectivity group;
6. define an orientated reference element, for each reference permutation of each group.

It is worth noticing that step 3 is time consuming, since it requires to compare the connectivity of all the vertices permutations. However, it only needs to be applied (offline) once per element type, and the resulting tree can be reused. Generating a permutation tree for the hexahedral case, which is the more complex, requires around 35 seconds. Figure 4.8 illustrates a permutation tree for a quadrangular element type, that has been tagged with its connectivity groups.

Let us remark that the newly tagged permutation tree becomes both a *natural* and *reference* orientation *dictionary*. Indeed, because of the first entry of the leaf tag, this permutation tree is the same as the previous tree: thus leading to a natural orientation dictionary. Moreover, because of the second entry of the leaf tag, each natural orientation is associated a reference orientation: so, this tree is also a reference orientation dictionary.

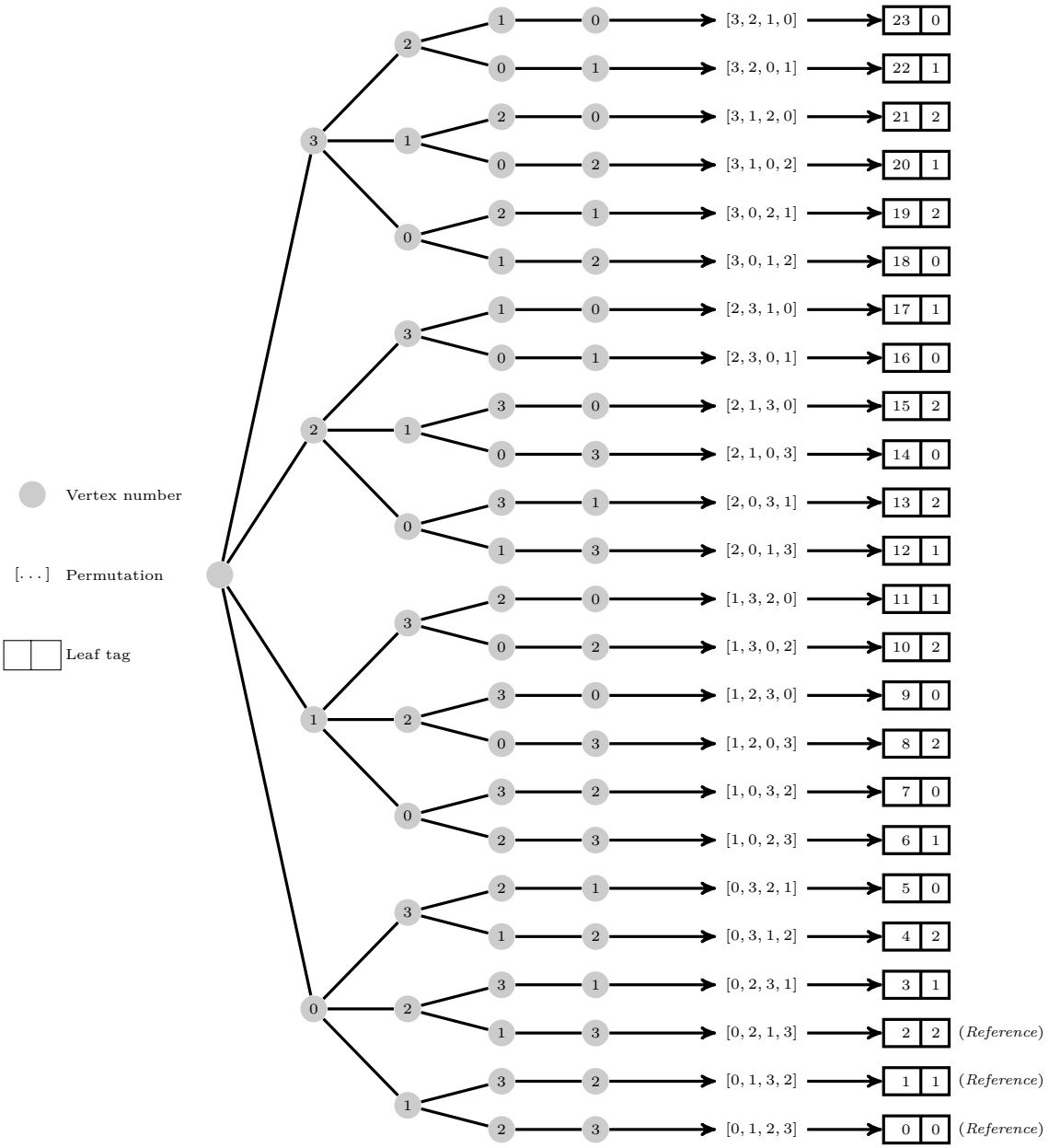


Figure 4.8: Permutation tree, using connectivity groups, for the quadrangular case.

4.2.2 Mappings

If we take connectivity into account, in order to reduce the number of possible finite element bases, we end up with three different spaces:

- the physical space, that is the space of the mesh elements;
- a reference space, usually given by the mesh module, used to define the mesh Lagrange functions (see section 2.3.1); this space keeps the natural orientation of the mesh element;
- a set of oriented reference spaces, one per connectivity group, used to define the finite element bases; these spaces are oriented thanks to the reference orientations.

Figure 4.9 illustrates this with a mesh composed of a single curved quadrangle. The first mapping, with Jacobian matrix $\boldsymbol{J}^0$, is usually handled by the mesh module. Thus, we need a way to compute the second mapping, with Jacobian matrix $\boldsymbol{J}^1$.

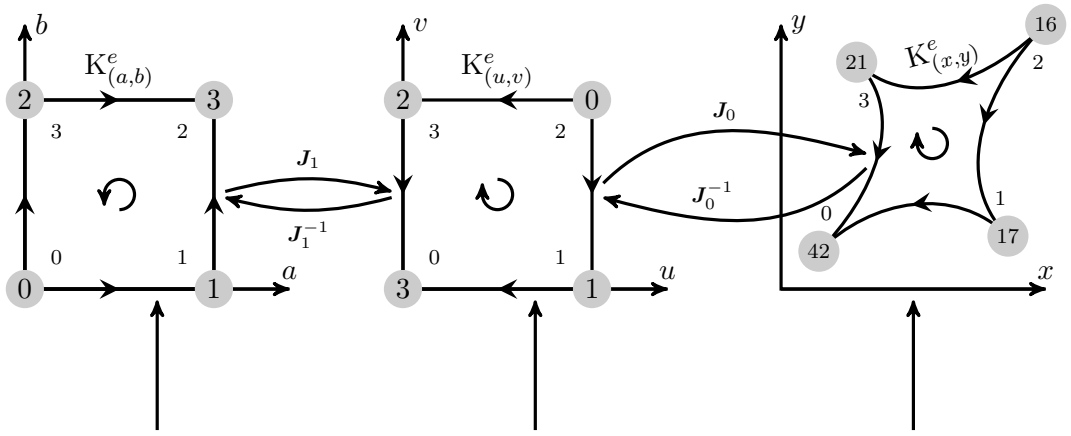


Figure 4.9: The physical space, the mesh reference space and an oriented FE reference spaces.

The spaces (u, v) and (a, b) are, by construction, defining the same geometry. They differ only on their vertex positions. Thus, the shape functions and the vertex positions from one space can be used in the other space, providing a *reindexing*. Therefore, we have everything we need to go from one space to another. Before going any further, it is worth noticing that the mappings between the spaces (a, b) and (u, v) involve straight sided mesh element. Therefore, only the first order shape functions are needed.

We call $\mathbf{p}^i_{(\gamma,\eta)}$ the position, in the space (γ, η), of the vertex with *local index* i. In order to

take the reindexing into account, we define two vectors $\mathbf{d}$ and $\mathbf{r}$ as follow:

$$\begin{cases} \mathbf{d}: & \mathbf{p}^i_{(u,v)} = \mathbf{p}^{\mathbf{d}(i)}_{(a,b)} \quad \forall i, & (4.3a) \\ \mathbf{r}: & \mathbf{p}^i_{(a,b)} = \mathbf{p}^{\mathbf{r}(i)}_{(u,v)} \quad \forall i. & (4.3b) \end{cases}$$

That is, vector $\mathbf{d}$ enables us to map a vertex from the (a, b) space to the (u, v) space; and vector $\mathbf{r}$ maps a vertex from (u, v) to (a, b).

Obviously, these vectors are related to permutations of the permutation tree. Let us assume the following situation: a given mesh element is associated with the natural permutation $\mathbf{v}_n$ in the tree[4]. Furthermore, it is also associated with the reference permutation $\mathbf{v}_r$[5]. Then, we have:

$$\begin{cases} \mathbf{v}_n(i) = \mathbf{v}_r(\mathbf{d}(i)) \quad \forall i; & (4.4a) \\ \mathbf{v}_n(\mathbf{r}(i)) = \mathbf{v}_r(i) \quad \forall i. & (4.4b) \end{cases}$$

Let us illustrate the above relations using Figure 4.9. In the situation depicted, the mesh element is maped onto a reference space (u, v), which corresponds, in the permutation tree, to the permutation:

$$\mathbf{v}_n = [3, 1, 0, 2].$$

This natural space is then mapped onto a reference space (a, b), oriented with the reference orientation of the mesh element. This reference space corresponds, in the permutation tree, to the permutation:

$$\mathbf{v}_r = [0, 1, 3, 2].$$

Applying (4.4), we have[6]:

$$\begin{cases} \mathbf{d} = [2, 1, 0, 3]; \\ \mathbf{r} = [2, 1, 0, 3]. \end{cases}$$

Let us now consider the mapping of vertex V^3, or equivalently vertex V^{42}. We have:

$$\begin{cases} \mathbf{K}^e_{(a,b)}(0) = 42; \\ \mathbf{K}^e_{(u,v)}(0) = 3; \\ \mathbf{K}^e_{(a,b)}(2) = 3. \end{cases}$$

Using the vectors $\mathbf{d}$ and $\mathbf{r}$ as defined above, we can indeed map V^3 from (a, b) to (u, v), or from (u, v) from (a, b), since:

$$\begin{cases} \mathbf{v}_r(2) = \mathbf{v}_n(\mathbf{r}(2)) = \mathbf{v}_n(0); \\ \mathbf{v}_n(0) = \mathbf{v}_r(\mathbf{d}(0)) = \mathbf{v}_r(2). \end{cases}$$

That is, the vertex with local index 0 in the (u, v) space is mapped into the vertex with local index 2 in the (a, b) space. This vertex in both spaces is V^3.

[4]In other words, the *natural* orientation of the mesh element is the one associated to permutation $\mathbf{v}_n$.

[5]In other words, the *reference* orientation of the mesh element is the one associated to permutation $\mathbf{v}_r$.

[6]In this particular case, we have $\mathbf{d} = \mathbf{r}$. However, this is not always the case: for instance $\mathbf{d}$ and $\mathbf{r}$ are different with $\mathbf{v}_n = [1, 3, 2, 0]$ and $\mathbf{v}_r = [0, 1, 3, 2]$.

Now let us define the mapping from any point in (a,b) (resp. (u,v)) onto (u,v) (resp. (a,b)). We call $h^i(\gamma,\eta)$ the geometrical shape function associated to the i^{th} vertex in the space (γ,η), and V is the number of vertices. Then, we have:

$$\begin{cases} \Big[\mathbf{x}(a,b)\Big](u,v) = \displaystyle\sum_{i=0}^{V-1} \mathbf{p}^i_{(u,v)}\, h^i(a,b) = \sum_{i=0}^{V-1} \mathbf{p}^{\mathbf{d}(i)}_{(a,b)}\, h^i(a,b); \\ \Big[\mathbf{x}(u,v)\Big](a,b) = \displaystyle\sum_{i=0}^{V-1} \mathbf{p}^i_{(a,b)}\, h^i(u,v) = \sum_{i=0}^{V-1} \mathbf{p}^{\mathbf{r}(i)}_{(u,v)}\, h^i(u,v). \end{cases}$$

For the Jacobian matrix associated to the mapping from (a,b) onto (u,v), we have:

$$\begin{aligned} \boldsymbol{J}_1(a,b) &= \begin{bmatrix} \displaystyle\sum_{i=0}^{V-1} \mathbf{p}^i_{(u,v)}(0)\, \frac{\partial h^i(a,b)}{\partial a} & \displaystyle\sum_{i=0}^{V-1} \mathbf{p}^i_{(u,v)}(0)\, \frac{\partial h^i(a,b)}{\partial b} \\ \displaystyle\sum_{i=0}^{V-1} \mathbf{p}^i_{(u,v)}(1)\, \frac{\partial h^i(a,b)}{\partial a} & \displaystyle\sum_{i=0}^{V-1} \mathbf{p}^i_{(u,v)}(1)\, \frac{\partial h^i(a,b)}{\partial b} \end{bmatrix}, \\ &= \begin{bmatrix} \displaystyle\sum_{i=0}^{V-1} \mathbf{p}^{\mathbf{d}(i)}_{(a,b)}(0)\, \frac{\partial h^i(a,b)}{\partial a} & \displaystyle\sum_{i=0}^{V-1} \mathbf{p}^{\mathbf{d}(i)}_{(a,b)}(0)\, \frac{\partial h^i(a,b)}{\partial b} \\ \displaystyle\sum_{i=0}^{V-1} \mathbf{p}^{\mathbf{d}(i)}_{(a,b)}(1)\, \frac{\partial h^i(a,b)}{\partial a} & \displaystyle\sum_{i=0}^{V-1} \mathbf{p}^{\mathbf{d}(i)}_{(a,b)}(1)\, \frac{\partial h^i(a,b)}{\partial b} \end{bmatrix}. \end{aligned} \tag{4.6}$$

Moreover, if we need to go from the space (a,b) to the space (x,y), we simply need to consider the Jacobian matrix:

$$\boldsymbol{J} = \boldsymbol{J}^0\, \boldsymbol{J}^1, \tag{4.7}$$

or its invert to go in the opposite direction. Finally, it is important to notice, that we did all the developments in two dimensions. However, we can take the exact same rules in the three dimensional case.

4.2.3 Integration with the fast quadrature procedure

As the full permutation approach, the connectivity orientation procedure is also able to feed the efficient quadrature algorithm with oriented bases, and to sort the mesh elements with respect to their (reference) orientation. The only difference with the previous solution, is that the Jacobian matrix (4.7) has to be used for the reference to physical element mapping.

It is worth mentioning that, even if the number of possible oriented bases is reduced by the connectivity approach, this number can remain large. Therefore, oriented bases are generated only for the reference orientations effectively encountered in the mesh. This information is available as soon as the mesh elements are sorted with respect to their orientation.

4.2.4 Analysis

As in the previous section, the proposed method solves the orientation problem. However, this new approach will lead to substantially less oriented bases. Indeed, this new approach do not generate a basis for every possible natural orientation, but only for the reference orientations. To do so, a new mapping was introduced, to go from a natural orientation to a reference orientation. Table 4.2 compares the naive approach and this new, connectivity based, solution.

		Number of bases	
Element type	Number of vertices	Naive version	Connectivity version
Line	2	2	1
Triangle	3	6	1
Quadrangle	4	24	3
Tetrahedron	4	24	1
Pyramid	5	120	15
Prism	6	720	60
Hexahedron	8	40 320	840

Table 4.2: Number of bases generated by the naive and the connectivity approaches.

For the hexahedral case, by analyzing Table 4.2, we can directly notice that the number of possible reference orientations remains large. To give some orders of magnitude, let us analyze the number of reference orientation effectively encountered in a simple mesh: a regular Cartesian mesh of a cube. Table 4.3 summarizes the obtained results for the naive and for the connectivity orientation strategies.

Number of bases	
Naive version	Connectivity version
26	6

Table 4.3: Number of bases constructed for a simple hexahedral mesh.

Chapter 5

Simulation of an ultrahigh finesse Fabry-Pérot superconducting resonator

It's a trap!
— Admiral Gial Ackbar.

In this chapter, we test our high-order finite element tools to simulate an ultrahigh finesse open resonator. Because of its high quality factor, finding a converged value for the damping requires very accurate simulations. Therefore, the use of higher-order numerical techniques seems appropriate. Let us note that a previous work [30], with a second-order finite element discretization on straight meshes, did not succeed to compute a damping independent of the mesh refinement.

5.1 Introduction

In 2012, Serge Haroche and David Jeffrey Wineland were jointly awarded the Nobel prize in physics for *ground-breaking experimental methods that enable measuring and manipulation of individual quantum systems*[1] [81]. Basically, they succeed in building devices, where quantum particles (such as photons or ions) can be trapped [80]. Among other applications of this new technology, this research has opened the door to *real* quantum mechanics experiments [79]. Indeed, until recently, quantum physicists were bound only to *thought* experiments: the most famous of them being probably the Schrödinger's cat experiment [116], bringing out the difficulties, on how to interpret quantum mechanics and especially the principle of quantum superposition.

In 2007, Serge Haroche and his coworkers were able to record the *birth and death of a photon in a cavity* [56, 61], using an ultrahigh finesse Fabry-Pérot superconducting resonator [77]. This open cavity consists of two superconducting (Niobium at 0.8[K]) toroidal mirrors, with

[1] Translated from Swedish: *banbrytande experimentella metoder som möjliggör mätning och styrning av enstaka kvantsystem.*

extremely small geometrical defects (see Figure 5.1), as shown on the geometrical parameters of Table 5.1.

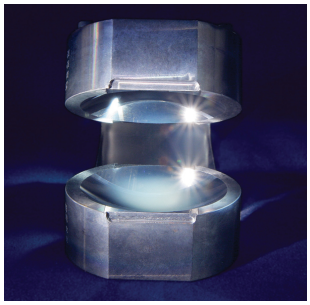


Figure 5.1: Picture of the two mirrors [115].

Diameter	Radius of curvature	Maximum deviation	Distance between apexes
50	40.6 (major) 39.4 (minor)	3×10^{-4} (peak-to-valley)	27.5

Table 5.1: Mirrors geometry in [mm].

The cavity is resonant at 51.1[GHz] with a damping time of approximately 130[ms]. In a very simple way, this high damping time means that, if a photon appears inside the cavity, it will somehow *bounce back and forth* for a very long time without interfering with the outside world. By taking advantage of the extremely long lifetime of the photon inside the cavity, Serge Haroche and his coworkers were then able to record the birth, life and death of a photon.

A few years later, in 2014, attempts were made to model the photon cavity using the classical time-harmonic Maxwell's equations, with the finite element method and a quasimodal analysis [30]: that is, by analysing the eigenmodes of the cavity, the resonant frequency, as well as the damping time, can be recovered. In [30], the authors succeeded in computing the resonant frequency of the cavity. However, they were not able to reach a convergent damping time with respect to the finite element mesh refinement. The purpose of this chapter is to compute the damping time of the photon cavity, using the high-order finite element method, presented in the previous chapters. To the best of our knowledge, the following analysis with high-order finite elements is performed for the first time.

5.2 Why using high-order finite elements?

Since the photon cavity is an open structure, the time-harmonic Maxwell's equations lead to a non-hermitian differential operator. Thus, the eigenvalues of the operator exhibit a non-zero imaginary part. It can be shown [30], that the frequency of an eigenmode can be computed by using the real part of its eigenvalue, and its damping time can be recovered with the imaginary part of its eigenvalue. Using the measurements in Serge Haroche's experiments,

the eigenvalue associated to the 51.1[GHz] mode, with a 130[ms] damping time, should be:

$$\omega^2_{\text{Haroche}} = \left(2\pi f_{\text{Haroche}} + \jmath \frac{2\pi}{\tau_{\text{Haroche}}}\right)^2 = \left(2\pi \times 51.1 \times 10^9 + \jmath \frac{2\pi}{130 \times 10^{-3}}\right)^2,$$
$$\simeq 1.03 \times 10^{23} + \jmath 3.10 \times 10^{13}.$$

Let us now look at the ratio between the real and imaginary parts of $\omega^2_{\text{Haroche}}$:

$$\mathcal{R} = \frac{\Re\left(\omega^2_{\text{Haroche}}\right)}{\Im\left(\omega^2_{\text{Haroche}}\right)} = \mathcal{O}\left(10^9\right).$$

This high real-imaginary parts ratio means, that any computational noise on the eigenvalue $\omega^2_{\text{Haroche}}$ will dramatically impact its imaginary part, and therefore the damping time. This explains the results from [30], where a high-precision was achieved for the real part of $\omega^2_{\text{Haroche}}$, and a quasi-random behavior was observed for the imaginary part. It is worth noticing that this ratio of 10^9 does not mean that the unknown field (in this case the electric field) must be computed with an accuracy of 10^{-9}.

Since high-precision is required for both the real and imaginary parts, related by a very high ratio, a natural choice is to use high-order finite element discretizations.

5.3 Computational setup

Instead of the second-order finite element scheme used in [30] with the software COMSOL[2], we apply the efficient high-order finite element code described in chapter 3, to compute the eigenmodes close the 51.1[GHz] resonant frequency. Among the eigenvalues found in this region of the spectrum, we expect to find one with a high real-imaginary part ratio. This eigenvalue corresponds then to a mode, where the cavity exhibits a high damping time (or equivalently, a high quality factor), which is the photon "trapping" mode. Finally, our objective is to find a converged damping time: *i.e.*, quasi-independent to both a FE discretization order increase and a mesh refinement.

As explained in the introduction, the eigenvalue problems are solved using the SLEPc library[3] [62, 107]. When an **LU** decomposition has to be carried out, the MUMPS library is called, and is set for parallel analysis with the ParMETIS[4] [75] reordering.

Before going any further, a last point must be raised. Since the mirrors are toroidal, the cavity actually exhibits two resonant modes: one associated to each radius. They are both $\text{TEM}_{9,0,0}$ modes, near 51.1[GHz], and separated by 1.2[MHz] [77]. These modes will be referred to as polarity 1 and 2.

[2]Available at: `http://www.comsol.com`.

[3]Available at: `http://slepc.upv.es`.

[4]ParMETIS is a graph partitioner, used by MUMPS for minimizing the fill-in, by reordering the matrix terms; ParMETIS is available at: `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`.

5.3.1 Geometry

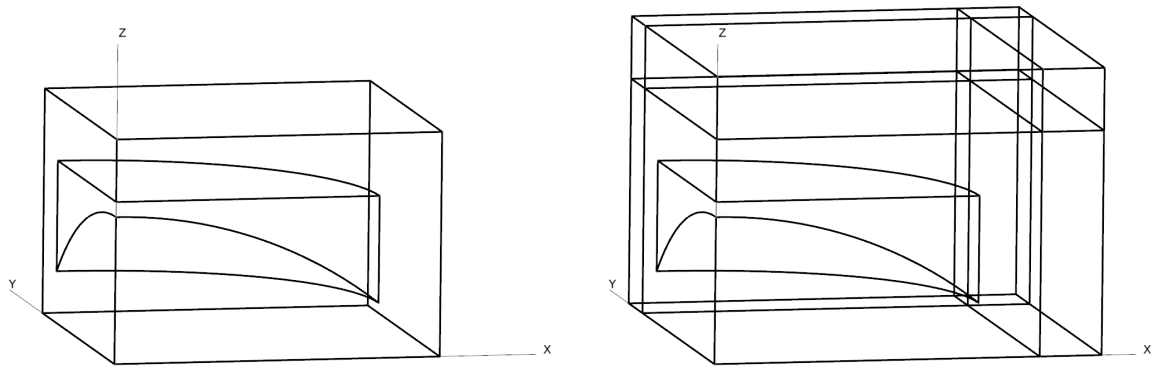
Let us now consider the geometry used for the simulations. In order to reduce the unknowns number, only ¼ of a single mirror will be modeled, as shown in Figure 5.2.

(a) Two mirrors.

(b) One quarter of a single mirror.

Figure 5.2: Mirrors geometry.

Furthermore, the air surrounding our quarter mirror is modeled by a rectangular parallelepiped, as shown in Figure 5.3a. Since the cavity is open, the geometry is infinite: to handle this, a PML surrounding the parallelepiped is added, as depicted in Figure 5.3b. As a first guess, the PML thickness is taken as 1λ (where $\lambda \simeq 5.9$[mm] is the wavelength at 51.1[GHz]), and its distance from the mirror is also set to 1λ. The final geometry is depicted in Figure 5.3b, and was generated by the Open CASCADE[5] CAD[6] engine. This library was driven using the python[7] interface provided by Gmsh[8] [54].



(a) Without a PML.

(b) With a PML.

Figure 5.3: Computational domain.

In order to compare with the results presented in [30], the mirror thickness it taken as 1.415[mm] at the mirror center, while the actual thickness of the mirror in the experiments [77]

[5]Available at: `http://www.opencascade.com`.

[6]Compute-Aided Design.

[7]Available at: `http://www.python.org`.

[8]Gmsh is a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities; available at: `http://gmsh.info`.

was larger. Finally, the computational domain is meshed using the Gmsh mesh engine. Since our objective is to use higher-order finite element discretizations, curved mesh elements are used [72], which allows a precise representation of the curved mirror, while keeping the number of mesh elements under control.

5.3.2 Boundary conditions

As motivated previously, only ¼ of a single mirror is modeled. To simulate the actual configuration, appropriate boundary conditions must then be imposed on the geometry of Figure 5.3b.

1. A null tangential magnetic field on the Oxy plane:

$$\mathbf{n} \times \boldsymbol{\mu}_r^{-1} \, \mathbf{curl} \, \boldsymbol{e} = \mathbf{0} \quad \text{on } Oxy.$$

2. Depending whether polarity 1 or 2 is simulated.

   Polarity 1: a null tangential magnetic field on the Oxz plane, and a null tangential electric field on the Oyz plane:

$$\begin{cases} \mathbf{n} \times \boldsymbol{\mu}_r^{-1} \, \mathbf{curl} \, \boldsymbol{e} = \mathbf{0} & \text{on } Oxz, \\ \mathbf{n} \times \boldsymbol{e} \times \mathbf{n} = \mathbf{0} & \text{on } Oyz. \end{cases}$$

   Polarity 2: a null tangential magnetic field on the Oyz plane, and a null tangential electric field on the Oxz plane:

$$\begin{cases} \mathbf{n} \times \boldsymbol{e} \times \mathbf{n} = \mathbf{0} & \text{on } Oxz, \\ \mathbf{n} \times \boldsymbol{\mu}_r^{-1} \, \mathbf{curl} \, \boldsymbol{e} = \mathbf{0} & \text{on } Oyz. \end{cases}$$

3. A null tangential electric field on the boundary of the PML:

$$\mathbf{n} \times \boldsymbol{e} \times \mathbf{n} = \mathbf{0} \quad \text{on } \partial\Omega_{\text{PML}}.$$

Moreover, since the mirror is made of superconducting material, we assume that it has a zero resistivity:

$$\mathbf{n} \times \boldsymbol{e} \times \mathbf{n} = \mathbf{0} \quad \text{on } \partial\Omega_{\text{Mirror}}.$$

For simplicity, we also consider the frame of the mirror to have a null resistivity:

$$\mathbf{n} \times \boldsymbol{e} \times \mathbf{n} = \mathbf{0} \quad \text{on } \partial\Omega_{\text{Frame}}.$$

5.3.3 Eigenvalue problem

As already explained, our objective is to find the eigenvalues of the photon cavity. In other words, we need to find every possible value of the angular frequency ω, such that:

$$\begin{cases} \mathbf{curl} \left(\boldsymbol{\mu}_r^{-1} \, \mathbf{curl} \, \boldsymbol{e} \right) - \omega^2 \dfrac{\boldsymbol{\varepsilon}_r}{c_0^2} \boldsymbol{e} = \mathbf{0} & \text{on } \Omega, \\ \text{boundary conditions of section 5.3.2,} \end{cases}$$

holds. This equation is derived from (1.5), by exploiting the definition of the wavenumber, and by using the fact that only perfect conductors are used, thus leading to $\tilde{\boldsymbol{\varepsilon}}_r = \boldsymbol{\varepsilon}_r$.

From the finite element point of view, by using (2.11), our eigenvalue problem writes:

$$\begin{cases} \displaystyle\int_\Omega \left(\boldsymbol{\mu}_r^{-1}\,\mathbf{curl}\,\boldsymbol{e}\right)\cdot\left(\mathbf{curl}\,\boldsymbol{e}^i\right)\,\mathrm{d}\Omega - \omega^2\int_\Omega\left(\frac{\boldsymbol{\varepsilon}_r}{c_0^2}\,\boldsymbol{e}\right)\cdot\boldsymbol{e}^i\,\mathrm{d}\Omega = 0 \quad \forall \boldsymbol{e}^i \in V_0(\mathbf{curl},\Omega), \\ \text{boundary conditions of section 5.3.2,} \end{cases}$$

where $V_0(\mathbf{curl},\Omega)$ is a finite-dimensional subspace, of size N, of $H_0(\mathbf{curl},\Omega)$. Moreover, the electrical field $\boldsymbol{e}$ is decomposed as:

$$\boldsymbol{e} = \sum_{j=0}^{N-1} a_j\,\boldsymbol{e}^j \quad \boldsymbol{e}^j \in V_0(\mathbf{curl},\Omega).$$

Thus, the (generalized) eigenvalue problem writes:

$$\text{find every } \omega_k^2, \text{ such that} \quad \mathbf{Ba} = \omega_k^2\mathbf{Ca}. \tag{5.1}$$

By identifying the terms in equation (5.1), we have:

- vector $\mathbf{a} = [a_0,\ldots,a_{N-1}]^T$, that contains the interpolation coefficients of $\boldsymbol{e}$;
- matrix $\mathbf{B}(i,j) = \displaystyle\int_\Omega \left(\boldsymbol{\mu}_r^{-1}\,\mathbf{curl}\,\boldsymbol{e}^j\right)\cdot\left(\mathbf{curl}\,\boldsymbol{e}^i\right)\,\mathrm{d}\Omega$;
- matrix $\mathbf{C}(i,j) = \displaystyle\int_\Omega\left(\frac{\boldsymbol{\varepsilon}_r}{c_0^2}\,\boldsymbol{e}^j\right)\cdot\boldsymbol{e}^i\,\mathrm{d}\Omega$.

It is worth stressing that the eigenvalues of (5.1) are the ω_k^2 and not the ω_k. The eigenvalue problem (5.1) is solved with the Krylov–Schur algorithm [122, 123] from the SLEPc library [62]. The iterative solver relative tolerance is set to 10^{-15}.

5.3.4 Perfectly matched layer

Let us now specify the PML used for the considered simulations. As explained in section 1.3, a Cartesian PML is characterized by its damping functions $\sigma_x(x)$, $\sigma_y(y)$ and $\sigma_z(z)$. We chose to use the following hyperbolic profiles [17]:

$$\begin{cases} \sigma_x(x) = \dfrac{c_0}{X_{\max}+\Delta X-|x|} - \dfrac{c_0}{\Delta X}, \\ \sigma_y(y) = \dfrac{c_0}{Y_{\max}+\Delta Y-|y|} - \dfrac{c_0}{\Delta Y}, \\ \sigma_z(z) = \dfrac{c_0}{Z_{\max}+\Delta Z-|z|} - \dfrac{c_0}{\Delta Z}, \end{cases}$$

where ΔX (or ΔY, or ΔZ) is the thickness on the PML in the x (or y, or z) direction, and where $X_{\max}$ (or $Y_{\max}$, or $Z_{\max}$) is the distance between the center of the mirror and the PML in the x (or y, or z) direction.

5.3.5 Spectral transform

In our cavity problem, it is not required to extract the entire spectrum of (5.1): only the high damping time mode is of interest. Therefore, a spectral transform will be used. The idea, is to apply a preconditioner to the eigenvalue problem, so that the eigenvalues, around a given target, are easier to compute [109].

Practically, the shift-and-invert transform [47, 109] will be used. That is, instead of solving (5.1), the following equivalent (from the eigenvalue point of view) problem is solved:

$$(\mathbf{B} - \sigma\mathbf{C})^{-1}\mathbf{C}\mathbf{a} = \theta_k \mathbf{a},$$

where σ is a well chosen shift, and where $\theta_k = (\omega_k^2 - \sigma)^{-1}$. Since the resonance frequency of the cavity is known to be $f_{\text{Haroche}} = 51.1[\text{GHz}]$ [77], the shift σ is then taken as:

$$\sigma = (2\pi \times f_{\text{Haroche}})^2 = 1.0309 \times 10^{23}.$$

It is worth noticing that the spectral transform involves the $\mathbf{LU}$ decomposition of $(\mathbf{B} - \sigma\mathbf{C})$. This will be handled by the MUMPS solver.

5.4 Simulations

5.4.1 Solution convergence: mesh density and finite element discretization

Based on the computational setup presented before, a first set of simulations was run, in order to assess the convergence of the model. Practically, five tetrahedral meshes were generated, with 3 to 7 mesh elements per wavelength. In each case, 3^{rd}-order mesh elements are used. From the finite element point of view, three discretizations are considered: orders 3, 4 and 5.

For the computational considerations, each simulation is parallelized with 120 MPI processes. Depending on the problem size, these processes will be distributed between 5 and 120 computing nodes. On each node, 64[GB] of memory is available.

For each simulation, three eigenvalues are computed[9]. In every case, only one eigenvalue exhibits a very large damping time. This eigenvalue is thus selected as the resonant one. Figure 5.4 depicts the computed resonance frequencies and damping times for polarity 1.

From the cavity frequency point of view, we may conclude that convergence is achieved. The polarity 1 resonant frequency is thus found at 51.0847[GHz]. On the other hand, from the damping time point of view, convergence is less easy to analyze. Clearly, the third-order simulations have not converged. Forth- and fifth-order simulations seem to converge to a damping time around 35[s] (34.8[s] for the finest fourth-order solution, 35.2[s] for the best fifth-order one).

Compared to the experimental damping time of 130[ms], the computed one is 270 times higher. Two major factors can explain this discrepancy. First, our model is ideal: the mirrors are perfect conductors, and are perfectly aligned. And second, our model consists of only the two mirrors floating alone in the void, while in the actual experiment, the mirrors are

[9] With the spectral transform, shifting the spectrum at the resonance frequency of the cavity.

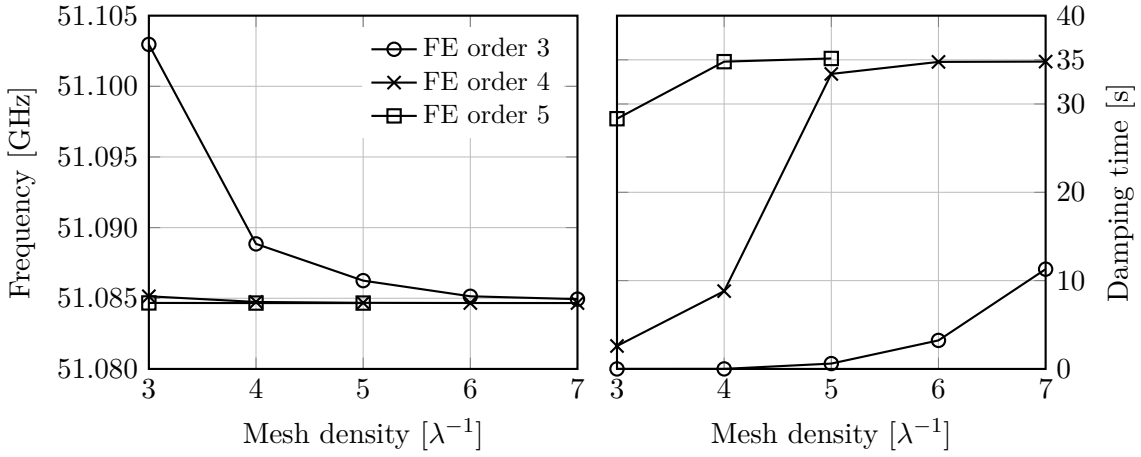
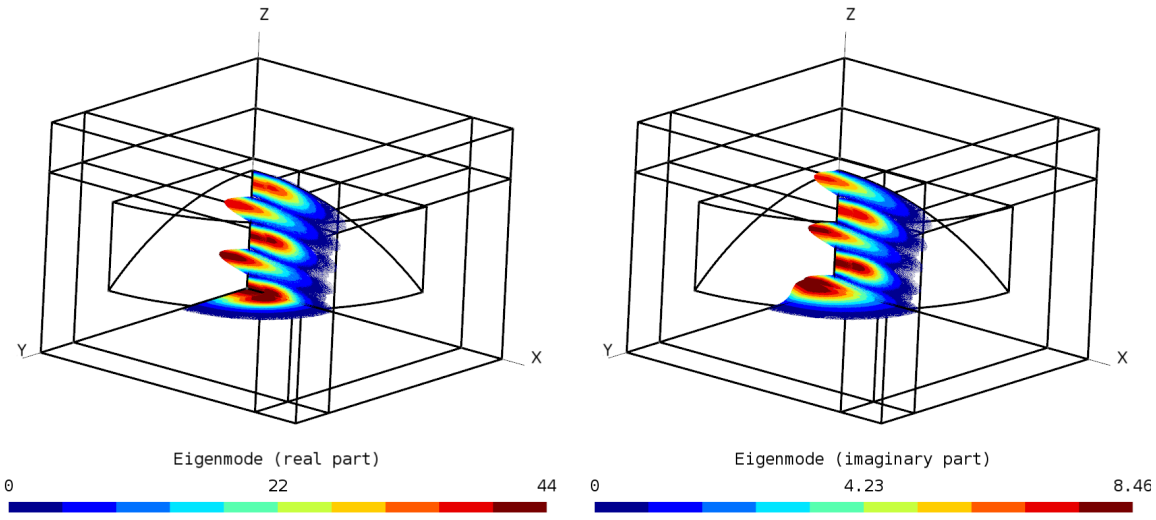


Figure 5.4: Resonance frequencies and damping times: polarity 1 and 3$^{\text{rd}}$-order mesh elements.

surrounded by a more sophisticated device [77]. These two differences lead to a modeled system with less interactions with the outside world, thus a higher expected damping time.

To conclude, and for illustration purposes, Figure 5.5 depicts the eigenmode associated to the cavity resonance (polarity 1). For clarity, the electric field is shown only on the Oxz and Oxy planes. It can be directly seen that polarity 1 is indeed a $\text{TEM}_{9,0,0}$ mode [77].



(a) Real part.

(b) Imaginary part.

Figure 5.5: Eigenmode associated to the cavity resonance (polarity 1).

5.4.2 Polarity 2

In the previous convergence test, only polarity 1 was considered. Let us now focus on polarity 2. Based on the previous simulations, only the fourth and fifth-order finite element solutions will be considered. Figure 5.6 depicts the computed frequencies and damping times for polarity 1 and 2. In addition, the frequency difference between the two polarities is provided in Table 5.2.

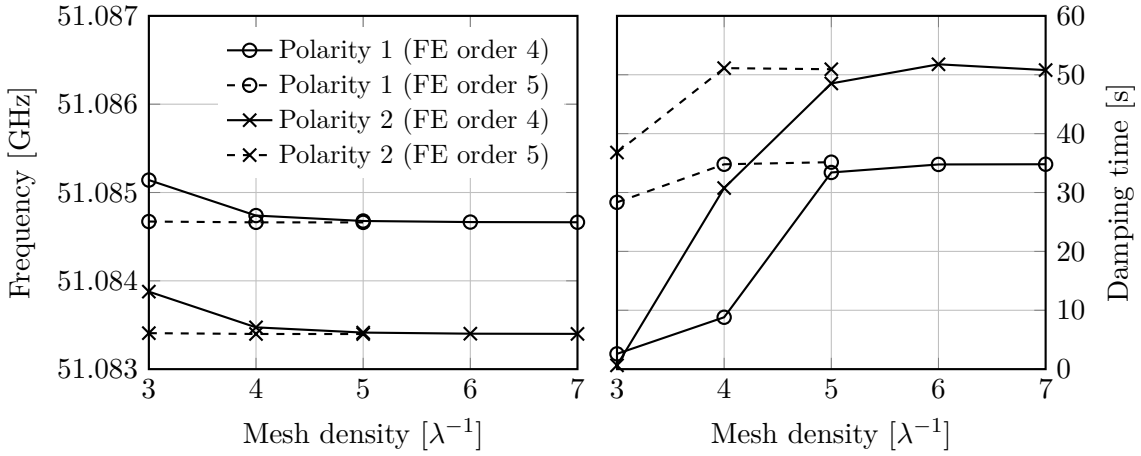


Figure 5.6: Resonance frequencies and damping times: 3^{rd}-order mesh elements, 4^{th} and 5^{th}-order finite element discretization.

	Frequency difference [MHz]	
Mesh density [λ^{-1}]	FE order 4	FE order 5
3	1.2631	1.2638
4	1.2659	1.2638
5	1.2636	1.2639
6	1.2639	—
7	1.2639	—

Table 5.2: Frequency difference between polarity 1 and 2: 3^{rd}-order mesh elements, 4^{th} and 5^{th}-order finite element discretization.

As for polarity 1, we can directly notice that the cavity resonance frequency is converged: the best computed value is 51.0834[GHz]. Moreover, we have reached a frequency difference of 1.26[MHz], which is close to the 1.2[MHz] measured experimentally [77].

For the damping time, we also get results similar to polarity 1: convergence is not strictly achieved, but the damping time seems to stabilize around 50.9[s]. Again, the computed damping time is significantly larger than the measured one.

5.4.3 Mesh curvature

So far, the geometry was meshed with third-order elements. Let us now analyze the impact of the mesh curvature on the simulations. To do so, only polarity 1 is considered for simplicity. Moreover, we will limit ourselves to finite element discretizations of order 4. As done previously, the geometry is meshed with a density of tetrahedra, varying between 3 and 7 elements per wavelength. However, this time, the geometrical order of the elements will range between 1 and 4. Figure 5.7 shows the computed frequencies and damping times.

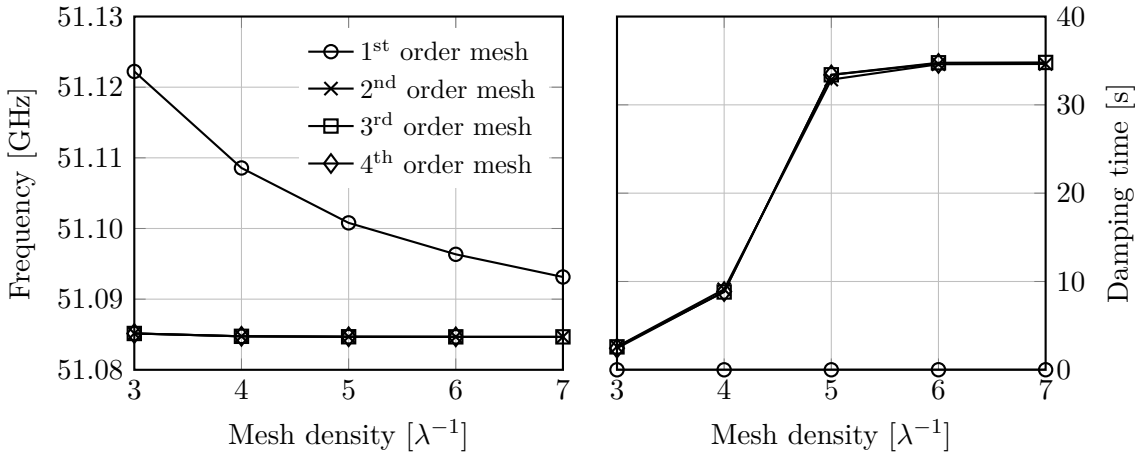


Figure 5.7: Resonance frequencies and damping times: polarity 1 and 4^{th}-order finite element discretization.

By analyzing the data of Figure 5.7, we can directly notice that the damping time does not converge with first-order mesh elements. On the other hand, there is no significant difference, between simulations using higher-order mesh elements. Obviously, straight mesh elements fail to compute the cavity damping time. Since the mirrors shape is curved, approximating it with first-order elements will introduce some kind of numerical rugosity on the surface of the mirrors. This rugosity will not impact dramatically the resonance frequency. However, it can lead to unwanted reflections destroying the stability of the wave. This problem has been noticed in the previous attempt to simulate the photon cavity [30].

A last question remains: can the lack of curvature of the first-order mesh elements be compensated, by a better mesh refinement of the mirrors? To answer this question, let us setup the following simulations. A mesh with 5 tetrahedra per wavelength is used everywhere in the computational domain, except on the surface of the mirror. On this surface, refinements of 5, 10 and 20 elements per wavelength will be used. For each mesh, the geometrical order will range between 1 and 4. From the finite element point of view, a fourth-order discretization will be applied. Only polarity 1 is considered. The simulated cavity damping times are reported in Figure 5.8.

Based on the results of Figure 5.8, we can directly notice that increasing the mesh density on the surface of the mirror, does not help to make the damping time convergent, at least when straight mesh elements are used. Thus, we can conclude that the damping time is highly sensitive to the numerical rugosity introduced by the mesh. Therefore, using curved mesh

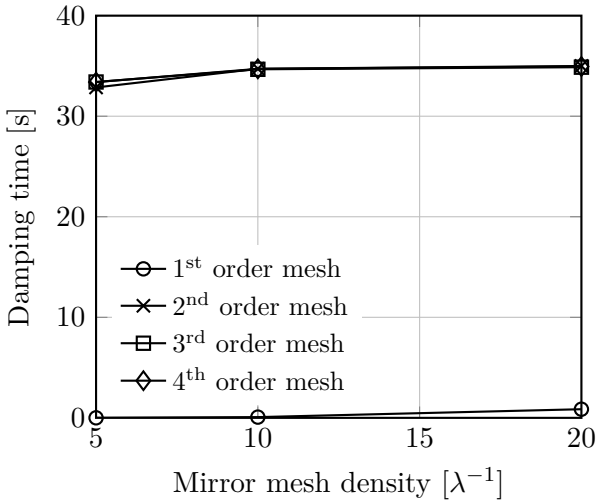


Figure 5.8: Damping times: polarity 1, 5 tetrahedra per wavelength (except on the mirror) and 4^{th}-order finite element discretization.

elements is mandatory to simulate the cavity, so that this virtual rugosity can be alleviated.

5.4.4 Perfectly matched layer sensitivity

Let us now study the solution stability with respect to the parameters of the PML: its distance from the mirror and its thickness. For this analysis, the following setup is used:

- tetrahedral mesh of order 4, with a density of 5 mesh elements per wavelength (λ);
- finite element discretization of order 4;
- a PML thickness ranging from 0.25λ to 2λ;
- two distances from the mirror to the PML, 1λ and 2λ.

The computed damping times are available in Figure 5.9, and missing data were not computed because of memory limitations (see section 5.6). The mean resonance frequency, and the maximum deviation from this mean, is given in Table 5.3.

Distance PML–mirror [λ]	Mean frequency [GHz]	Maximum deviation [GHz]
1	51.08468	2.5×10^{-6}
2	51.08468	2.3×10^{-6}

Table 5.3: Resonance frequency: polarity 1, 5 mesh elements per wavelength, 4^{th}-order mesh elements and 4^{th}-order finite element discretization.

Let us start by analyzing the results presented in Table 5.3. Based on the available data, we can conclude that the resonance frequency is independent to a PML variation. On the other hand, the sensitivity is quite different for the damping time. By looking in Figure 5.9, we cannot conclude, that the damping time is convergent with the PML–mirror distance and

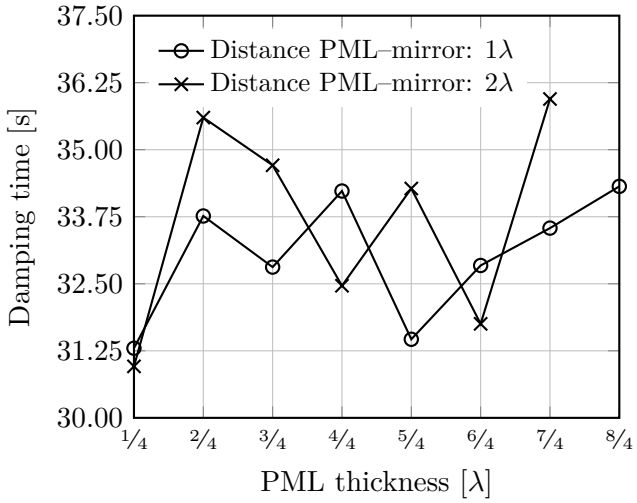


Figure 5.9: Damping times: polarity 1, 5 mesh elements per wavelength, 4^{th}-order mesh elements and 4^{th}-order finite element discretization.

the PML thickness, since the behavior is too oscillatory. However, those variations of the damping time seems to remain bounded. So once again, even if convergence is not reached for the damping time, we are probably close to it.

5.5 First-order simulations

In this section, let us consider another strategy to compute the damping time and the resonance frequency of the cavity: a full first-order discretization. In other words, instead of exploiting higher-order finite element discretizations on coarse curved meshes, let us try a first-order FE approach on a fine straight mesh. Simulations were run for polarity 1 with a mesh density ranging between 10 and 20 mesh elements per wavelength. It is worth noticing that the smallest simulation lead to a linear system of 197792 unknowns, while the largest simulation lead to 11589800 unknowns. Again, larger problems were not possible because of memory limitations.

Figure 5.10 depicts the obtained results. It can be directly noticed that even with a fine mesh, the first-order discretization fails to compute the damping time and, to a lesser extent, the resonance frequency.

5.6 Memory scaling

In the previous section, we systematically ended with the same conclusion: the damping has not completely converged with the simulation parameters, but it seems close to it. To improve the simulations accuracy, we then need to increase the mesh refinement and/or the FE discretization order. However, for now, it is not possible to increase the simulations size: the memory scaling of the MUMPS solver seems to be reached. This phenomenon is illustrated in the two examples that follow.

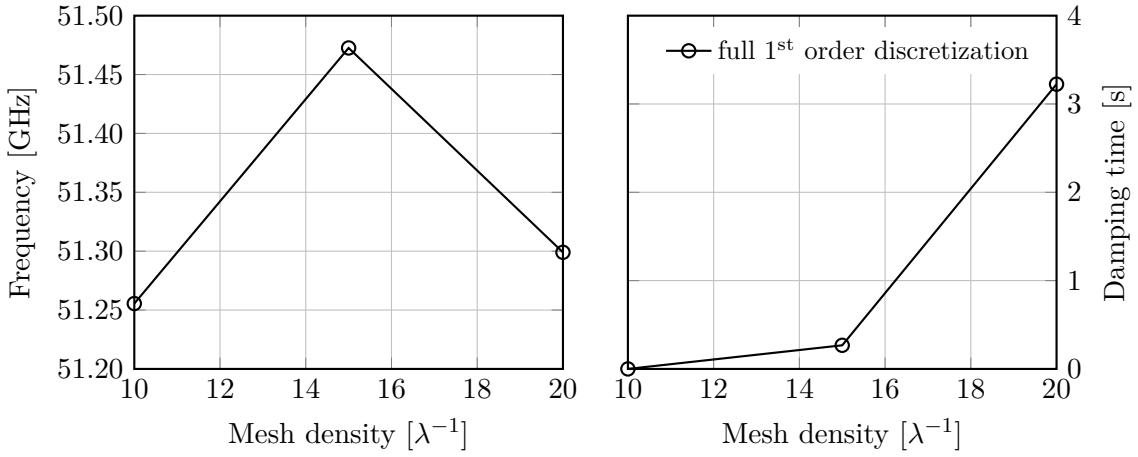


Figure 5.10: Resonance frequencies and damping times: polarity 1 and full first-order discretization.

5.6.1 Two fifth-order test cases

Let us look at the following simulation:

- finite element discretization of order 5;
- mesh curvature of order 5;
- polarity 1.

We then consider two mesh densities: 5 and 6 mesh elements per wavelength. These two simulations lead to, respectively, 8019588 and 13363722 unknowns. The test cases were launched on 120 computing nodes, with 64[GB] of memory on each node. The smallest simulation ran successfully, and the largest failed because not enough memory was available at the MUMPS **LU** factorization stage. So, we ran this latter test case on 240 nodes... and it also failed for the same reason.

To summarize, we increased the system size by 67%, and we increased by 100% the total available memory. Nevertheless, not enough memory was available. Thus, our simulations have probably reached the memory scaling limit of the MUMPS solver (with the ParMETIS reordering).

To illustrate better this limit, let us look at the peak virtual memory allocated by each process, for the 8019588 unknowns problem, as shown in Figure 5.11. Statistics are available in Table 5.4.

Mean value	Standard deviation	Maximum value	Minimum value
17[GB]	3[GB]	28[GB]	13[GB]

Table 5.4: Peak virtual memory allocated: statistics (8019588 unknowns, fifth-order case).

From these data, we can see that the memory usage is not uniformly distributed across the computing processes. On average, we use 17[GB] with a quite low standard deviation.

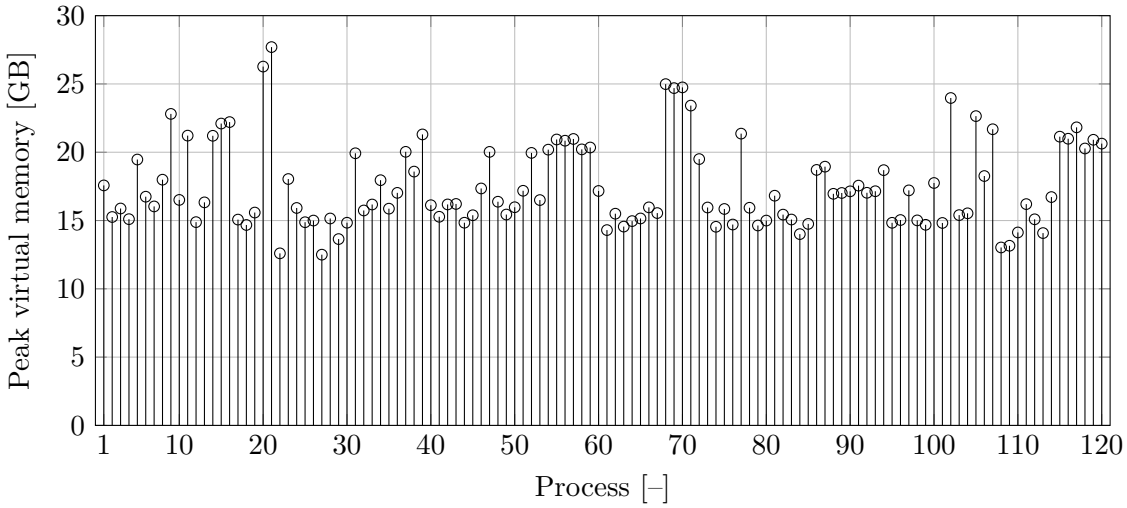


Figure 5.11: Peak virtual memory allocated per node: 8019588 unknowns, fifth-order case.

However, we have two spikes above 25.5[GB] (*i.e.* 50% above the mean). It is those spikes, that will limit the memory scaling of our simulations.

5.6.2 Two fourth-order test cases

Let us take another example:

- finite element discretization of order 4;
- tetrahedral mesh of order 4, with a density of 7 mesh elements per wavelength;
- polarity 1.

We ran successfully this setup (11443760 unknowns) on 120 and 240 computing nodes. The peak virtual memory distribution is available in Figure 5.12, and statistics are available in Table 5.5.

Nodes	Mean value	Standard deviation	Maximum value	Minimum value
120	28[GB]	7[GB]	61[GB]	17[GB]
240	19[GB]	7[GB]	46[GB]	13[GB]

Table 5.5: Peak virtual memory allocated: statistics (120 and 240 nodes, 11443760 unknowns, fourth-order case).

From the above results, we have, once again, some memory spikes far above the mean value. Let us focus only on the two largest spikes: 46[GB] (240 nodes) and 61[GB] (120 nodes). We can directly notice, that by increasing by 100% the total available memory, the largest spike is decreased by only 25%, thus limiting the scaling performances.

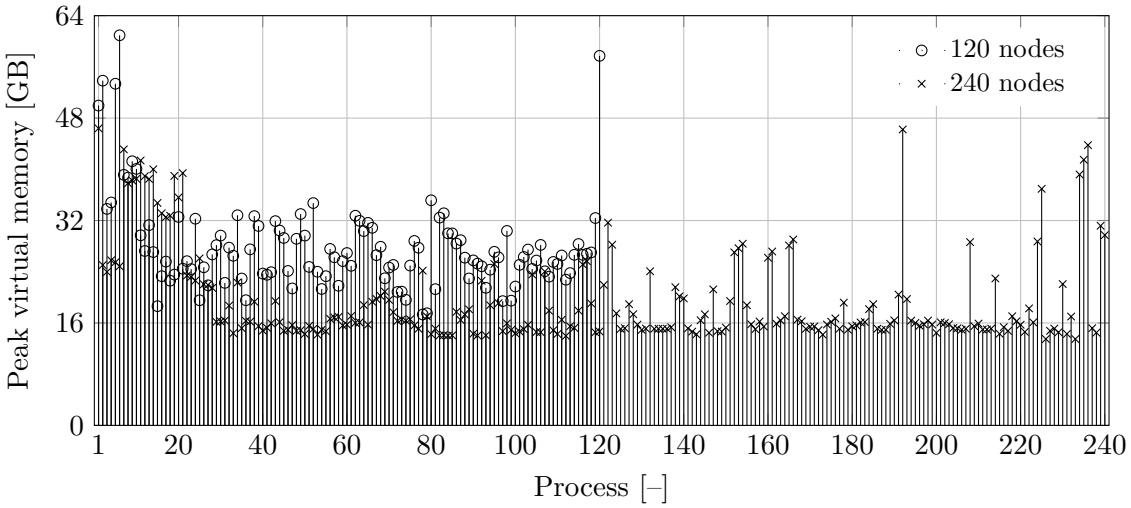


Figure 5.12: Peak virtual memory allocated per node: 120 and 240 nodes, 11443760 unknowns, fourth-order case.

5.7 Simulation time

Before concluding this chapter, let us give some order of magnitude on the wall clock time taken by the simulations. The smallest simulation (3 mesh elements per wavelength with a third-order FE discretization) took less than 1 minute and counted 517556 unknowns. It was run using 5 computing nodes. On the other hand, the largest simulation (7 mesh elements per wavelength with a fourth-order FE discretization) took less than 11 hours and counted 11443760 unknowns. It was run using 120 computing nodes.

5.8 Conclusion

In this chapter, we have simulated the photon cavity designed by Serge Haroche and his coworkers, for recording the *birth and death of photon* [56], by using the classical electromagnetic theory and the finite element method. This problem has been already treated in [30], with straight tetrahedral meshes and a second-order finite element discretization. Unfortunately, the authors were not able to compute the damping time, because of the high sensitivity with respect to the mesh refinement.

In this work, we used both curved mesh elements and high-order finite element discretizations. A few high-precision simulations are still missing, to claim that the damping time convergence is reached. Indeed, because of the memory scaling limitations, simulations larger than roughly 11×10^6 were not possible. However, comparing to [30], far more stable results were obtained. To the best of our knowledge, this is the first time mesh convergence is achieved on this problem.

Let us conclude by a few ideas that could improve this memory scaling limitation.

1. Using another reordering strategy to minimize the direct solver fill-in (*e.g.*, the PT-

Scotch[10] [26] reordering instead of the ParMETIS reordering of the MUMPS solver).

2. Using another finite element matrix distribution across the computing nodes, to help the direct solver reordering process.
3. Using a strategy to compute the interior of the spectrum[11], that does not require an **LU** decomposition, such as the Jacobi-Davidson method [120].
4. Using a domain decomposition method for solving eigenvalue problem [90, 91].

Regarding the last point, let us mention that domain decomposition methods are treated in chapter 6. However, in this context, only direct problems will be discussed: the extension to eigenvalue problem is one of our perspectives for future investigations.

[10] PT-Scotch is another graph partitioner, used by MUMPS for minimizing the fill-in, by reordering the matrix terms; PT-Scotch is available at: `http://www.labri.fr/perso/pelegrin/scotch/`.

[11] That is not just the first lowest (or highest) eigenvalues, as in our case.

Chapter 6

Non-overlapping optimized Schwarz algorithm

As already mentioned in chapter 1, and as experienced in chapter 5, direct solvers do not scale from a memory point of view, because of the fill-in effect, thus limiting the size of the simulations. In this chapter, we present some domain decomposition methods, which are promising alternatives to both direct and iterative methods, when handling high-frequency wave problems. As it will be presented, the key idea of this approach is to couple both direct and iterative methods. This chapter concludes by a performance analysis through numerical experiments.

6.1 Introduction

Historically, the domain decomposition method, or DDM, was introduced by Hermann Schwarz, as a mathematical tool for proving the Dirichlet principle[1] [51, 118]. Years later, this mathematical tool was revisited by Pierre-Louis Lions as a parallel computing strategy [51, 88].

Let us note that in this thesis, we will only focus on a sub-class of DDM: the so-called *optimized Schwarz methods*. Other families of domain decompositions exist, such as the Schur complement method or the finite element tearing and interconnect method. More details can be found in [36, 50, 59, 108].

Let us introduce this domain decomposition approach, using the following classical example. The idea is to solve Laplace's equation on an arbitrary domain Ω, composed by two overlapping simpler geometries, Ω_0 and Ω_1, on which the solution can be easily computed (using Fourier analysis for instance), as shown in Figure 6.1:

$$\text{Find } p(x,y) \text{ such that: } \begin{cases} \operatorname{div} \mathbf{grad}\, p = 0 & \text{on } \Omega, \quad (6.1a) \\ p = f & \text{on } \partial\Omega, \quad (6.1b) \end{cases}$$

where $f(x,y)$ is a known function defined on the boundary of Ω. In order to solve (6.1), the

[1] The Dirichlet principle states that, if $p(x,y)$ is the solution of Laplace's equation $\operatorname{div} \mathbf{grad}\, p = 0$ on a bounded domain Ω, with the Dirichlet boundary condition $p = f$ on $\partial\Omega$, then $p(x,y)$ corresponds to the infimum of the Dirichlet integral $\int_\Omega |\operatorname{div} \mathbf{grad}\, v|^2 \, \mathrm{d}\Omega$ over all functions $v(x,y)$ satisfying $v = f$ on $\partial\Omega$.

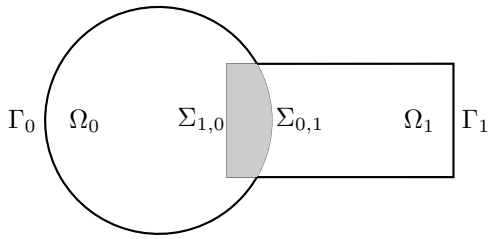


Figure 6.1: Domain considered for the domain decomposition method.

iterative scheme below can be constructed:

$$\begin{cases} \operatorname{div} \mathbf{grad}\, p_0^{n+1} = 0 & \text{on } \Omega_0, \quad (6.2a) \\ p_0^{n+1} = p_1^n & \text{on } \Sigma_{0,1}, \quad (6.2b) \\ p_0^{n+1} = f & \text{on } \Gamma_0, \quad (6.2c) \\ \\ \operatorname{div} \mathbf{grad}\, p_1^{n+1} = 0 & \text{on } \Omega_1, \quad (6.2d) \\ p_1^{n+1} = p_0^{n+1} & \text{on } \Sigma_{1,0}, \quad (6.2e) \\ p_1^{n+1} = f & \text{on } \Gamma_1, \quad (6.2f) \end{cases}$$

where p_i^n is the solution on Ω_i at iteration n, where Γ_i is such that $\Gamma_i = \partial\Omega_i \cap \partial\Omega$, and where $\Sigma_{i,j}$ is such that $\Sigma_{i,j} = \partial\Omega_i \cap \overline{\Omega_j}$. This scheme can be initialized by the initial guess: $p_1^0 = 0$. By analyzing (6.2), we can directly notice that the proposed algorithm *alternatively* solves Laplace's equation on Ω_0 and Ω_1, hence its name: the alternating Schwarz method. Moreover, we can see that each sub-problem (on Ω_0 or Ω_1) makes use of the solution of the neighbor domain as Dirichlet boundary condition. Furthermore, it can be proved [88] that this method converges to the solution of the original problem (6.1). More precisely, after a sufficient number of iterations, we have that:

$$\begin{cases} p_0^n(\mathbf{x}) = p(\mathbf{x}) & \forall \mathbf{x} \in \Omega_0, \\ p_1^n(\mathbf{x}) = p(\mathbf{x}) & \forall \mathbf{x} \in \Omega_1, \end{cases}$$

with $p(\mathbf{x})$ the solution of (6.1). We call this type of procedure a *fixed point* algorithm. Indeed, it stops as soon as the quantities p_i^n do not evolve anymore: *i.e.*, once they are fixed.

Obviously, this former approach is purely sequential. However, as already stated, a parallel version of the previous algorithm has been proposed. In this case, the following fixed point iterative scheme is used:

$$\begin{cases} \operatorname{div} \mathbf{grad}\, p_0^{n+1} = 0 & \text{on } \Omega_0, \quad (6.3a) \\ p_0^{n+1} = p_1^n & \text{on } \Sigma_{0,1}, \quad (6.3b) \\ p_0^{n+1} = f & \text{on } \Gamma_0, \quad (6.3c) \\ \\ \operatorname{div} \mathbf{grad}\, p_1^{n+1} = 0 & \text{on } \Omega_1, \quad (6.3d) \\ p_1^{n+1} = p_0^n & \text{on } \Sigma_{1,0}, \quad (6.3e) \\ p_1^{n+1} = f & \text{on } \Gamma_1. \quad (6.3f) \end{cases}$$

This time, p_0^{n+1} and p_1^{n+1} can be independently computed. Once again, it can be proved [88] that this method converges to the solution of the original problem (6.1). Because of its concurrent nature, this algorithm is called the parallel Schwarz method.

So far, we considered that Ω_0 and Ω_1 were overlapping, which can be a severe restriction [89]. In the case of a non-overlapping decomposition, as depicted in Figure 6.2, a parallel Schwarz algorithm can also be constructed. However, in this case, the sub-problems cannot be glued anymore by using Dirichlet conditions, but by using Robin (or impedance) conditions. For two sub-domains, we thus can write the following scheme:

$$\begin{cases} \operatorname{div} \mathbf{grad}\, p_0^{n+1} = 0 & \text{on } \Omega_0, \quad (6.4a) \\ (\mathbf{n}_0 \cdot \mathbf{grad} + \lambda)\, p_0^{n+1} = (\mathbf{n}_0 \cdot \mathbf{grad} + \lambda)\, p_1^{n} & \text{on } \Sigma_{0,1}, \quad (6.4b) \\ p_0^{n+1} = f & \text{on } \Gamma_0, \quad (6.4c) \\ \\ \operatorname{div} \mathbf{grad}\, p_1^{n+1} = 0 & \text{on } \Omega_1, \quad (6.4d) \\ (\mathbf{n}_1 \cdot \mathbf{grad} + \lambda)\, p_1^{n+1} = (\mathbf{n}_1 \cdot \mathbf{grad} + \lambda)\, p_0^{n} & \text{on } \Sigma_{1,0}, \quad (6.4e) \\ p_1^{n+1} = f & \text{on } \Gamma_1, \quad (6.4f) \end{cases}$$

where $\mathbf{n}_i$ is the unit vector outwardly oriented normal to Ω_i, and where λ belongs to $\mathbb{R}_0^+$. Let us note that classically, equations (6.4b) and (6.4e) are referred to as the *transmission conditions* of the non-overlapping domain decomposition algorithm.

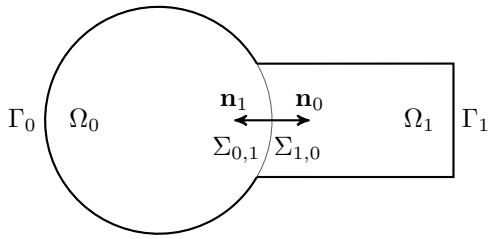
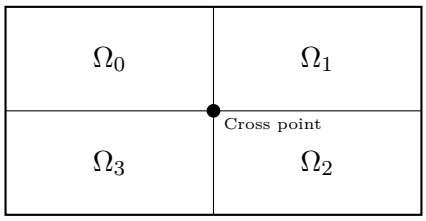


Figure 6.2: Non-overlapping domain decomposition.

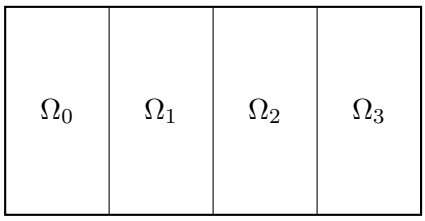
This approach has also been proved convergent for Laplace's equation [89]. The question of the optimal choice for λ remains, and will be treated in sections 6.2 and 6.3 for time-harmonic wave problems. In the remaining of this thesis, we will focus only on the non-overlapping variants of the parallel Schwarz method.

To conclude this introduction, a last point must be raised. So far, we considered a decomposition with only two sub-domains. Fortunately, there are no difficulties to extend the non-overlapping Schwarz algorithm to more sub-domains, as long as an interface is shared by no more than two neighbors, as depicted in Figure 6.3b. For more general situations, exhibiting the so-called cross points (see Figure 6.3a), a special treatment is needed [14, 22, 52, 103]. In this thesis, a given interface will be allowed to share at most two sub-domains, thus avoiding this problematic scenario. More precisely, our decompositions must satisfy [88]:

$$\text{for all distinct } i,j,k \in \{0,\dots,N-1\},\ \text{if } \Omega_i \cap \Omega_j \neq \emptyset \text{ and } \Omega_i \cap \Omega_k \neq \emptyset,\ \text{then } \Omega_j \cap \Omega_k = \emptyset, \quad (6.5)$$



(a) Decomposition with more than two sub-domains sharing a common interface.



(b) Decomposition with at most two sub-domains sharing a common interface.

Figure 6.3: Different types of decompositions.

where N is the total number of sub-domains. With this assumption, the non-overlapping parallel Schwarz algorithm writes:

$$\begin{cases} \operatorname{div} \mathbf{grad}\, p_i^{n+1} = 0 & \text{on } \Omega_i,\ \forall i \in \{0,\dots,N-1\}, \quad (6.6a) \\ (\mathbf{n}_i \cdot \mathbf{grad} + \lambda)\, p_i^{n+1} = (\mathbf{n}_i \cdot \mathbf{grad} + \lambda)\, p_j^n & \text{on } \Sigma_{i,j},\ \forall i \in \{0,\dots,N-1\},\ \forall j \in D_i, \quad (6.6b) \\ p_i^{n+1} = f & \text{on } \Gamma_i,\ \forall i \in \{0,\dots,N-1\}, \quad (6.6c) \end{cases}$$

where

$$D_i = \Big\{ j \in \{0,\dots,N-1\} \text{ such that } j \neq i \text{ and } \Sigma_{i,j} \neq \emptyset \Big\}. \tag{6.7}$$

6.2 Time-harmonic wave problems

In the previous section, we presented the non-overlapping Schwarz method for many sub-domains (6.6) and Laplace's equation. The cross point problem was avoided by requiring that the domain decomposition satisfies (6.5). Let us now consider the case of time-harmonic waves.

6.2.1 Acoustic case

For simplicity, we start by the scalar acoustic case on the infinite two-dimensional domain $\Omega = \mathbb{R} \times \mathbb{R}$. This domain is further decomposed in two sub-domains $\Omega_0 =]-\infty, 0] \times \mathbb{R}$ and $\Omega_1 = [0, +\infty[\times \mathbb{R}$. Using the non-overlapping Schwarz method, the following iterative scheme can be written:

$$\begin{cases} (\operatorname{div} \mathbf{grad} + k^2) p_0^{n+1} = 0 & \text{on } \Omega_0, \\ (\mathbf{n}_0 \cdot \mathbf{grad} + \lambda)\, p_0^{n+1} = (\mathbf{n}_0 \cdot \mathbf{grad} + \lambda)\, p_1^n & \text{on } \Sigma_{0,1}, \\ \\ (\operatorname{div} \mathbf{grad} + k^2) p_1^{n+1} = 0 & \text{on } \Omega_1, \\ (\mathbf{n}_1 \cdot \mathbf{grad} + \lambda)\, p_1^{n+1} = (\mathbf{n}_1 \cdot \mathbf{grad} + \lambda)\, p_0^n & \text{on } \Sigma_{1,0}. \end{cases}$$

Moreover, we require that the solution satisfies the Sommerfeld radiation condition (1.19). Let us note that since there is no overlap, we have $\mathbf{n}_0 = -\mathbf{n}_1$. Thus, the last scheme can be

simplified in:

$$\begin{cases} (\operatorname{div}\mathbf{grad}+k^2)p_0^{n+1}=0 & \text{on } \Omega_0, \quad (6.8a)\\ (\mathbf{n}_0\cdot\mathbf{grad}+\lambda)\,p_0^{n+1}=(\mathbf{n}_0\cdot\mathbf{grad}+\lambda)\,p_1^{n} & \text{on } \Sigma_{0,1}, \quad (6.8b)\\ \\ (\operatorname{div}\mathbf{grad}+k^2)p_1^{n+1}=0 & \text{on } \Omega_1, \quad (6.8c)\\ (-\mathbf{n}_0\cdot\mathbf{grad}+\lambda)\,p_1^{n+1}=(-\mathbf{n}_0\cdot\mathbf{grad}+\lambda)\,p_0^{n} & \text{on } \Sigma_{1,0}. \quad (6.8d)\end{cases}$$

For the Laplace case, the parameter λ of the Robin conditions was allowed to be any positive real number. This is unfortunately not the case for the acoustics. Let us analyze the convergence of the Schwarz algorithm (6.8) on our simple domain $\Omega=\Omega_0\cap\Omega_1$. Let us start be taking the Fourier transform of $p_i^n(x,y)$ in the y direction:

$$\begin{cases}\hat{p}_i^n(x,s)=\mathcal{F}\left[p_i^n(x,y)\right]=\displaystyle\int_{-\infty}^{+\infty}e^{-\jmath sy}p_i^n(x,y)\,\mathrm{d}y,\\ p_i^n(x,y)=\mathcal{F}^{-1}\left[\hat{p}_i^n(x,s)\right]=\dfrac{1}{2\pi}\displaystyle\int_{-\infty}^{+\infty}e^{+\jmath sy}\hat{p}_i^n(x,s)\,\mathrm{d}s.\end{cases}$$

Then, by inserting $\hat{p}_i^n$ in (6.8), we have:

$$\begin{cases}\dfrac{\partial^2\hat{p}_0^{n+1}}{\partial x^2}+(k^2-s^2)\hat{p}_0^{n+1}=0 & \forall x<0,\forall s\in\mathbb{R}, \quad (6.9a)\\ \dfrac{\partial\hat{p}_0^{n+1}}{\partial x}+\lambda\,\hat{p}_0^{n+1}=\dfrac{\partial\hat{p}_1^{n}}{\partial x}+\lambda\,\hat{p}_1^{n} & x=0,\forall s\in\mathbb{R}, \quad (6.9b)\\ \\ \dfrac{\partial^2\hat{p}_1^{n+1}}{\partial x^2}+(k^2-s^2)\hat{p}_1^{n+1}=0 & \forall x>0,\forall s\in\mathbb{R}, \quad (6.9c)\\ -\dfrac{\partial\hat{p}_1^{n+1}}{\partial x}+\lambda\,\hat{p}_1^{n+1}=-\dfrac{\partial\hat{p}_0^{n}}{\partial x}+\lambda\,\hat{p}_0^{n} & x=0,\forall s\in\mathbb{R}. \quad (6.9d)\end{cases}$$

Furthermore, the solution of this ordinary differential equation is simply:

$$\hat{p}_i^{n+1}=A_ie^{+\alpha(s)x}+B_ie^{-\alpha(s)x},i\in\{1,2\}, \quad (6.10)$$

where[2]

$$\alpha=\begin{cases}\sqrt{s^2-k^2} & \text{if } |s|\geq k, \quad (6.11a)\\ -\jmath\sqrt{k^2-s^2} & \text{if } |s|\leq k. \quad (6.11b)\end{cases}$$

Classically, solutions where $|s|>k$ are called *evanescent modes*, and solutions where $|s|<k$ are called *propagating modes*; the limit case $|s|=k$ is referred to as the *cutoff mode*.

Because of the Sommerfeld radiation condition (1.19), we must set $A_1=0$ and $B_0=0$. Therefore, equation (6.10) becomes:

$$\begin{cases}\hat{p}_0^{n+1}(x,s)=\hat{p}_0^{n+1}(0,s)e^{\alpha(s)x} & \forall x<0, \quad (6.12a)\\ \hat{p}_1^{n+1}(x,s)=\hat{p}_1^{n+1}(0,s)e^{-\alpha(s)x} & \forall x>0. \quad (6.12b)\end{cases}$$

[2]Let us remember that with our time convention in $e^{-\jmath\omega t}$, the spatial component of the solution is of the form $e^{+\jmath kx}$ with $x>0$.

Let us now insert $\hat{p}_0^{n+1}(x,s)$ and $\hat{p}_1^{n+1}(x,s)$ in the transmission conditions (6.9b) and (6.9d). By remarking that

$$\begin{cases} \dfrac{\partial \hat{p}_0^{n+1}(x,s)}{\partial x} = \alpha\, \hat{p}_0^{n+1}(x,s), \\ \dfrac{\partial \hat{p}_1^{n+1}(x,s)}{\partial x} = -\alpha\, \hat{p}_1^{n+1}(x,s), \end{cases}$$

we may write:

$$\begin{cases} \alpha\, \hat{p}_0^{n+1}(0,s) + \lambda\, \hat{p}_0^{n+1}(0,s) = -\alpha\, \hat{p}_1^{n}(0,s) + \lambda\, \hat{p}_1^{n}(0,s), \\ \alpha\, \hat{p}_1^{n+1}(0,s) + \lambda\, \hat{p}_1^{n+1}(0,s) = -\alpha\, \hat{p}_0^{n}(0,s) + \lambda\, \hat{p}_0^{n}(0,s). \end{cases}$$

By solving this equation, we finally have:

$$\begin{cases} \hat{p}_0^{n+1}(0,s) = \rho^2(s)\, \hat{p}_0^{n-1}(0,s), & \text{(6.13a)} \\ \hat{p}_1^{n+1}(0,s) = \rho^2(s)\, \hat{p}_1^{n-1}(0,s), & \text{(6.13b)} \end{cases}$$

where the complex function $\rho(s)$ is such that

$$\rho(s) = \frac{-\alpha(s) + \lambda}{+\alpha(s) + \lambda} \in \mathbb{C}. \tag{6.14}$$

With the equations (6.12), (6.13) and (6.14), we may conclude that the scheme is convergent if and only if $|\rho| < 1$.

As a possible choice for λ, Bruno Després proposed to take $\lambda = -\jmath k$ [32]. By substituting λ in (6.14), we have:

$$|\rho(s)|\,\Big|_{\lambda=-\jmath k} = \left|\frac{-\alpha(s) - \jmath k}{+\alpha(s) - \jmath k}\right|.$$

Because of (6.11), we need to consider the following two scenarios: $s = \sqrt{a}\, k$ with $a \leq 1$ or $a \geq 1$. We have then:

$$|\rho(a)|\,\Big|_{\lambda=-\jmath k} = \begin{cases} \left|\dfrac{+\sqrt{1-a} - 1}{-\sqrt{1-a} - 1}\right| \leq 1 & \forall a \leq 1, \\ \left|\dfrac{-\sqrt{a-1} - \jmath}{+\sqrt{a-1} - \jmath}\right| = 1 & \forall a \geq 1. \end{cases}$$

In other words, our domain decomposition scheme (6.8), with a Robin parameter set to $\lambda = -\jmath k$, is converging for the interface modes $|s| < k$, and is stagnating for the interface modes $|s| \geq k$. It is worth recalling, that the above convergence analysis was made for a very simple case. However a more general proof is available in [32].

While not optimal, the domain decomposition algorithm (6.8), with the choice $\lambda = -\jmath k$, converges when using a Krylov solver[3]. This topic will be addressed in section 6.4. However, even without Krylov acceleration, it is possible to design convergent schemes for time-harmonic wave problems. Let us go back to equation (6.14): we can easily notice that $\rho(s)$ can be set to zero by taking $\lambda = -\alpha(s)$. This will thus lead to an optimal convergence of the domain decomposition scheme[4]! However, let us remark that taking λ as function of s in

[3] However, the Krylov solver will converge slowly.

[4] The algorithm will then converge in two steps [53].

the Fourier plane, means that λ becomes a (differential or integral) operator in the regular Cartesian coordinates system. So, instead of a Robin condition with a scalar parameter λ, it is a better choice to use a generalized Robin condition with an operator $\mathcal{S}$. Schwarz methods using this approach are referred to as optimized Schwarz methods, and are the topic of section 6.3.

To conclude this part on time-harmonic acoustic wave equation, let us remark that the scheme (6.8) can be easily generalized, to the case of many sub-domains, as follow:

$$\begin{cases} (\operatorname{div}\mathbf{grad}+k^2)p_i^{n+1} = 0 & \text{on } \Omega_i, & \text{(6.15a)} \\ (\mathbf{n}_i\cdot\mathbf{grad}+\mathcal{S})\,p_i^{n+1} = (\mathbf{n}_i\cdot\mathbf{grad}+\mathcal{S})\,p_j^n & \text{on } \Sigma_{i,j}, \forall j\in D_i, & \text{(6.15b)} \\ p_i^{n+1} = f & \text{on } \Gamma_i, & \text{(6.15c)} \end{cases}$$

for all $i \in \{0,\dots,N-1\}$ (N being the number of sub-domains), where D_i is defined as in (6.7), and where the Robin interface condition has been replaced by a generalized one. Let us also remark that we introduced boundary Dirichlet conditions through equation (6.15c).

6.2.2 Electromagnetic case

Let us now focus on electromagnetic time-harmonic waves. A scheme similar the the acoustic one can be derived [33]. Since no new features need to be introduced, the many sub-domains non-overlapping optimized parallel Schwarz method is directly presented:

$$\begin{cases} (\mathbf{curl}\,\mathbf{curl}-k^2)\boldsymbol{e}_i^{n+1} = \mathbf{0} & \text{on } \Omega_i, & \text{(6.16a)} \\ \gamma_i^t\left(\mathbf{curl}\,\boldsymbol{e}_i^{n+1}\right)+\mathcal{S}\left[\gamma_i^T\left(\boldsymbol{e}_i^{n+1}\right)\right] = \gamma_i^t\left(\mathbf{curl}\,\boldsymbol{e}_j^{n}\right)+\mathcal{S}\left[\gamma_i^T\left(\boldsymbol{e}_j^{n}\right)\right] & \text{on } \Sigma_{i,j}, \forall j\in D_i, & \text{(6.16b)} \\ \gamma_i^T\left(\boldsymbol{e}_i^{n+1}\right) = \boldsymbol{f} & \text{on } \Gamma_i, & \text{(6.16c)} \end{cases}$$

for all $i \in \{0,\dots,N-1\}$ (N being the number of sub-domains), and with

$$\begin{cases} \gamma_i^T(\boldsymbol{a}):\ \boldsymbol{a}\mapsto \mathbf{n}_i\times\boldsymbol{a}\times\mathbf{n}_i, \\ \gamma_i^t(\boldsymbol{a}):\ \boldsymbol{a}\mapsto \mathbf{n}_i\times\boldsymbol{a}. \end{cases}$$

In the pure Robin interface condition case, the operator $\mathcal{S}$ is taken as the scalar value [33]:

$$\mathcal{S} = \lambda = +\jmath k.$$

With this choice, a behavior similar to the acoustic case is observed: the interface modes bellow k are convergent, and the modes above k are stagnating. However, as remarked previously, the operator $\mathcal{S}$ can be chosen to optimize the convergence rate of the scheme (see section 6.3).

6.3 Optimized transmission conditions

As previously mentioned, when analyzing the convergence rate of the non-overlapping parallel Schwarz method, it is extremely advantageous to use a generalized Robin (impedance)

condition, instead of a classical one, to link two sub-domains. The transmission condition between two sub-domains, Ω_i and Ω_j for instance, writes then:

$$\begin{aligned}
(\mathbf{n}_i \cdot \mathbf{grad} + \mathcal{S})\, p_i^{n+1} &= (\mathbf{n}_i \cdot \mathbf{grad} + \mathcal{S})\, p_j^n && \text{on } \Sigma_{i,j}, \quad \text{(Acoustics)} \\
\gamma_i^t \left(\mathbf{curl}\, \boldsymbol{e}_i^{n+1}\right) + \mathcal{S}\left[\gamma_i^T \left(\boldsymbol{e}_i^{n+1}\right)\right] &= \gamma_i^t \left(\mathbf{curl}\, \boldsymbol{e}_j^{n}\right) + \mathcal{S}\left[\gamma_i^T \left(\boldsymbol{e}_j^{n}\right)\right] && \text{on } \Sigma_{i,j}, \quad \text{(Maxwell)}
\end{aligned}$$

where $\Sigma_{i,j}$ is the interface between Ω_i and Ω_j, and where $\mathcal{S}$ is a well chosen operator, classically referred to as the *transmission operator*.

In the previous section, for the particular case of acoustic waves, we saw that the optimal operator is defined, *in the Fourier plane*, as:

$$\lambda(s) = -\alpha(s) = \begin{cases} -\sqrt{s^2 - k^2} & \text{if } |s| \geq k, \\ \jmath\sqrt{k^2 - s^2} & \text{if } |s| \leq k. \end{cases}$$

This function in the Fourier plane corresponds to a non-local operator, usually referred to as the *Dirichlet-to-Neumann* map. Because of its non-locality, this operator is hard to handle. Thus, over the years, localized approximations of this optimal map were proposed, and are reviewed in the following. The same approach can be followed for Maxwell's equations, and an optimal map can be derived [44]. In an electromagnetic context, this optimal operator is usually referred to as the *magnetic-to-electric* map. Again, this operator is non-local, and localized approximations were proposed.

Schwarz algorithms implementing these approximated Dirichlet-to-Neumann or magnetic-to-electric maps are called *optimized* Schwarz methods. In the remainder of this section, popular approximations are proposed. We will first start by the acoustic case, and then conclude by the electromagnetic counterpart. What follows is inspired from [128].

6.3.1 Acoustic case

Després' operator

As already mentioned, the first operator proposed is simply the scalar quantity [32]:

$$\mathcal{S}_{\text{Despres}}(p) = -\jmath k\, p. \tag{6.17}$$

Evanescent modes damping algorithm

This last operator can be further generalized by introducing a real parameter χ [20, 22]:

$$\mathcal{S}_{\text{EMDA}}(p) = (-\jmath k + \chi)(p). \tag{6.18}$$

This operator is called the *evanescent modes damping algorithm*, or EMDA. Indeed, the parameter χ can be chosen to optimize the convergence of the evanescent modes[5].

[5]Let us recall that these modes correspond to the case $|s| > k$, with k the wavenumber and s the value of the Fourier variable.

Optimized second-order transmission condition

As already mentioned, the optimal Dirichlet-to-Neumann map is represented by a square-root function in the Fourier plane[6]. This operator can be approximated by a Taylor expansion, and truncated at the second-order term. Following this strategy, the operator below was proposed [53]:

$$\mathcal{S}_{\mathrm{OO2}}(p) = (a + b\,\mathrm{div}_\Sigma\,\mathbf{grad}_\Sigma)(p), \tag{6.19}$$

where a and b are two complex numbers used to optimize the convergence, and computed by solving a min-max problem on the convergence radius [53]. This operator is called the *optimized second-order*, or OO2.

Padé-localized square-root transmission condition

As done for the optimized second-order case, the Padé-localized square-root transmission condition starts from the optimal Dirichlet-to-Neumann map. However, instead of localizing it by a Taylor expansion, this condition uses a Padé decomposition of order N_p. This strategy leads to the following operator [21]:

$$\mathcal{S}_{\mathrm{Pade}}(p) = -\jmath k C_0\, p - \jmath k \sum_{\ell=1}^{N_p} A_\ell\,\mathrm{div}_\Sigma \left[\frac{1}{k_\varepsilon^2}\,\mathbf{grad}_\Sigma\,\varphi_l\right]\left[\mathcal{I} + B_\ell\,\mathrm{div}_\Sigma\left(\frac{1}{k_\varepsilon^2}\,\mathbf{grad}_\Sigma\right)\right]^{-1}(p), \tag{6.20}$$

with k_ε defined as

$$k_\varepsilon = k + \jmath\varepsilon, \tag{6.21}$$

where $\varepsilon = 0.39\,k^{1/3}\mathcal{H}^{2/3}$, and where $\mathcal{H}$ is the local mean curvature of the interface.

In addition to the k_ε parameter, equation (6.20) make use of the C_0, A_ℓ and B_ℓ Padé coefficients

$$C_0 = e^{\jmath\alpha/2} R_{N_p}\left(e^{-\jmath\alpha} - 1\right), \quad A_\ell = \frac{e^{-\jmath\alpha/2} a_\ell}{\left[1 + b_\ell(e^{-\jmath\alpha} - 1)\right]^2}, \quad B_\ell = \frac{e^{-\jmath\alpha} b_\ell}{1 + b_\ell(e^{-\jmath\alpha} - 1)}, \tag{6.22}$$

where:

- α is a rotation angle in the complex plane, usually taken as $\pi/4$;
- $R_{N_p}(z)$ is the standard real-valued Padé approximation of order N_p of $\sqrt{1+z}$, that is

$$R_{N_p}(z) = 1 + \sum_{\ell=1}^{N_p} \frac{a_\ell z}{1 + b_\ell z};$$

- a_ℓ and b_ℓ are defined as

$$a_\ell = \frac{2}{2N_p + 1}\sin^2\left(\frac{\ell\pi}{2N_p+1}\right), \qquad b_\ell = \cos^2\left(\frac{\ell\pi}{2N_p+1}\right).$$

Finally, we call this operator the *Padé-localized square-root*, or Pade.

[6] More precisely, we showed that this operator is a square-root in a two-dimensional context. The complete three-dimensional derivation can be found in [21]; again, a square-root operator is found.

6.3.2 Electromagnetic case

Després' operator

Let us now consider the transmission operators for Maxwell's equations. As we already know, the most simple operator is the scalar value [33]:

$$\mathcal{S}_{\text{Despres}}\left[\gamma^T(\boldsymbol{e})\right] = +\jmath k\,\gamma^T(\boldsymbol{e}). \tag{6.23}$$

Optimized second-order transmission condition

Equivalently to the acoustic case, we know that the optimal transmission operator is the magnetic-to-electric map[7]. As done for the acoustic case, this optimal operator can be localized using a Taylor expansion. Using this approach, the following operator was proposed [34, 35, 104]:

$$\mathcal{S}_{\text{OO2}}\left[\gamma^T(\boldsymbol{e})\right] = \jmath k\left[\mathcal{I}+\frac{a}{k^2}\,\mathbf{grad}_\Sigma\,\text{div}_\Sigma\right]^{-1}\left[\mathcal{I}-\frac{b}{k^2}\,\mathbf{curl}_\Sigma\,\mathbf{curl}_\Sigma\right]\left[\gamma^T(\boldsymbol{e})\right], \tag{6.24}$$

where a and b are again two complex numbers used to optimize the converge. This time, the min-max problem on the convergence radius is computed to optimize both TE and TM modes [34, 35, 104]. As for the acoustic case, this operator is called the *optimized second-order*, or OO2[8].

Padé-localized square-root transmission condition

Finally, it is also possible to localize the optimal magnetic-to-electric map, by using a Padé expansion instead of a Taylor one [45]:

$$\mathcal{S}_{\text{Pade}}\left[\gamma^T(\boldsymbol{e})\right] = \jmath k\left[C_0+\sum_{\ell=1}^{N_p} A_\ell\mathcal{X}\left(\mathcal{I}+B_\ell\mathcal{X}\right)^{-1}\right]^{-1}\left[\mathcal{I}-\mathbf{curl}_\Sigma\,\frac{1}{k_\varepsilon^2}\,\mathbf{curl}_\Sigma\right]\left[\gamma^T(\boldsymbol{e})\right], \tag{6.25}$$

with $\mathcal{X}$ defined as

$$\mathcal{X} = \mathbf{grad}_\Sigma\,\frac{1}{k_\varepsilon^2}\,\text{div}_\Sigma - \mathbf{curl}_\Sigma\,\frac{1}{k_\varepsilon^2}\,\mathbf{curl}_\Sigma\,.$$

The quantities k_ε, C_0, A_ℓ and B_ℓ are the same as the ones defined in (6.21) and (6.22). Again, as for the acoustic case, we call this operator the *Padé-localized square-root*, or Pade[8].

6.4 Krylov acceleration

Until now, the (optimized) non-overlapping parallel Schwarz method was presented as a fixed point scheme. That is, the iterative process (6.15) (or (6.16)) terminates once the pressure (or electric) field remains unchanged from iteration to iteration. Let us now show, how this fixed

[7] Electromagnetic counterpart of the Dirichlet-to-Neumann map.

[8] Based on the context, it should be clear whether the acoustic or electromagnetic operator is used.

point algorithm can be recast into a linear system. Indeed, this classical technique allows fixed point schemes to benefit from a Krylov acceleration. The expected advantages will then be twofold: a decreased iteration count [21, 53, 108], and convergence even when $\rho = 1$ for certain modes [53, 104, 108].

6.4.1 Acoustic case

For simplicity, we start by considering the acoustic case. In order to recast the fixed point scheme (6.15) into a linear system, let us write the transmission condition (6.15b) on both $\Sigma_{i,j}$ and $\Sigma_{j,i}$:

$$\begin{cases} \mathbf{n}_i \cdot \mathbf{grad}\, p_i^{n+1} + \mathcal{S}\,(p_i^{n+1}) = \mathbf{n}_i \cdot \mathbf{grad}\, p_j^n + \mathcal{S}\,(p_j^n), \\ \qquad\qquad = \underbrace{-\mathbf{n}_j \cdot \mathbf{grad}\, p_j^n + \mathcal{S}\,(p_j^n)}_{g_{i,j}^n}, & \text{on } \Sigma_{i,j}, \quad (6.26a) \\ \mathbf{n}_j \cdot \mathbf{grad}\, p_j^{n+1} + \mathcal{S}\,(p_j^{n+1}) = \mathbf{n}_j \cdot \mathbf{grad}\, p_i^n + \mathcal{S}\,(p_i^n), \\ \qquad\qquad = \underbrace{-\mathbf{n}_i \cdot \mathbf{grad}\, p_i^n + \mathcal{S}\,(p_i^n)}_{g_{j,i}^n} & \text{on } \Sigma_{j,i}, \quad (6.26b) \end{cases}$$

since $\mathbf{n}_i = -\mathbf{n}_j$ when there is no overlap. From the above equations, we can directly notice that two new quantities have been defined:

1. $g_{i,j}^n$, constructed using only data coming from the sub-problem posed on Ω_j;
2. $g_{j,i}^n$, constructed using only data coming from the sub-problem posed on Ω_i.

Furthermore, those two quantities link the information known at step n of the algorithm. Is it then possible to compute these g quantities at step $n+1$? By definition, we have:

$$\begin{cases} g_{i,j}^{n+1} = -\mathbf{n}_j \cdot \mathbf{grad}\, p_j^{n+1} + \mathcal{S}\,(p_j^{n+1}) = -\mathbf{n}_j \cdot \mathbf{grad}\, p_j^{n+1} - \mathcal{S}\,(p_j^{n+1}) + 2\,\mathcal{S}\,(p_j^{n+1}), \\ g_{j,i}^{n+1} = -\mathbf{n}_i \cdot \mathbf{grad}\, p_i^{n+1} + \mathcal{S}\,(p_i^{n+1}) \ = -\mathbf{n}_i \cdot \mathbf{grad}\, p_i^{n+1} - \mathcal{S}\,(p_i^{n+1}) + 2\,\mathcal{S}\,(p_i^{n+1}). \end{cases}$$

Then, by using (6.26a) and (6.26b), the update laws write:

$$\begin{cases} g_{i,j}^{n+1} = -g_{j,i}^n + 2\,\mathcal{S}\,(p_j^{n+1}) & \text{on } \Sigma_{i,j}, \\ g_{j,i}^{n+1} = -g_{i,j}^n + 2\,\mathcal{S}\,(p_i^{n+1}) & \text{on } \Sigma_{j,i}. \end{cases}$$

With these new auxiliary unknowns $g_{i,j}$, and by introducing Dirichlet boundary conditions on Γ_i, the fixed point scheme (6.15) becomes:

$$\begin{cases} (\operatorname{div} \mathbf{grad} + k^2) p_i^{n+1} = 0 & \text{on } \Omega_i, \quad (6.27a) \\ (\mathbf{n}_i \cdot \mathbf{grad} + \mathcal{S})\, p_i^{n+1} = g_{i,j}^n & \text{on } \Sigma_{i,j}, \forall j \in D_i, \quad (6.27b) \\ p_i^{n+1} = f & \text{on } \Gamma_i, \quad (6.27c) \end{cases}$$

with the update law

$$g_{i,j}^{n+1} = -g_{j,i}^n + 2\,\mathcal{S}\,(p_j^{n+1}) \quad \text{on } \Sigma_{i,j}. \quad (6.27d)$$

By the linearity of the problem, the solution p_i^n can be separated into two contributions: $p_i^n = v_i^n + w_i^n$, where v_i^n and w_i^n are the solutions of:

$$\begin{cases} (\operatorname{div}\mathbf{grad} + k^2) v_i^{n+1} = 0 & \text{on } \Omega_i, \quad (6.28a) \\ (\mathbf{n}_i \cdot \mathbf{grad} + \mathcal{S})\, v_i^{n+1} = g_{i,j}^n & \text{on } \Sigma_{i,j}, \forall j \in D_i, \quad (6.28b) \\ v_i^{n+1} = 0 & \text{on } \Gamma_i, \quad (6.28c) \end{cases}$$

and

$$\begin{cases} (\operatorname{div}\mathbf{grad} + k^2) w_i^{n+1} = 0 & \text{on } \Omega_i, \quad (6.29a) \\ (\mathbf{n}_i \cdot \mathbf{grad} + \mathcal{S})\, w_i^{n+1} = 0 & \text{on } \Sigma_{i,j}, \forall j \in D_i, \quad (6.29b) \\ w_i^{n+1} = f & \text{on } \Gamma_i. \quad (6.29c) \end{cases}$$

In other words:

- w_i^n is the solution of each sub-problem *without* coupling, and with the *non-homogeneous* Dirichlet boundary conditions;
- v_i^n is the solution of each sub-problem *with* coupling, and with *homogeneous* Dirichlet boundary conditions.

It is worth noticing that since w_i^n is the solution of an uncoupled sub-problem, it does not evolve between two iterations: thus, $w_i^n = w_i$. Let us now insert the decomposition $p_i^n = v_i^n + w_i$ into the update law (6.27d). It writes then:

$$\begin{aligned} g_{i,j}^{n+1} &= -g_{j,i}^n + 2\,\mathcal{S}\,(v_j^{n+1}) + 2\,\mathcal{S}\,(w_j), \\ &= -g_{j,i}^n + 2\,\mathcal{S}\,(v_j^{n+1}) + b_j \qquad \text{on } \Sigma_{i,j}, \end{aligned} \quad (6.30)$$

where $b_j = 2\,\mathcal{S}\,(w_j)$ is a known quantity, computed by solving (6.29) and by applying $\mathcal{S}$. Furthermore, by looking at (6.28) and (6.30), the fixed point algorithm can be formally written as:

$$\boldsymbol{g}^{n+1} = \mathcal{A}\,(\boldsymbol{g}^n) + \boldsymbol{b}, \quad (6.31)$$

where $\boldsymbol{g}$ (resp. $\boldsymbol{b}$) is the concatenation of every $g_{i,j}$ (resp. b_j), and where an application of the operator $\mathcal{A}$ is one step of the procedure, defined by (6.28) and the following update law:

$$g_{i,j}^{n+1} = -g_{j,i}^n + 2\,\mathcal{S}\,(v_j^{n+1}) \qquad \text{on } \Sigma_{i,j}. \quad (6.32)$$

Finally, we know that once the algorithm has converged, the following is true: $\boldsymbol{g}^{n+1} = \boldsymbol{g}^n$. Therefore, the fixed point algorithm (6.31) can be recast into the following linear system:

$$(\mathcal{I} - \mathcal{A})\boldsymbol{g} = \boldsymbol{b}, \quad (6.33)$$

which can be solved by a Krylov method. At each iteration of the solver, an application of $\mathcal{A}$ is required on some vector generated by the Krylov algorithm. As already mentioned, this application is computed by solving (6.28) on each sub-problem, and by applying the update law (6.32).

Before going any further, let us remark that solving (6.33) will not solve the original problem (6.27). Indeed, the unknown of (6.33) is the $\boldsymbol{g}$ auxiliary quantity, and not the original

pressure field. To recover this physical unknown, it is then necessary to solve:

$$\begin{cases} (\operatorname{div}\mathbf{grad}+k^2)p_i = 0 & \text{on } \Omega_i, \quad (6.34a) \\ (\mathbf{n}_i \cdot \mathbf{grad}+\mathcal{S})\, p_i = g_{i,j} & \text{on } \Sigma_{i,j}, \forall j \in D_i, \quad (6.34b) \\ p_i = f & \text{on } \Gamma_i, \quad (6.34c) \end{cases}$$

where the quantity $g_{i,j}$ is nothing but a component of $\boldsymbol{g}$, the solution of (6.33). With this last step, the Krylov accelerated procedure can be summarized in this way.

1. Compute the right hand side of (6.33) by:
   (a) computing w_j, the solution of (6.29);
   (b) constructing $b_j = 2\,\mathcal{S}\,(w_j)$.
2. Compute $\boldsymbol{g}$ by solving (6.33) with an iterative solver. An application of $\mathcal{A}$ amounts to solve (6.28) and to apply the update law (6.32).
3. Compute p_i by solving (6.34), where the quantity $g_{i,j}$ is an entry in the vector $\boldsymbol{g}$ found previously.

6.4.2 Electromagnetic case

The strategy developed above, for acoustic problems, can be also applied to electromagnetic situations. However, this time, the right hand side of (6.33) is constructed by finding $\boldsymbol{w}_i$ as the solution of:

$$\begin{cases} (\mathbf{curl}\,\mathbf{curl}-k^2)\boldsymbol{w}_i = \mathbf{0} & \text{on } \Omega_i, \quad (6.35a) \\ \gamma_i^t\,(\mathbf{curl}\,\boldsymbol{w}_i) + \mathcal{S}\left[\gamma_i^T\,(\boldsymbol{w}_i)\right] = \mathbf{0} & \text{on } \Sigma_{i,j}, \forall j \in D_i, \quad (6.35b) \\ \gamma_i^T\,(\boldsymbol{w}_i) = \boldsymbol{f} & \text{on } \Gamma_i, \quad (6.35c) \end{cases}$$

and by forming $\boldsymbol{b}_i = 2\,\mathcal{S}\,(\boldsymbol{w}_i)$. Regarding operator $\mathcal{A}$, its application to a vector $\boldsymbol{g}$ amounts to solve the following sub-problems in $\boldsymbol{v}_i^{n+1}$:

$$\begin{cases} (\mathbf{curl}\,\mathbf{curl}-k^2)\boldsymbol{v}_i^{n+1} = \mathbf{0} & \text{on } \Omega_i, \quad (6.36a) \\ \gamma_i^t\left(\mathbf{curl}\,\boldsymbol{v}_i^{n+1}\right) + \mathcal{S}\left[\gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right)\right] = \boldsymbol{g}_{i,j}^n & \text{on } \Sigma_{i,j}, \forall j \in D_i, \quad (6.36b) \\ \gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right) = \mathbf{0} & \text{on } \Gamma_i, \quad (6.36c) \end{cases}$$

with the update law

$$\boldsymbol{g}_{i,j}^{n+1} = -\boldsymbol{g}_{j,i}^n + 2\,\mathcal{S}\,(\boldsymbol{v}_j^{n+1}) \qquad \text{on } \Sigma_{i,j}. \quad (6.37)$$

Finally, the original electric field is recovered by solving:

$$\begin{cases} (\mathbf{curl}\,\mathbf{curl}-k^2)\boldsymbol{e}_i = \mathbf{0} & \text{on } \Omega_i, \quad (6.38a) \\ \gamma_i^t\,(\mathbf{curl}\,\boldsymbol{e}_i) + \mathcal{S}\left[\gamma_i^T\,(\boldsymbol{e}_i)\right] = \boldsymbol{g}_{i,j} & \text{on } \Sigma_{i,j}, \forall j \in D_i, \quad (6.38b) \\ \gamma_i^T\,(\boldsymbol{e}_i) = \boldsymbol{f} & \text{on } \Gamma_i, \quad (6.38c) \end{cases}$$

where the $\boldsymbol{g}_{i,j}$ are sub-vectors of $\boldsymbol{g}$, the solution of (6.33).

6.5 Finite element discretization

With all the above tools, we have every thing we need to solve our acoustic, or electromagnetic, time-harmonic wave problems *at a continuous* level. In order to handle complex geometries and non-homogeneous media, a finite element discretization is needed, and is the subject of this section. For the sake of clarity, only the equations (6.28) and (6.32) (or (6.36) and (6.37) for electromagnetic problems) will be treated. The treatment of (6.29) and (6.34) (or (6.35) and (6.38) for electromagnetic problems) is straightforward. Again, the acoustic case will be treated first, and will be followed by its electromagnetic counterpart.

6.5.1 Acoustic case

By applying the strategy developed in section 2.1.3, the weak formulation of an acoustic wave problem (6.28) writes:

$$\begin{cases} \text{Find } v_i^{n+1} \text{ in } H_0^1(\Omega_i) \text{ such that, for every } v_i' \in H_0^1(\Omega_i): \\ \displaystyle\int_{\Omega_i} \mathbf{grad}\, v_i^{n+1} \cdot \mathbf{grad}\, v_i' \,\mathrm{d}\Omega_i - k^2 \int_{\Omega_i} v_i^{n+1}\, v_i' \,\mathrm{d}\Omega_i + \sum_{j\in D_i} \int_{\Sigma_{i,j}} \mathcal{S}\left(v_i^{n+1}\right) v_i' \,\mathrm{d}\Sigma_{i,j} \\ \qquad\qquad = \displaystyle\sum_{j\in D_i} \int_{\Sigma_{i,j}} g_{i,j}^n\, v_i' \,\mathrm{d}\Sigma_{i,j}, \end{cases} \tag{6.39}$$

for all $i \in \{0, \ldots, N-1\}$, N being the number of sub-domains. Let us remark that the normal derivative term in (2.12), has been computed by exploiting the transmission condition (6.28b). For the update law (6.32), we have the following weak formulation:

$$\begin{cases} \text{Find } g_{j,i}^{n+1} \text{ in } H^1(\Sigma_{i,j}) \text{ such that, for every } g_{j,i}' \in H^1(\Sigma_{i,j}): \\ \displaystyle\int_{\Sigma i,j} g_{j,i}^{n+1}\, g_{j,i}' \,\mathrm{d}\Sigma_{i,j} = -\int_{\Sigma_{i,j}} g_{i,j}^n\, g_{j,i}' \,\mathrm{d}\Sigma_{i,j} + 2\int_{\Sigma_{i,j}} \mathcal{S}\left(v_i^{n+1}\right) g_{j,i}' \,\mathrm{d}\Sigma_{i,j}. \end{cases} \tag{6.40}$$

In the weak formulations (6.39) and (6.40), the following quantities have to be computed: $\displaystyle\int_{\Sigma_{i,j}} \mathcal{S}\left(v_i^{n+1}\right) v_i' \,\mathrm{d}\Sigma_{i,j}$ and $\displaystyle\int_{\Sigma_{i,j}} \mathcal{S}\left(v_i^{n+1}\right) g_{j,i}' \,\mathrm{d}\Sigma_{i,j}$. Depending on the transmission operator used, these integrals expand as follows.

- Després' operator:

$$\int_{\Sigma_{i,j}} \mathcal{S}\left(v_i^{n+1}\right) v_i' \,\mathrm{d}\Sigma_{i,j} = \int_{\Sigma_{i,j}} -\jmath k\, v_i^{n+1}\, v_i' \,\mathrm{d}\Sigma_{i,j}, \tag{6.41}$$

$$\int_{\Sigma_{i,j}} \mathcal{S}\left(v_i^{n+1}\right) g_{j,i}' \,\mathrm{d}\Sigma_{i,j} = \int_{\Sigma_{i,j}} -\jmath k\, v_i^{n+1}\, g_{j,i}' \,\mathrm{d}\Sigma_{i,j}. \tag{6.42}$$

- Evanescent modes damping algorithm:

$$\int_{\Sigma_{i,j}} \mathcal{S}\left(v_i^{n+1}\right) v_i' \,\mathrm{d}\Sigma_{i,j} = \int_{\Sigma_{i,j}} (-\jmath k + \chi) v_i^{n+1}\, v_i' \,\mathrm{d}\Sigma_{i,j}, \tag{6.43}$$

$$\int_{\Sigma_{i,j}} \mathcal{S}\left(v_i^{n+1}\right) g_{j,i}' \,\mathrm{d}\Sigma_{i,j} = \int_{\Sigma_{i,j}} (-\jmath k + \chi) v_i^{n+1}\, g_{j,i}' \,\mathrm{d}\Sigma_{i,j}. \tag{6.44}$$

- Optimized second-order transmission condition:

$$\int_{\Sigma_{i,j}} \mathcal{S}\left(v_i^{n+1}\right) v_i' \,\mathrm{d}\Sigma_{i,j} = \int_{\Sigma_{i,j}} a\, v_i^{n+1}\, v_i' \,\mathrm{d}\Sigma_{i,j} - \int_{\Sigma_{i,j}} b\, \mathbf{grad}\, v_i^{n+1} \cdot \mathbf{grad}\, v_i' \,\mathrm{d}\Sigma_{i,j}, \tag{6.45}$$

$$\int_{\Sigma_{i,j}} \mathcal{S}\left(v_i^{n+1}\right) g_{j,i}' \,\mathrm{d}\Sigma_{i,j} = \int_{\Sigma_{i,j}} a\, v_i^{n+1}\, g_{j,i}' \,\mathrm{d}\Sigma_{i,j} - \int_{\Sigma_{i,j}} b\, \mathbf{grad}\, v_i^{n+1} \cdot \mathbf{grad}\, g_{j,i}' \,\mathrm{d}\Sigma_{i,j}. \tag{6.46}$$

- Padé-localized square-root transmission condition:

$$\int_{\Sigma_{i,j}} \mathcal{S}\left(v_i^{n+1}\right) v_i' \,\mathrm{d}\Sigma_{i,j} = -\jmath k\, C_0 \int_{\Sigma_{i,j}} v_i^{n+1}\, v_i' \,\mathrm{d}\Sigma_{i,j} + \jmath k \sum_{\ell=1}^{Np} A_\ell \int_{\Sigma_{i,j}} \frac{1}{k_\varepsilon^2}\, \mathbf{grad}_{\Sigma_{i,j}}\, \varphi_\ell \cdot \mathbf{grad}_{\Sigma_{i,j}}\, v_i' \,\mathrm{d}\Sigma_{i,j}, \tag{6.47}$$

where, for every $\ell = 1, \ldots, N_p$, the function φ_ℓ is obtained through the resolution of

$$\begin{cases} \text{Find } \varphi_\ell \text{ in } H^1(\Sigma_{i,j}) \text{ such that, for every } \varphi_\ell' \in H^1(\Sigma_{i,j}): \\ -\displaystyle\int_{\Sigma_{i,j}} v_i^{n+1}\, \varphi_\ell' \,\mathrm{d}\Sigma_{i,j} - B_\ell \int_{\Sigma_{i,j}} \frac{1}{k_\varepsilon^2}\, \mathbf{grad}_{\Sigma_{i,j}}\, \varphi_\ell \cdot \mathbf{grad}_{\Sigma_{i,j}}\, \varphi_\ell' \,\mathrm{d}\Sigma_{i,j} \\ \qquad\qquad + \displaystyle\int_{\Sigma_{i,j}} \varphi_\ell \cdot \varphi_\ell' \,\mathrm{d}\Sigma_{i,j} = 0. \end{cases} \tag{6.48}$$

Furthermore, we have:

$$\int_{\Sigma_{i,j}} \mathcal{S}\left(v_i^{n+1}\right) g_{j,i}' \,\mathrm{d}\Sigma_{i,j} = -\jmath k\, C_0 \int_{\Sigma_{i,j}} v_i^{n+1}\, g_{j,i}' \,\mathrm{d}\Sigma_{i,j} - \jmath k \sum_{\ell=1}^{N_p} \frac{A_\ell}{B_\ell} \int_{\Sigma_{i,j}} \left(v_i^{n+1} - \varphi_\ell\right) g_{j,i}' \,\mathrm{d}\Sigma_{i,j}. \tag{6.49}$$

6.5.2 Electromagnetic case

Let us now consider the electromagnetic case. Again, by applying the strategy developed in section 2.1.2, the weak formulation of an electromagnetic wave problem (6.36) writes:

$$\begin{cases} \text{Find } \boldsymbol{v}_i^{n+1} \in H_0(\mathbf{curl}, \Omega_i) \text{ such that, for every } \boldsymbol{v}_i' \in H_0(\mathbf{curl}, \Omega_i): \\ \displaystyle\int_{\Omega_i} \mathbf{curl}\, \boldsymbol{v}_i^{n+1} \cdot \mathbf{curl}\, \boldsymbol{v}_i' \,\mathrm{d}\Omega_i - k^2 \int_{\Omega_i} \boldsymbol{v}_i^{n+1} \cdot \boldsymbol{v}_i' \,\mathrm{d}\Omega_i \\ \qquad - \displaystyle\sum_{j \in D_i} \int_{\Sigma_{i,j}} \mathcal{S}\left[\gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right)\right] \cdot \boldsymbol{v}_i' \,\mathrm{d}\Sigma_{i,j} = -\sum_{j \in D_i} \int_{\Sigma_{i,j}} \boldsymbol{g}_{i,j}^n \cdot \boldsymbol{v}_i' \,\mathrm{d}\Sigma_{i,j}, \end{cases} \tag{6.50}$$

for all $i \in \{0, \ldots, N-1\}$, N being the number of sub-domains. Let us remark that the boundary term in (2.11), has been computed by exploiting the transmission condition (6.36b).

For the update law (6.37), we have the following weak formulation:

$$\begin{cases} \text{Find } \boldsymbol{g}_{j,i}^{n+1} \text{ in } H(\mathbf{curl}, \Sigma_{i,j}) \text{ such that, for every } \boldsymbol{g}'_{j,i} \in \mathbf{H}(\mathbf{curl}, \Sigma_{i,j}): \\ \displaystyle\int_{\Sigma_{i,j}} \boldsymbol{g}_{j,i}^{n+1} \cdot \boldsymbol{g}'_{j,i} \,\mathrm{d}\Sigma_{i,j} = -\int_{\Sigma_{i,j}} \boldsymbol{g}_{i,j}^{n} \cdot \boldsymbol{g}'_{j,i} \,\mathrm{d}\Sigma_{i,j} + 2\int_{\Sigma_{i,j}} \mathcal{S}\left[\gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right)\right] \cdot \boldsymbol{g}'_{j,i} \,\mathrm{d}\Sigma_{i,j}. \end{cases} \tag{6.51}$$

In the weak formulations (6.50) and (6.51), the following quantities have to be computed: $\displaystyle\int_{\Sigma_{i,j}} \mathcal{S}\left[\gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right)\right] \cdot \boldsymbol{v}'_i \,\mathrm{d}\Sigma_{i,j}$ and $\displaystyle\int_{\Sigma_{i,j}} \mathcal{S}\left[\gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right)\right] \cdot \boldsymbol{g}'_{j,i} \,\mathrm{d}\Sigma_{i,j}$. Depending on the transmission operator used, these integrals expand as follows.

- Després' operator:

$$\int_{\Sigma_{i,j}} \mathcal{S}\left[\gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right)\right] \cdot \boldsymbol{v}'_i \,\mathrm{d}\Sigma_{i,j} = \int_{\Sigma_{i,j}} \jmath k\, \gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right) \cdot \boldsymbol{v}'_i \,\mathrm{d}\Sigma_{i,j}, \tag{6.52}$$

$$\int_{\Sigma_{i,j}} \mathcal{S}\left[\gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right)\right] \cdot \boldsymbol{g}'_{j,i} \,\mathrm{d}\Sigma_{i,j} = \int_{\Sigma_{i,j}} \jmath k\, \gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right) \cdot \boldsymbol{g}'_{j,i} \,\mathrm{d}\Sigma_{i,j}. \tag{6.53}$$

- Optimized second-order transmission condition:

$$\int_{\Sigma_{i,j}} \mathcal{S}\left[\gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right)\right] \cdot \boldsymbol{v}'_i \,\mathrm{d}\Sigma_{i,j} = \int_{\Sigma_{i,j}} \jmath k\, \boldsymbol{r} \cdot \boldsymbol{v}'_i \,\mathrm{d}\Sigma_{i,j}, \tag{6.54}$$

where the function $\boldsymbol{r} \in H(\mathbf{curl}, \Sigma_{i,j})$ is obtained through the solution of

$$\begin{cases} \text{Find } \boldsymbol{r} \text{ in } H(\mathbf{curl}, \Sigma_{i,j}) \text{ and } \rho \text{ in } H^1(\Sigma_{i,j}) \text{ such that} \\ \forall \boldsymbol{r}' \in H(\mathbf{curl}, \Sigma_{i,j}) \text{ and } \forall \rho' \in H^1(\Sigma_{i,j}): \\ \\ \displaystyle -\int_{\Sigma_{i,j}} \frac{a}{k^2}\, \mathbf{grad}_{\Sigma_{i,j}} \rho \cdot \boldsymbol{r}' \,\mathrm{d}\Sigma_{i,j} - \int_{\Sigma_{i,j}} \boldsymbol{r} \cdot \boldsymbol{r}' \,\mathrm{d}\Sigma_{i,j} + \int_{\Sigma_{i,j}} \gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right) \cdot \boldsymbol{r}' \,\mathrm{d}\Sigma_{i,j} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad - \displaystyle\int_{\Sigma_{i,j}} \frac{b}{k^2}\, \mathbf{curl}_{\Sigma_{i,j}} \left[\gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right)\right] \cdot \mathbf{curl}_{\Sigma_{i,j}} \boldsymbol{r}' \,\mathrm{d}\Sigma_{i,j} = 0, \quad (6.55a) \\ \\ \displaystyle\int_{\Sigma_{i,j}} \rho\, \rho' \,\mathrm{d}\Sigma_{i,j} + \int_{\Sigma_{i,j}} \boldsymbol{r} \cdot \mathbf{grad}_{\Sigma_{i,j}} \rho' \,\mathrm{d}\Sigma_{i,j} = 0. \quad (6.55b) \end{cases}$$

Furthermore, we have:

$$\int_{\Sigma_{i,j}} \mathcal{S}\left[\gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right)\right] \cdot \boldsymbol{g}'_{j,i} \,\mathrm{d}\Sigma_{i,j} = \int_{\Sigma_{i,j}} \jmath k\, \boldsymbol{r} \cdot \boldsymbol{g}'_{j,i} \,\mathrm{d}\Sigma_{i,j}. \tag{6.56}$$

- Padé-localized square-root transmission condition:

$$\int_{\Sigma_{i,j}} \mathcal{S}\left[\gamma_i^T\left(\boldsymbol{v}_i^{n+1}\right)\right] \cdot \boldsymbol{v}'_i \,\mathrm{d}\Sigma_{i,j} = \int_{\Sigma_{i,j}} \jmath k\, \boldsymbol{r} \cdot \boldsymbol{v}'_i \,\mathrm{d}\Sigma_{i,j}, \tag{6.57}$$

where the function $\boldsymbol{r} \in H(\mathbf{curl}, \Sigma_{i,j})$ is obtained through the solution of

$$
\begin{cases}
\text{Find } \boldsymbol{r} \text{ in } H(\mathbf{curl}, \Sigma_{i,j}), \\
\text{with } \ell = 1, \ldots, N_p, \, \boldsymbol{\varphi}_\ell \text{ in } H(\mathbf{curl}, \Sigma_{i,j}) \text{ and } \rho_\ell \text{ in } H^1(\Sigma_{i,j}), \\
\text{such that } \forall \boldsymbol{r}' \in H(\mathbf{curl}, \Sigma_{i,j}), \forall \boldsymbol{\varphi}'_\ell \in H(\mathbf{curl}, \Sigma_{i,j}) \text{ and } \forall \rho'_\ell \in H^1(\Sigma_{i,j}): \\
\\
\displaystyle\int_{\Sigma_{i,j}} C_0 \, \boldsymbol{r} \cdot \boldsymbol{r}' \, \mathrm{d}\Sigma_{i,j} - \int_{\Sigma_{i,j}} \gamma_i^T \left(\boldsymbol{v}_i^{n+1} \right) \cdot \boldsymbol{r}' \, \mathrm{d}\Sigma_{i,j} \\
\qquad + \displaystyle\int_{\Sigma_{i,j}} k_\varepsilon^{-2} \, \mathbf{curl}_{\Sigma_{i,j}} \left[\gamma_i^T \left(\boldsymbol{v}_i^{n+1} \right) \right] \cdot \mathbf{curl}_{\Sigma_{i,j}} \boldsymbol{r}' \, \mathrm{d}\Sigma_{i,j} \\
\qquad + \displaystyle\sum_{\ell=1}^{N_p} A_\ell \left[\int_{\Sigma_{i,j}} \mathbf{grad}_{\Sigma_{i,j}} \rho_\ell \cdot \boldsymbol{r}' \, \mathrm{d}\Sigma_{i,j} \right. \\
\qquad\qquad \left. - \displaystyle\int_{\Sigma_{i,j}} k_\varepsilon^{-2} \, \mathbf{curl}_{\Sigma_{i,j}} \boldsymbol{\varphi}_\ell \cdot \mathbf{curl}_{\Sigma_{i,j}} \boldsymbol{r}' \, \mathrm{d}\Sigma_{i,j} \right] = 0, \quad \text{(6.58a)} \\
\\
\displaystyle\int_{\Sigma_{i,j}} \boldsymbol{\varphi}_\ell \cdot \boldsymbol{\varphi}'_\ell \, \mathrm{d}\Sigma_{i,j} + B_\ell \left[\int_{\Sigma_{i,j}} \mathbf{grad}_{\Sigma_{i,j}} \rho_\ell \cdot \boldsymbol{\varphi}'_\ell \, \mathrm{d}\Sigma_{i,j} \right. \\
\qquad\qquad \left. - \displaystyle\int_{\Sigma_{i,j}} k_\varepsilon^{-2} \, \mathbf{curl}_{\Sigma_{i,j}} \boldsymbol{\varphi}_\ell \cdot \mathbf{curl}_{\Sigma_{i,j}} \boldsymbol{\varphi}'_\ell \, \mathrm{d}\Sigma_{i,j} \right] \\
\qquad\qquad - \displaystyle\int_{\Sigma_{i,j}} \boldsymbol{r} \cdot \boldsymbol{\varphi}'_\ell \, \mathrm{d}\Sigma_{i,j} = 0 \qquad \ell = 1, \ldots, N_p, \quad \text{(6.58b)} \\
\\
\displaystyle\int_{\Sigma_{i,j}} \rho_\ell \, \rho'_\ell \, \mathrm{d}\Sigma_{i,j} + \int_{\Sigma_{i,j}} k_\varepsilon^{-2} \boldsymbol{\varphi}_\ell \cdot \mathbf{grad}_{\Sigma_{i,j}} \rho'_\ell \, \mathrm{d}\Sigma_{i,j} = 0 \qquad \ell = 1, \ldots, N_p. \quad \text{(6.58c)}
\end{cases}
$$

Furthermore, we have:

$$
\int_{\Sigma_{i,j}} \mathcal{S} \left[\gamma_i^T \left(\boldsymbol{v}_i^{n+1} \right) \right] \cdot \boldsymbol{g}'_{j,i} \, \mathrm{d}\Sigma_{i,j} = \int_{\Sigma_{i,j}} \jmath k \, \boldsymbol{r} \cdot \boldsymbol{g}'_{j,i} \, \mathrm{d}\Sigma_{i,j}. \tag{6.59}
$$

6.6 Performance analysis: first-order case

In this section, the performance of the previously presented operators is studied on two simple cases: the propagation of a wave through a three-dimensional rectangular metallic waveguide, and the two-dimensional scattering of a plane wave by a cylinder. Let us note, that only first-order discretizations are considered.

6.6.1 Propagation through a rectangular metallic waveguide

The guide is excited at $k = 25\,[\mathrm{rad/m}]$, it is 2λ long (λ being the wavelength), and has a cross section of $\lambda \times \lambda$. The computation domain is divided into two sub-domains (of 1λ long

each), and meshed with 20 tetrahedra per wavelength. For the acoustics analysis, a first-order FE basis is used, and the first mode is excited. On the other hand, a Nédélec [100][9] FE basis is used for the electromagnetic test cases, and both $TE_{1,1}$ and $TM_{1,1}$ modes are tested. Depending on the considered case, the waveguide is terminated by a Sommerfeld (1.19) or a Silver-Müller (1.20) condition. The linear system (6.33) is solved by an non-restarted GMRES (provided by the PETSc library) with a relative tolerance set to 10^{-6}, and the sub-problems are solved with the MUMPS direct solver.

To conclude the description of the benchmark, let us note that the parameters of the transmission conditions were chosen as follow.

1. For the evanescent modes damping algorithm, the optimal parameter χ was experimentally found at $\chi = 0.25\,k$.
2. For the optimized second-order transmission conditions, the parameters a and b were chosen according to [53] (acoustics) and [104] (electromagnetism).
3. For the Padé-localized square-root transmission condition, the optimal parameter ε is simply 0 (for both scalar and vector cases), since the boundary between the subdomains has no curvature [21, 45]. Moreover, Padé expansions with 4 and 8 terms will be considered.

Convergence speed

Now that our experimental setup is described, let us start by the simplest analysis: the convergence of the GMRES. For the acoustic case, Figure 6.4 depicts the residual history of the iterative solver. From this figure, we can directly notice that the optimized transmission operators are improving significantly the convergence rate of the GMRES, compared to the classical Després' operator. The evanescent modes damping algorithm and the Padé-localized square-root transmission condition, with 4 Padé terms, exhibit similar behaviors. Finally, the best performance is obtained with the second-order transmission condition and the Padé-localized square-root transmission condition, with 8 Padé terms.

Let us now analyze the performance on an electromagnetic case. The solver residual history is available in Figure 6.5 for TE modes, and in Figure 6.6 for TM modes. From those data, we can first notice that the transmission conditions behave similarly with TE or TM excitations. Moreover, as for the scalar case, the optimized operators improve significantly the iteration count of the solver. This time, best performance was obtained with the Padé-localized square-root transmission condition. Using 4 or 8 Padé terms does not change notably the convergence speed.

[9] The Nédélec basis function is a classical incomplete first-order basis, widely used in electromagnetic simulation.

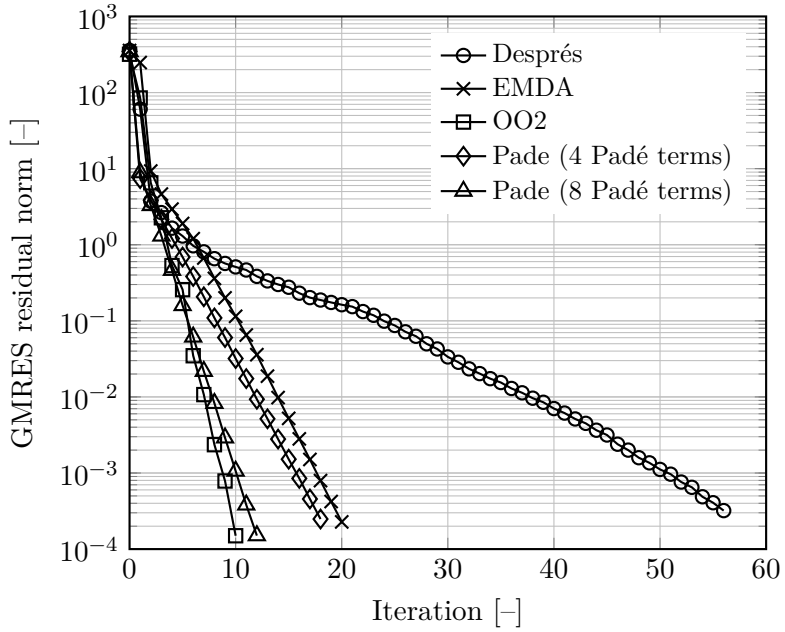


Figure 6.4: GMRES residual history for different scalar transmission conditions, waveguide test case.

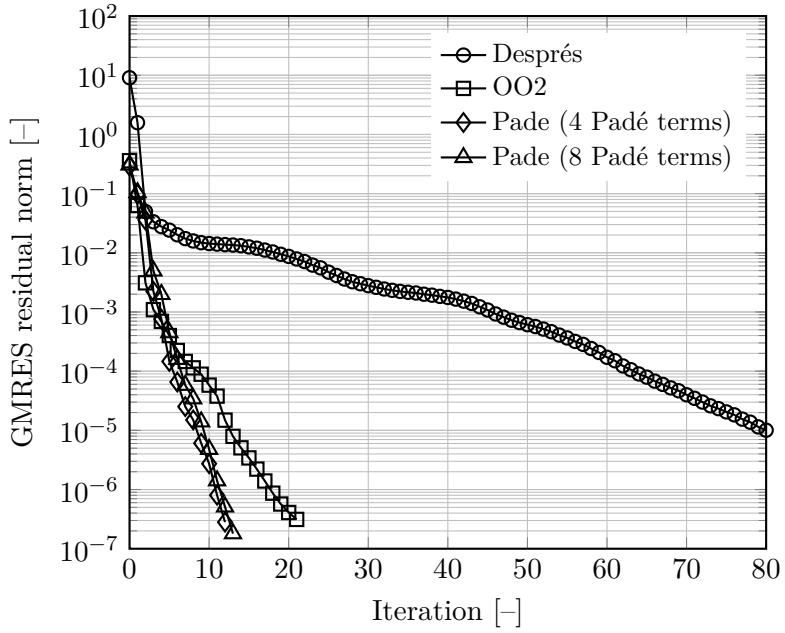


Figure 6.5: GMRES residual history for different vector transmission conditions, waveguide test case: TE mode.

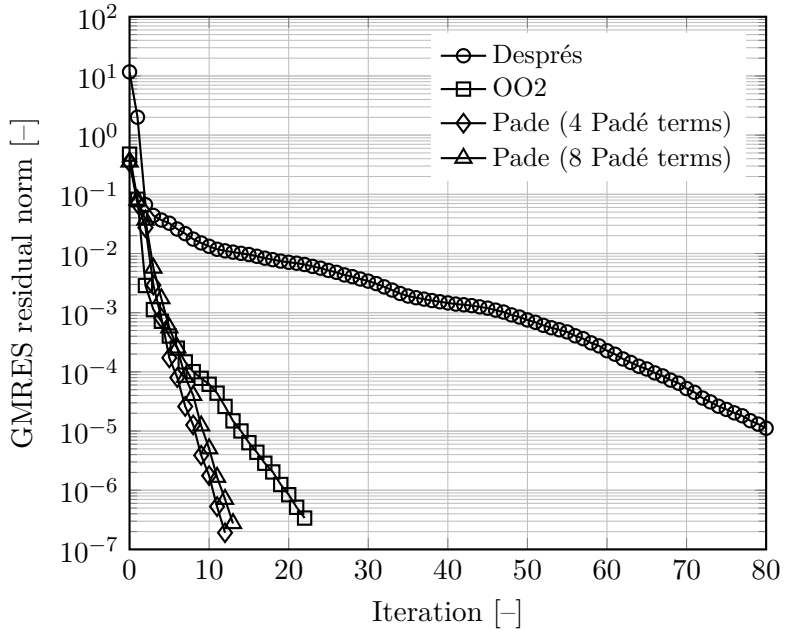


Figure 6.6: GMRES residual history for different vector transmission conditions, waveguide test case: TM mode.

Computation time

With the previous analysis, we assessed the performance on the basis of the GMRES residual history. Let us now consider a more pragmatic criterion: the total computation time. In the following we recorded the computation time for every transmission condition. This timing takes into account:

1. the computation of the right hand side of (6.33),
2. the resolution of (6.33),
3. the final solve (6.34) (acoustics) or (6.38) (electromagnetism).

The obtained results are presented in Table 6.1 for acoustic tests, and in Table 6.2 for electromagnetic cases.

Després	EMDA	OO2	Pade (4 Padé terms)	Pade (8 Padé terms)
18[s]	12[s]	12[s]	16[s]	17[s]

Table 6.1: Computation time for different scalar transmission conditions, waveguide test case.

Després	OO2	Pade (4 Padé terms)	Pade (8 Padé terms)
101[s]	68[s]	72[s]	82[s]

Table 6.2: Computation time for different vector transmission conditions, waveguide test case.

From those tables, we can directly notice that the optimized second-order transmission conditions lead to the faster computation time in both acoustic and electromagnetic test cases. Based in Figures 6.5 and 6.6, we may wonder why the Padé-localized square-root transmission condition is not the best in the electromagnetic tests. Actually, as shown in equation (6.58), in order to discretize the operator, N_p[10] auxiliary systems must be solved, thus increasing the cost of one iteration. Therefore, even if this condition leads to a faster convergence of the GMRES, it does not necessarily lead to a faster total computation time.

6.6.2 Scattering by a cylinder

Based on the last results, one may erroneously conclude that the optimized second-order is the best transmission condition in any circumstances. By analyzing deeper the construction of the Padé-localized square-root transmission condition [21, 45], we may actually find that this operator has been designed, to improve the iterative solver convergence in the case of:

- *curved* boundaries between sub-domains;
- high-frequency regimes.

In this situation, the Pade operator may lead to a faster computation than the OO2 condition, even at a higher iteration cost.

[10] N_p being the number of Padé terms.

To illustrate this, let us now consider the case of the scattering of a plane wave by a cylinder. The plane wave has a wavenumber of $k = 50$[rad/m], and it illuminates a cylinder with a radius of 4λ (λ being the wavelength). The infinite domain is truncated by a concentric cylinder with a radius of 8λ. On its outer boundary, a Sommerfeld (1.19) or Silver-Müller (1.20) condition is imposed. The computational domain is meshed by 15 triangles per wavelength, and is divided in two sub-domains by another concentric cylinder.

Since high-frequency is required, we opted for a two-dimensional simulation, which demands less computational resources than its three-dimensional counterpart. Moreover, since only two sub-domains are considered, it is not possible to distribute the computations across many computing nodes, thus the necessity of the two-dimensionality.

The same solvers as for the waveguide case are used. The transmission conditions optimized parameters are taken from [21, 45] for the OO2 and Pade operators. For the EMDA operator, the parameter χ was experimentally found at $\chi = 0.5\,k$. The number of Padé terms is set to 4.

Convergence speed

As done previously for the waveguide case, let us first analyze the convergence of the GMRES. Results for the acoustic case are reported in Figure 6.7, and in Figure 6.8 for the electromagnetic counterpart. Based on these results, we can directly notice that the Pade operator exhibits the best performance in both cases.

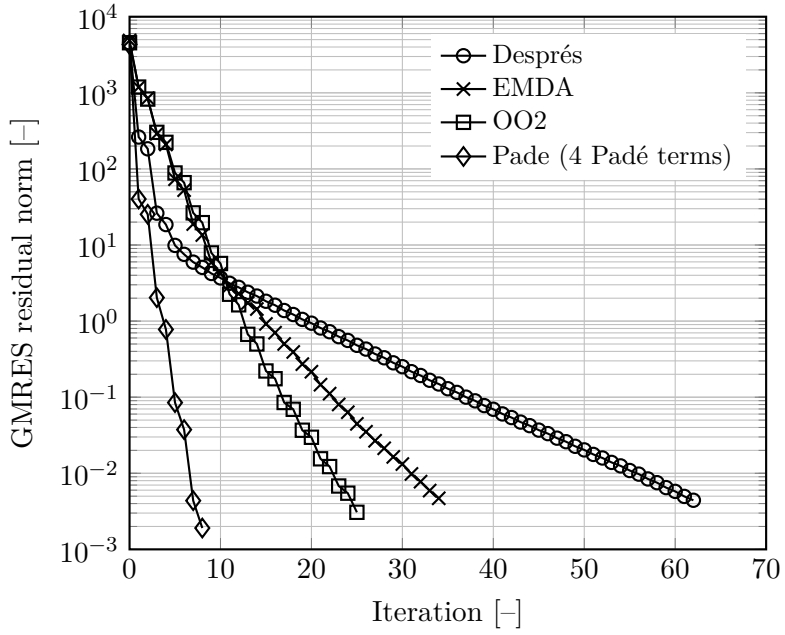


Figure 6.7: GMRES residual history for different scalar transmission conditions, scattering by a circle.

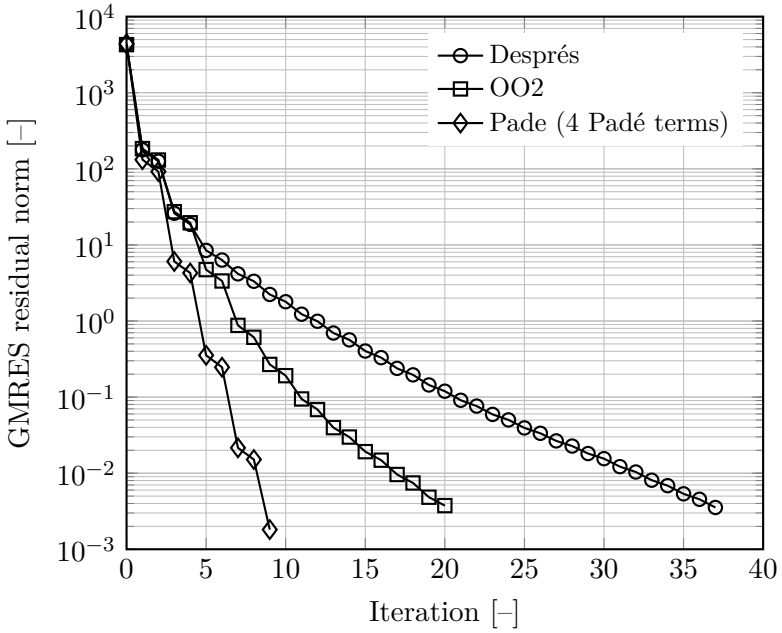


Figure 6.8: GMRES residual history for different vector transmission conditions, scattering by a circle.

Computation time

Let us now consider the total computation time, as reported in Table 6.3. This time, even if the Pade operator has the highest iteration cost, it leads the shortest computation time.

Acoustic				Electromagnetism		
Després	EMDA	OO2	Pade	Després	OO2	Pade
31[s]	22[s]	21[s]	19[s]	67[s]	55[s]	51[s]

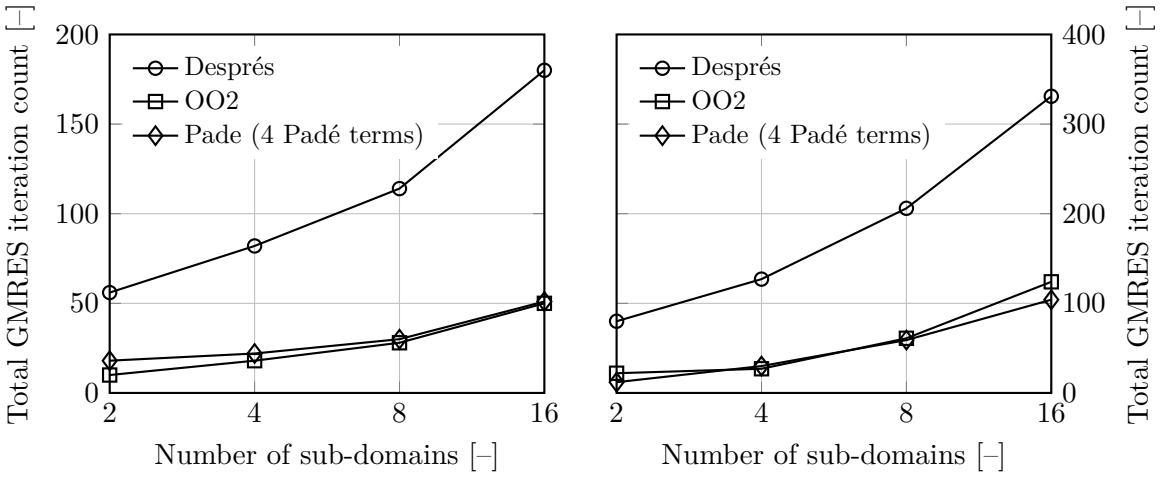
Table 6.3: Computation time for different transmission conditions, scattering by a circle.

6.7 Preconditioners and number of sub-domains

Thanks to its parallel nature, the presented domain decomposition method is appropriate for the treatment of large-scale simulations. Moreover, this approach does not suffer from the memory scaling issue of direct methods, as shown in chapter 5. Indeed, this time, the **LU** factorizations are performed on *independent* systems of *smaller* size (compared to the original problem). One may wonder if the optimized Schwarz algorithm is the ultimate tool for handling large-scale time-harmonic wave problems. Unfortunately not as-is, since this approach suffers from the following drawback.

It can be shown that the number of iterations needed to solve (6.33), increases with the number of sub-domains [130], as shown in Figure 6.9. These data were obtained by solving

the waveguide test case of the previous section.



(a) Acoustic case.

(b) Electromagnetic case.

Figure 6.9: Evolution of the optimized non-overlapping Schwarz method iteration count, with respect to a sub-domain number increase.

Intuitively, this phenomenon is easy to understand. Even if the optimal Dirichlet-to-Neumann (or magnetic-to-electric) transmission operator is used, it takes $N-1$ (N being the number of sub-domains) iterations for a source located a the input of the waveguide to reach the other end; and it takes another $N-1$ steps for this information to return at the input of the waveguide.

This problem can be addressed thanks to preconditioners for the linear system (6.33) [129, 130]; or by using the so-called coarse grids [28, 73][11], which enable a fast information propagation between the sub-problems. This topic is beyond the scope of this thesis.

6.8 Performance analysis: high-order case

In this section, we extend the analysis carried out in section 6.6 to higher-order discretizations. More precisely, the behavior of the Schwarz algorithm is studied for different tetrahedra densities and finite element discretizations. This analysis is carried out through numerical experiments on the waveguide of section 6.6.1. For simplicity, only the electromagnetic $\mathrm{TM}_{1,1}$ case is considered. Moreover, since highly accurate simulations will be carried out, the iterative solver relative tolerance is set to 10^{-9}.

6.8.1 Solution accuracy

Before analyzing the convergence rate of the DDM, let us study how the L_2 error, between a simulation and the analytical solution [119], evolves. Since the domain decomposition method is equivalent to the original problem, we expect to recover the optimal rate of $\mathcal{O}\left(h^{p+1}\right)$, where

[11] In an overlapping Schwarz context.

p is the polynomial order used, and h the mesh elements size (see section 1.5.1 and [69]). By varying the FE discretization order from 1 to 4, and the mesh density from 1 tetrahedron to 32 tetrahedra per wavelength, we obtain the relative errors of Figure 6.10. The measured convergence rates are reported in Table 6.4. Let us note that these last results are obtained by considering, for each FE order, the two finest meshes available.

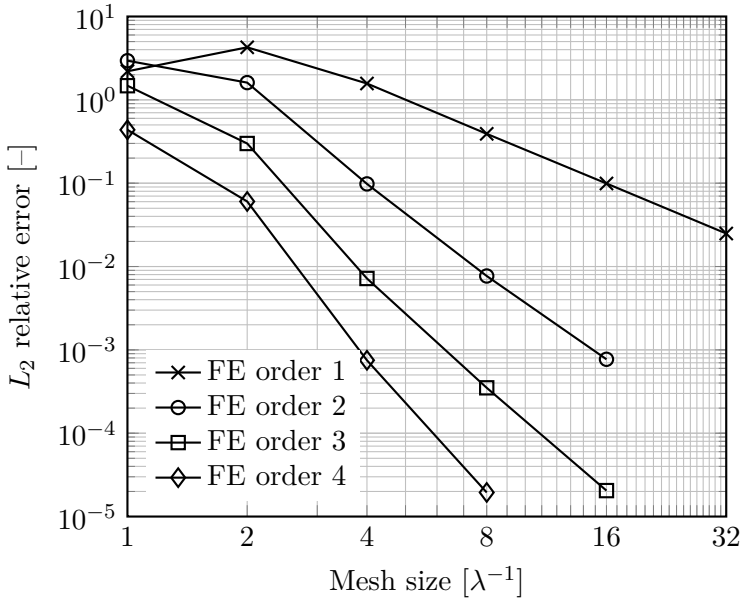


Figure 6.10: Relative errors between analytic and FE solution for different discretization orders and mesh sizes.

FE order	Measured convergence rate	Optimal convergence rate
1	2.0	2
2	3.3	3
3	4.1	4
4	5.3	5

Table 6.4: Convergence rates measured in Figure 6.10 for the finest meshes available.

Based on these data, we may conclude that the DDM solution accuracy evolves as expected. Furthermore, it is worth mentioning that the same results are obtained for every transmission operator. Again, this is expected, since the DDM is equivalent to the original problem.

6.8.2 Iteration count

Let us now focus on the iteration count of the Schwarz algorithm. Figure 6.11 reports the total iteration count required by the DDM iterative solver for different transmission operators. To asses the performance of higher-order FE discretizations with respect to the lower ones, we must compare the accuracy level given in Figure 6.10, and the corresponding iteration

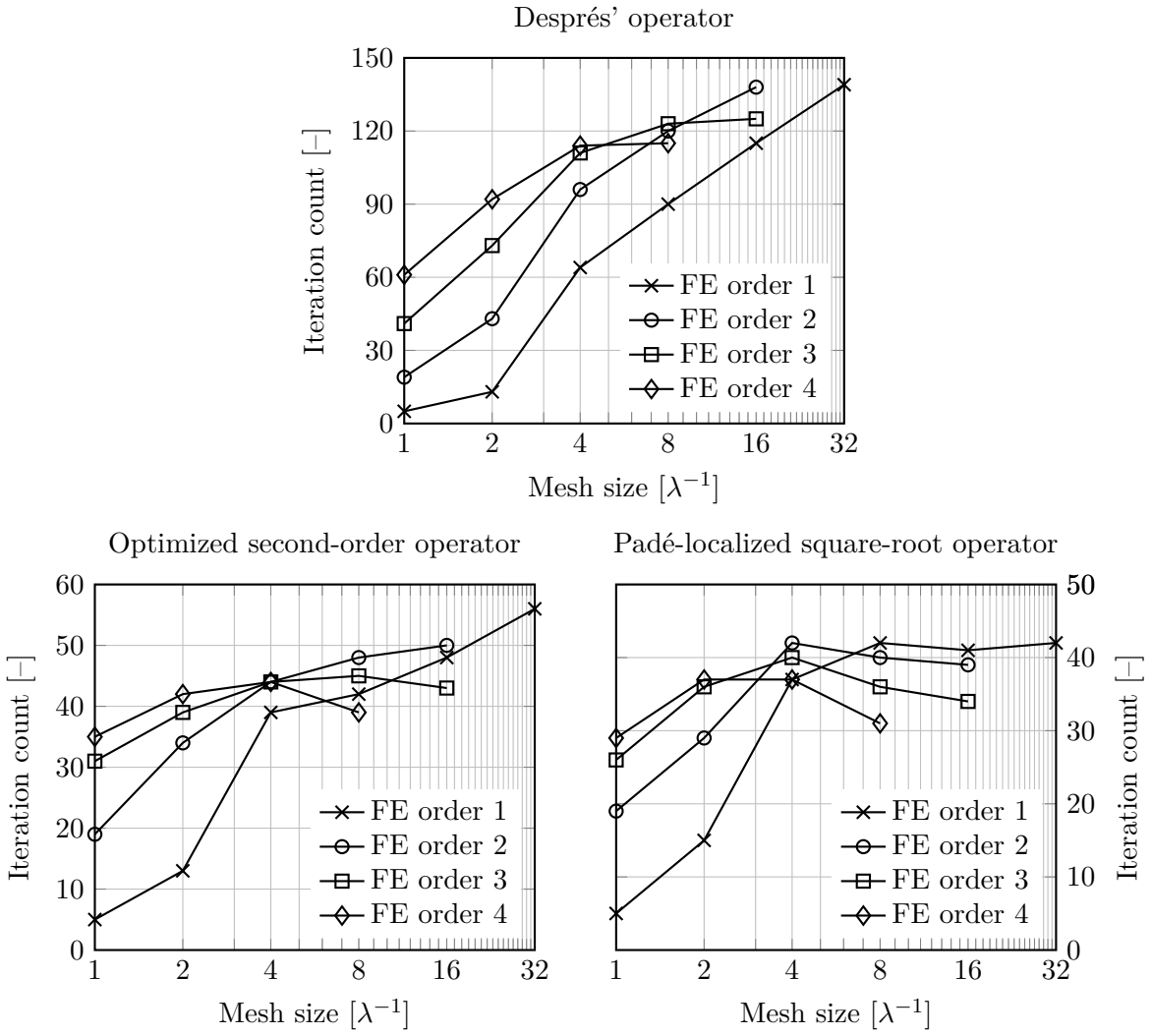


Figure 6.11: Iteration count for different discretization orders and mesh sizes.

count in Figure 6.11. To ease this analysis, Table 6.5 reports the accuracy level, as well as the iteration count, for a chosen set of simulations.

Without constituting a formal proof, Table 6.5 suggests that, for a given transmission operator, simulations with close relative L_2 errors lead to close iteration counts (with a maximum deviation of 20%). Furthermore, for the Padé-localized square-root operator, we can notice that increasing the accuracy of the simulation reduces the iteration count of the DDM. This behavior is not as clear for the two other operators.

6.9 Performance analysis: mixed discretizations

As we saw at the beginning of this chapter, domain decomposition methods split the original problem in two parts: a set of sub-problems defined on the sub-domains; and a single problem

FE order [–]	Mesh density [λ^{-1}]	Relative L_2 error [–]	Iteration count [–]		
			Despres	OO2	Pade
1	16	9.9×10^{-2}	115	48	41
2	4	9.8×10^{-2}	96	44	42
2	8	7.7×10^{-3}	120	48	40
3	4	7.2×10^{-3}	111	44	40
2	16	7.7×10^{-4}	138	50	39
4	4	7.5×10^{-4}	114	44	37
3	16	2.1×10^{-5}	125	43	34
4	8	1.9×10^{-5}	115	39	31

Table 6.5: Comparisons of simulations leading to a comparable accuracy.

defined on the interfaces between the sub-domains. In this section, we propose a brief analysis of the DDM, when the sub-problems and the interface problem are discretized with different FE orders.

Intuitively, by decreasing the FE order on the interface problem, we may expect to somehow low-pass filter high-frequency interface modes, which are known to penalize the DDM convergence rate (see section 6.2 or [45, 104] for instance). Thus, by using mixed discretization, we anticipate a DDM iteration count decrease, at a the cost of an accuracy loss.

6.9.1 Three-dimensional analysis

Again, we consider the waveguide of the previous section. However, this time, the discretization order for the sub-problems is set to 4, while the order for the interface problem ranges from 1 to 4. Since different discretization orders are involved, the following notation is used: $O\{e, g\}$, where e is FE order used to discretize the sub-problems, and g the order used for the interface problem.

Before presenting the results, let us focus on the FE discretization of the two optimized transmission conditions: OO2 and Pade. As explained in section 6.5.2, discretizing these operators requires solving auxiliary problems, which also need to be discretized by auxiliary function spaces. Empirically, we found the best performance by using a basis:

- of order g for discretizing the auxiliary problems in the OO2 operator,
- of order e for discretizing the auxiliary problems in the Pade operator.

As we did previously for the non-mixed version, let us first analyze the accuracy of this mixed approach. Figure 6.12 presents the relative L_2 error for different mesh densities and discretization orders for the interface problem. The sub-problems are solved with a 4^{th}-order finite element basis.

From these data, we can directly notice that by mixing orders, the accuracy of the solution is reduced, compared to the non-mixed case. This phenomenon is expected, since the interface field $\boldsymbol{g}_{i,j}$ of equations (6.50) and (6.51) is now represented by a lower-order basis, thus limiting

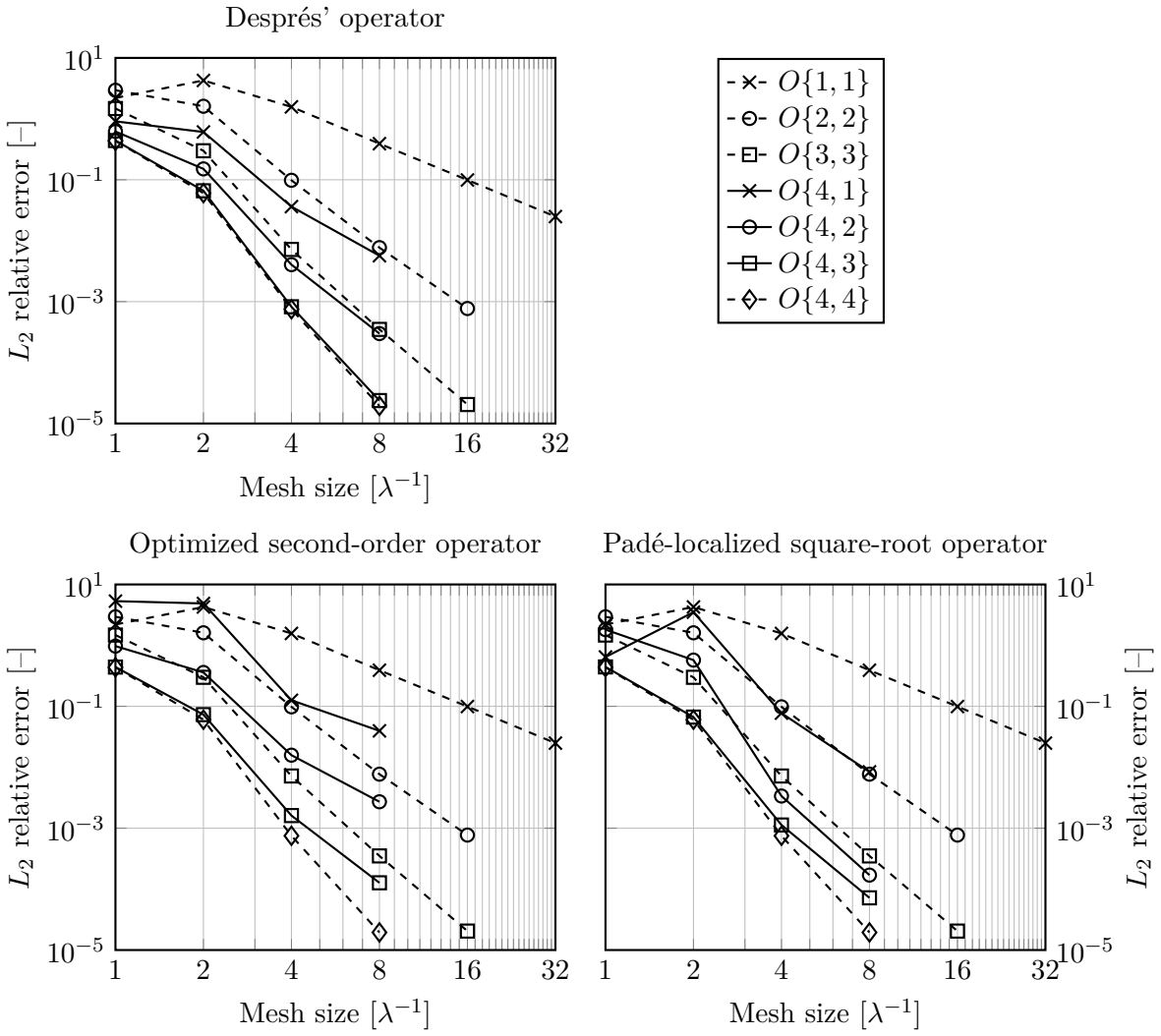


Figure 6.12: Relative errors between analytic and FE solution for different interface discretization orders and mesh sizes.

the precision on the original electric field. Intuitively, we expect the convergence rate to be bounded by the lowest discretization order. Table 6.6 presents the convergence rates, measured between 4 and 8 tetrahedra per wavelength.
For the Despres and Pade operators (in the mixed order cases) we are systematically above the expected rate (except for order $O\{4,3\}$, where the Pade operator exhibits the expected rate). It actually seems that the slope is $g+2$ instead of $g+1$, where g is the discretization order of the interface field. For the OO2 operator, the rate is systematically slightly less than the expected one. While these results are quite promising, we have to keep in mind that they are based on quite coarse meshes, and thus fall in a pre-asymptotic behavior.

	Measured convergence rate			
FE orders	Despres	OO2	Pade	Expected rate
$O\{4,1\}$	2.7	1.7	3.2	2
$O\{4,2\}$	3.8	2.5	4.3	3
$O\{4,3\}$	5.1	3.7	4.0	4
$O\{4,4\}$	5.3	5.3	5.3	5

Table 6.6: Convergence rates measured in Figure 6.12 between 4 and 8 tetrahedra per wavelength.

To conclude this analysis, let us now consider the iteration count of the DDM. Figure 6.13 presents the iteration count for the different test cases. From these data, we can directly notice that the iteration counts for the Despres and Pade operators have significantly decreased. Thus, at the cost of an accuracy decrease, the iteration count can be improved. This is not the case for the OO2 operator.

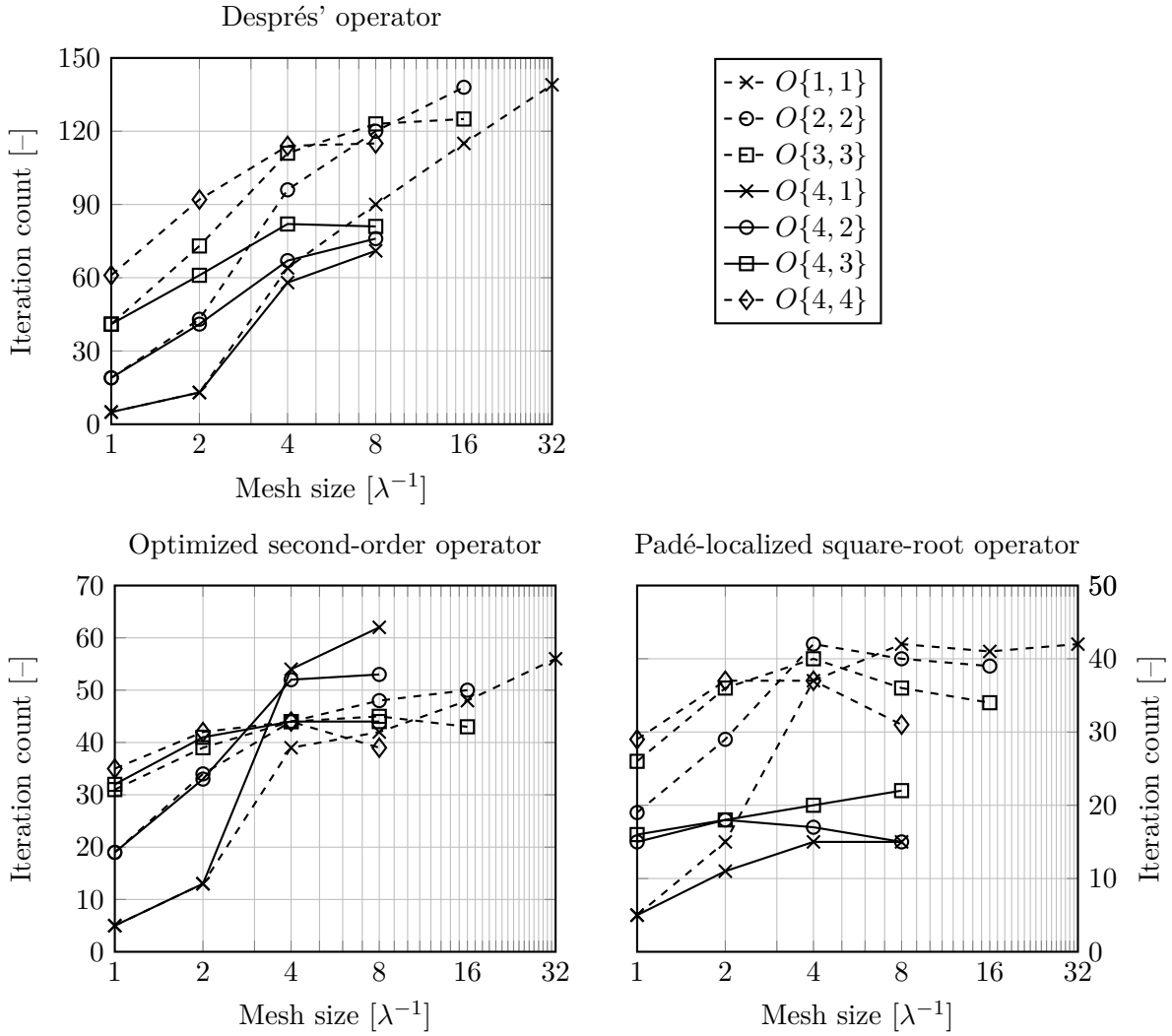


Figure 6.13: Iteration count of the DDM for mixed discretizations.

6.9.2 Two-dimensional analysis

In order to analyze more deeply the convergence rate of mixed order discretizations, let us now consider a two-dimensional case. This allows us to treat larger meshes with high-order discretizations. The new setup is identical to the three-dimensional one, except that the geometry loses one dimension, and that mode TM_1 is now considered.

Figure 6.14 depicts the measured errors, and Table 6.14 presents the convergence rates, using the simulations with 32 and 64 triangles per wavelength.

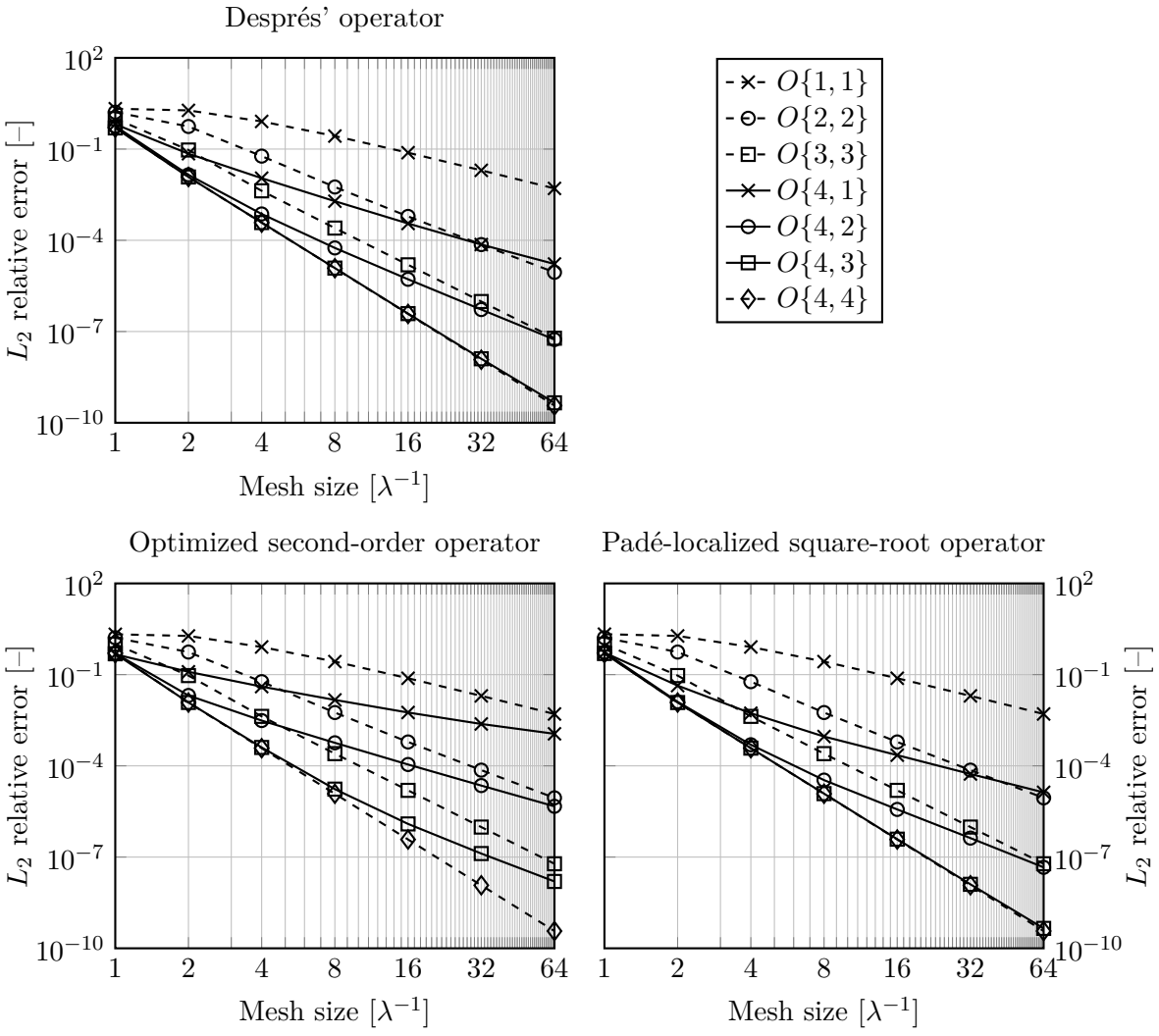


Figure 6.14: Relative errors between analytic and FE solution for different interface discretization orders and mesh sizes (two-dimensional case).

Based on these results, for the Despres and Pade operators, the convergence rate seems to be the expected one, except at order $O\{4,3\}$, where it seems to be equal to $g+2$ (g being the discretization order for the interface field). On the other hand, for the OO2 operator, the convergence rate seems to be equal to g.

	Measured convergence rate			
FE orders	Despres	OO2	Pade	Expected rate
$O\{4,1\}$	2.1	1.0	2.0	2
$O\{4,2\}$	3.2	2.3	3.2	3
$O\{4,3\}$	4.8	3.0	4.8	4
$O\{4,4\}$	5.0	5.0	5.0	5

Table 6.7: Convergence rates measured in Figure 6.14 between 32 and 64 triangles per wavelength.

6.10 Conclusion

In this chapter, we have presented the optimized Schwarz algorithm, and we compared different transmission condition, both for acoustic and electromagnetic simulations. We showed that the OO2 and Pade operators are significantly improving the convergence rate of the DDM. While the Pade leads the fastest convergence, it also exhibits the highest iteration cost.

We also analyzed the behavior of the Schwarz algorithm, when high-order finite element discretizations are used, by using a numerical experiment: the propagation through a three-dimensional rectangular metallic waveguide. First, we observed, that by keeping the relative L_2 error (between the simulation and the exact solution) constant, the DDM iteration count was not impacted by a modification in the mesh refinement and the FE discretization order. Moreover, for the Padé-localized square-root operator, we noticed an iteration count decrease on simulations with higher accuracy.

We also considered simulations, with different FE discretization orders for the sub-domain problems and the interface problem. If we call g the discretization order for the interface problem, we found a convergence rate of $g+2$ for the Despres and Pade operators, and a rate of $g+1$ for the OO2 operator. Let us note that g was lower than the discretization order used for sub-domains problem. Since these results were obtained on quite coarse meshes, a two-dimensional waveguide was further considered. For this configuration, we found a convergence rate of $g+1$ for the Despres and Pade operators, and a rate of g for the OO2 operator. Interestingly, for the Despres and Pade operators, when g was just one order below the order used for the sub-problems, we measured a convergence rate of $g+2$.

Chapter 7

Large-scale simulations of a segmented waveguide

In this final chapter, the scaling of the optimized Schwarz algorithm is tested, by considering simulations on a segmented waveguide. Both strong and weak scaling are analyzed, as well as the memory distribution across the computing nodes.

7.1 Introduction

For this final chapter, let us apply the high-order finite element method and the optimized Schwarz algorithm to a more realistic test case: an open segmented waveguide. Basically, these waveguides consist in a chain of several equispaced non-metallic cylinders[1] in open space. For some frequencies, the interference pattern between the cylinders leads to a guided wave, even if the system is open [101, 102, 114]. This kind of guides are of high interest when building integrated photonic circuits, notably for their low cross-talk at waveguides intersections [114].

In this section, a device similar to [101] will be used as a realistic test case, for testing the scaling of the Schwarz algorithm. In what follows, the Padé-localized square-root operator is used [45], with a non-restarted GMRES set to a relative tolerance of 10^{-6}.

7.2 Numerical setup

The waveguide consists in a periodic arrangement, into vacuum, of cylinders with a relative electric permittivity of 6. They have a radius of 5[mm] and a height of 30[mm]; the distance between two cylinder centers is $T = 15$[mm]. In what follows, a Cartesian coordinate system is assumed: the cylinders are aligned along the x-coordinate, and that their axis are aligned with the z-coordinate.

The system is excited by a signal at 6[GHz]: *i.e.*, with a wavelength of $\lambda = 50$[mm]. The source electric field is imposed by a sphere Ω_s with a radius of 2.5[mm], and at a distance

[1] Or any other kind of geometrical shapes.

of 15[mm] of the first cylinder (taken between centers). On its boundary $\partial\Omega_s$, the following signal is imposed:

$$\gamma^T(\boldsymbol{e}) = \left([0, 0, \jmath\omega\mu_0]^T \cdot \boldsymbol{\tau}\right) \boldsymbol{\tau} \qquad \text{on } \partial\Omega_s,$$

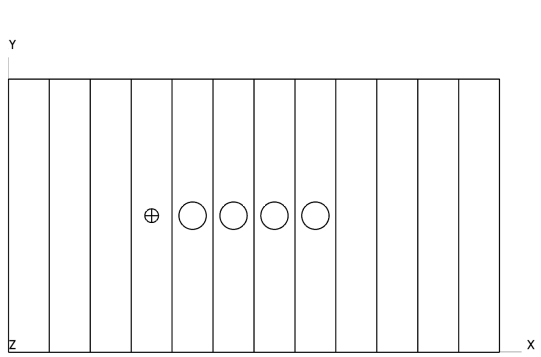
where $\boldsymbol{\tau}$ is the unit vector tangent to the sphere boundary, and where ω is the signal angular frequency.

The infinite domain is truncated by a rectangular parallelepiped, and a Silver-Müller condition (1.20) is imposed on its boundary. This truncation is made at a distance of:

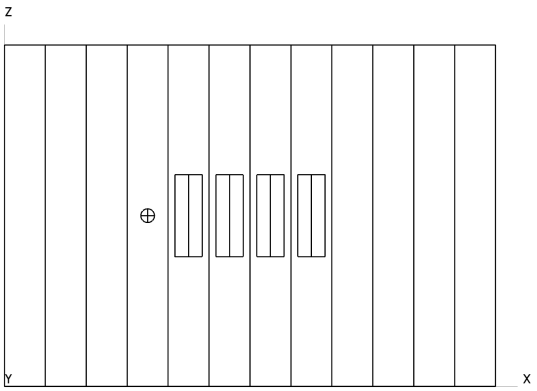
- 1λ of the cylinders along the y- and z-coordinates,
- $4T$ of the cylinders along the x-coordinate[2].

Because of its periodic nature, this geometry is well suited for applying the Schwarz algorithm. It is then decomposed along the x-coordinate into cells of size T. A cell contains either a cylinder, the spherical source, or is void.

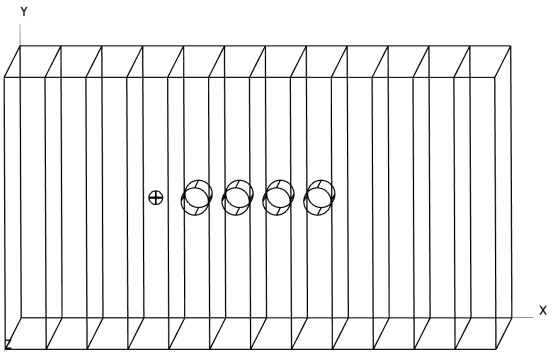
Finally, the computational domain is discretized with 10 tetrahedra per wavelength. Because of the curved nature of the system, second-order geometrical mesh elements are used. Figure 7.1 illustrates this geometry for a configuration with 4 cylinders and 12 cells.



(a) View of the xy-plane.



(b) View of the xz-plane.



(c) Three-dimensional view.

Figure 7.1: Geometry of the segmented waveguide: 4 cylinders and 12 cells.

[2]Let us note that $4T$ is equal to 1.2λ.

7.3 Strong scaling

For this first analysis, let us consider a case with 32 cells, that is with 24 cylinders. It is worth noticing that at a frequency of 6[GHz], this corresponds to a 9.6λ long structure.

This domain is decomposed into 2^n sub-domains, where the integer n ranges between 1 and 5. A sub-domain is then constructed by grouping cells: thus, each sub-domain is composed of 2^{5-n} cells.

For each value of n, a simulation with 2^n processes is carried out (a process is responsible for one sub-domain). Let us note that each process is associated to only one thread. Moreover, two finite element discretizations are considered: one of order 2, and one of order 3.

7.3.1 Second-order case

Let us begin with the second-order case. Figure 7.2 presents the total computation time for each simulation, as well as the DDM iteration count. As expected, we can directly notice that the iteration count increases with the number of sub-domains (or, equivalently, the number of processes). This explains the behavior of the computational time. As the number of processes increases:

1. more sub-problems of smaller size are solved in parallel, which tends to decrease the computation time;
2. the iteration count of the DDM increases, which tends to increase the computation time.

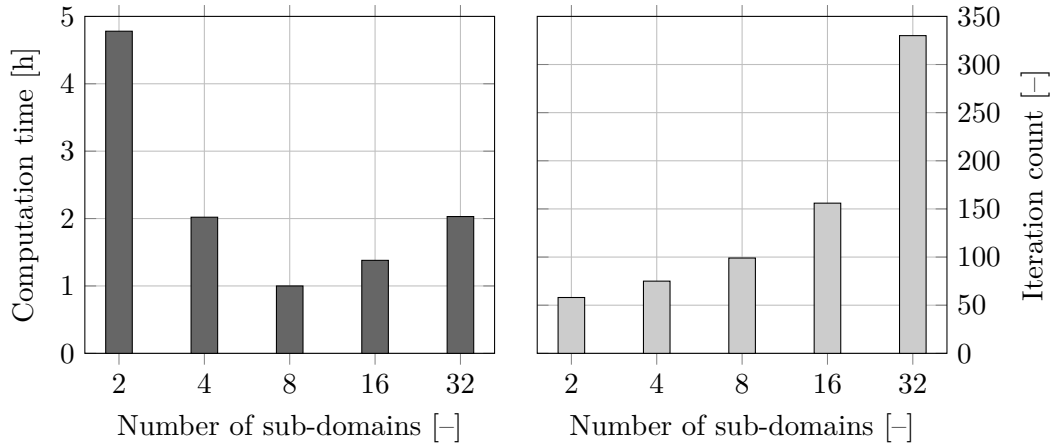


Figure 7.2: Computation time and iteration count: second-order strong scaling.

Another interesting quantity to look at, is the ratio between the number of iteration and the computation time. This is presented in Figure 7.3. We can directly notice that, the number of iteration per hour increases with the number of sub-domain. Thus, the scaling of the computation time is only limited by the iteration count increase, and not by other factors, such as for instance, the network bandwidth.

Figure 7.4 depicts the mean (from all the processes) peak virtual memory, or VmPeak, and

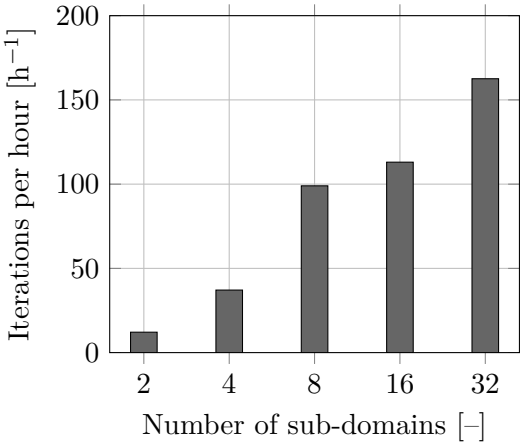


Figure 7.3: Iterations per hour: second-order strong scaling.

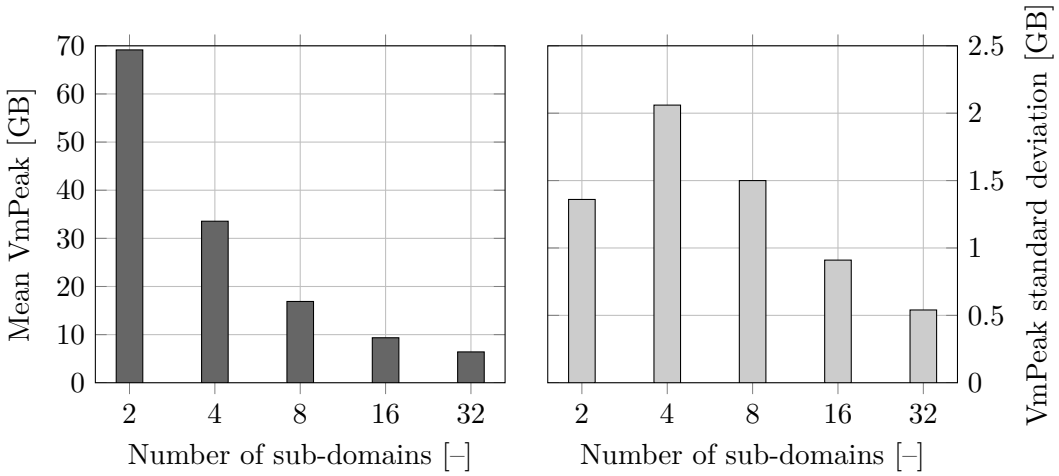


Figure 7.4: Mean peak virtual memory (VmPeak) and its standard deviation: second-order strong scaling.

its standard deviation. Let us note that the original problem exhibits 2568483 unknowns. Based on these data, we can directly notice that, as we increase the number of sub-domains, the memory requirement of each process decreases. Moreover, the decreasing standard deviation suggests an excellent memory distribution between the processes. This is further confirmed by Figure 7.5, which depicts the peak virtual memory for each process in the 32 sub-domains case.

By going back to Figure 7.5, we notice that the first and last processes deviate significantly from the ideal memory distribution. This phenomenon is explained as follow. These two processes are associated to the first and last sub-domains. Thus, these processes have only one neighbor: this means that only one DDM interface term is needed. Therefore, their memory requirements will be lower than the other processes.

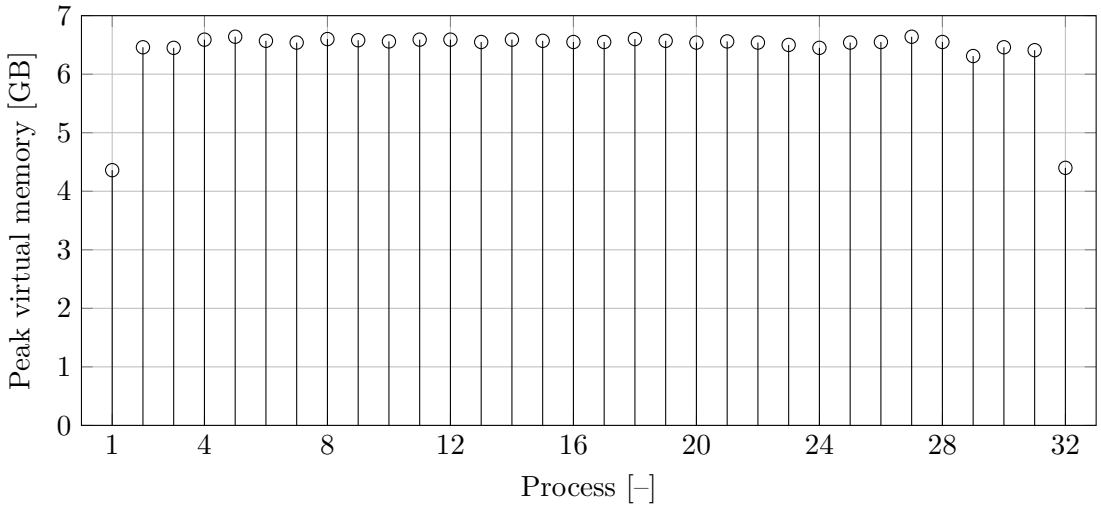
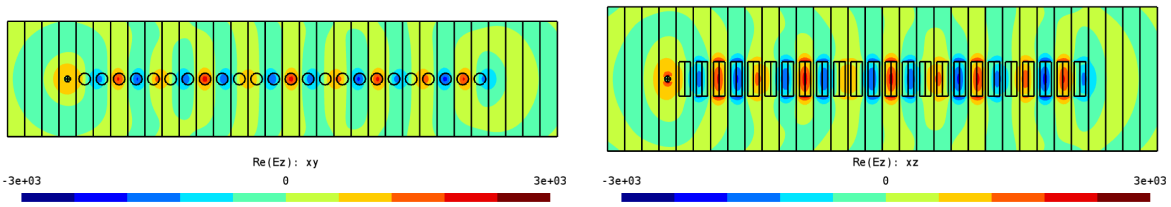


Figure 7.5: Per process peak virtual memory: second-order strong scaling, 32 sub-domains case.

Finally, for illustration purposes, Figure 7.6 depicts the real part of the z-component of the electric field. Two cuts crossing the cylinders centers are considered: one along the xy-plane, and another along the xz-plane. It is worth remarking that the wave is indeed guided by the segmented structure.



(a) Cut along the xy-plane.

(b) Cut along the xz-plane.

Figure 7.6: Real part of the z-component of the electric field: second-order strong scaling, 32 sub-domains case.

7.3.2 Third-order case

Let us now consider the third-order case. For memory reasons, the 2 and 4 sub-domains cases were not possible. It is worth noticing that the original problem has 6599284 unknowns.

As we did for the second-order case, we start by the total computation time, the DDM iteration count, and their ratio. Figures 7.7 and 7.8 present the obtained results, which are very similar to the second-order case. Indeed, we may observe that by increasing the number of sub-domains:

1. the iteration count increases,
2. the computation time first decreases and then increases,

3. the number of iteration per hour increases.

Again, the scaling is only limited by the iteration count increase with the number of sub-domains.

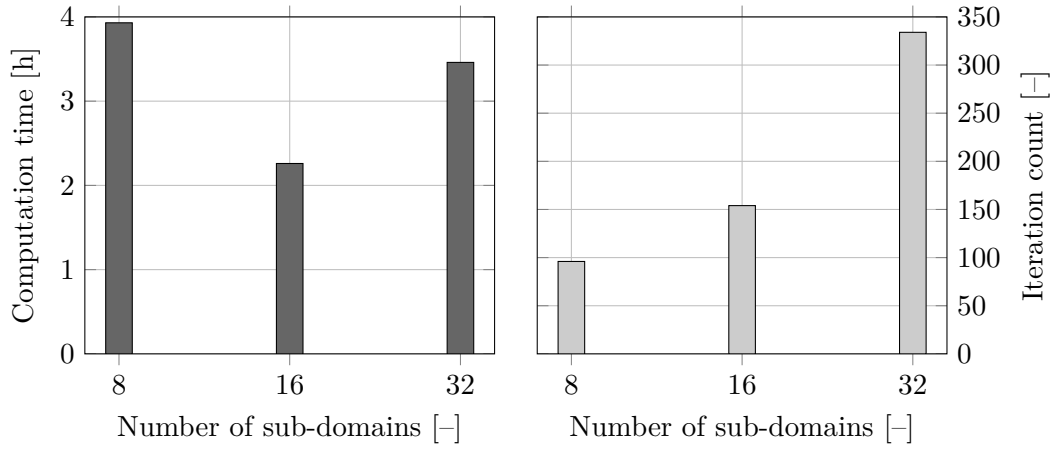


Figure 7.7: Computation time and iteration count: third-order strong scaling.

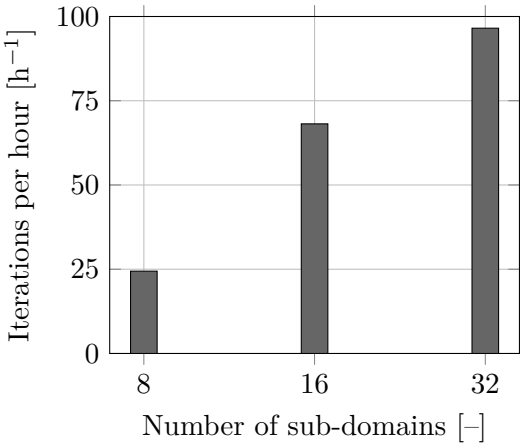


Figure 7.8: Iterations per hour: third-order strong scaling.

On the memory side, we can also notice the same behavior as for the second-order case in Figure 7.9. As we increase the number of sub-domains (or, equivalently, the number of processes), we decrease the per process memory requirements, as well as the standard deviation to the mean value: this also suggests an excellent memory distribution. This is confirmed in Figure 7.10, which depicts the per process peak virtual memory for the 32 sub-domains case. Again, the first and last processes allocate a lower amount of memory.

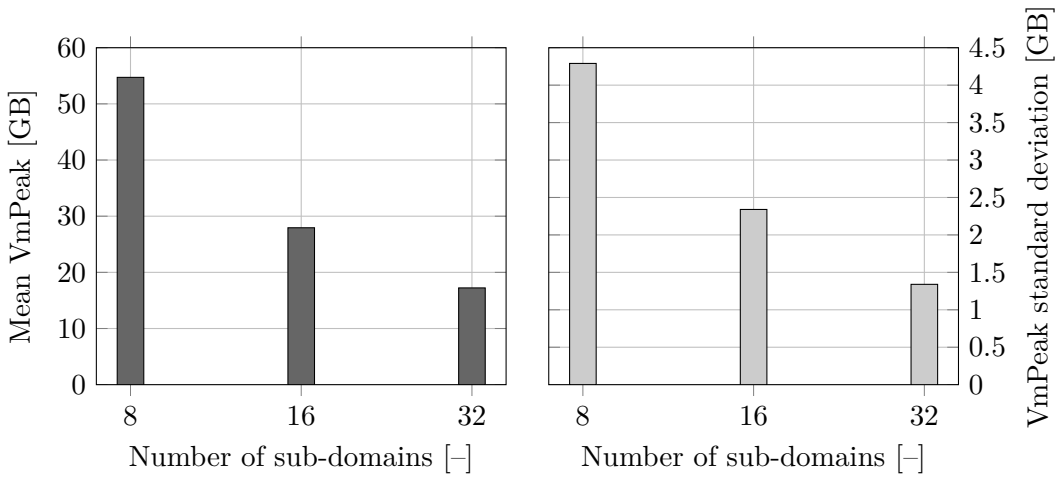


Figure 7.9: Mean peak virtual memory (VmPeak) and its standard deviation: third-order strong scaling.

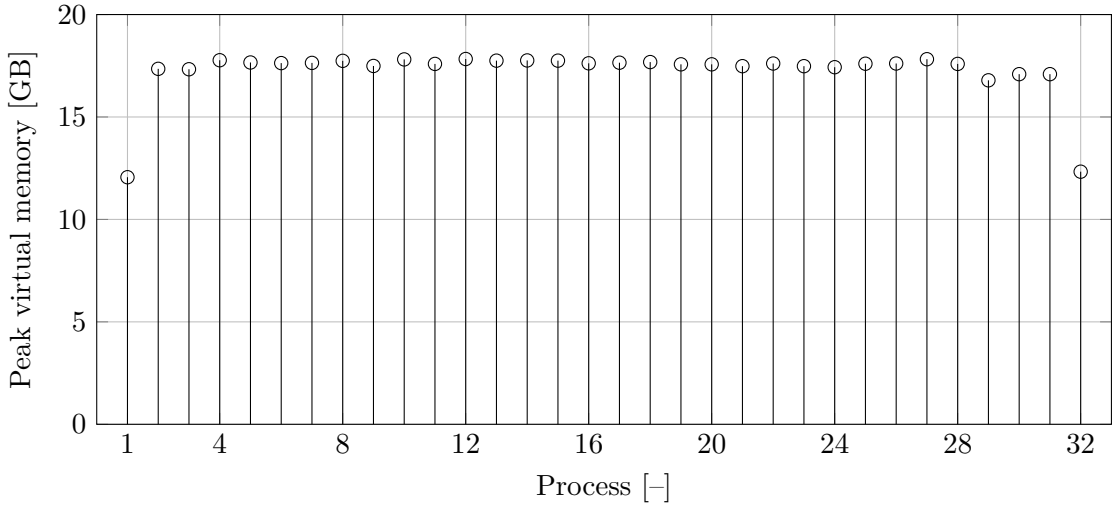


Figure 7.10: Per process peak virtual memory: third-order strong scaling, 32 sub-domains case.

7.4 Weak scaling

For this last analysis, instead of a constant problem, we consider a set of simulations with an increasing size. More precisely, simulations will be carried out on configurations with 2^n cells (or, equivalently, $2^n - 8$ cylinders), where the integer n ranges between 5 and 9. Each cell is then associated to a single sub-domain (or, equivalently, a process) of the Schwarz algorithm. It is worth noticing that for the 512 cells case, the structure is 153.6λ long.

As for the strong scaling case, second- and third-order simulations will be treated. However, this time, the number of threads per process is set to 4. Furthermore, the cluster used in this

case has a faster interconnect than the one used for the strong scaling case[3].

7.4.1 Second-order case

As done previously, let us start be the second-order case. By analyzing the total computation time and the DDM iteration count. Figure 7.11 presents those data. We can directly notice that both the computation times and the iteration counts are increasing with the number of sub-domains. Basically, since all the cells are approximately of the same size, increasing the parallelism level cannot decrease the computation time associated to a sub-domain. Thus, if the iteration count increases, the computation time must also increase. This explanation is confirmed by Figure 7.12, depicting the ratio between the iteration count and the computation time: it is roughly constant with the number of sub-domains. Therefore, we can conclude that, the scaling of the computation time is limited only by the iteration count increase.

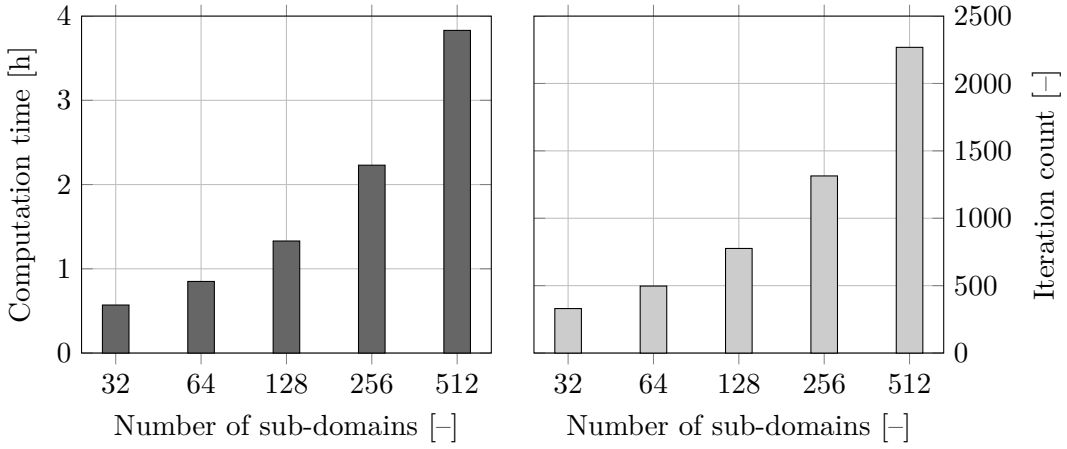


Figure 7.11: Computation time and iteration count: second-order weak scaling.

Let us now focus on the peak virtual memory, as depicted in Figure 7.13. By analyzing the results, we can note that the mean memory usage increases with the number of sub-domains. As first guess, one may have expected this value to remain constant, since the size of a sub-domains does not change. This is indeed the case, however, as the number of iterations increases, the GMRES Krylov sub-space size also increases[4]: thus the peak virtual memory increases. It is worth noticing that by multiplying the problem size by 16, the memory usage increased by a factor of 2.98. Finally, the low standard deviation indicates an excellent distribution of the memory across the processes.

[3]It was not possible to carry out the simulations of the strong scaling case on this fast cluster; indeed, because of their smaller size, they were bellow the minimum size limit of the fast cluster.

[4]Let us remember that our GMRES is not restarted.

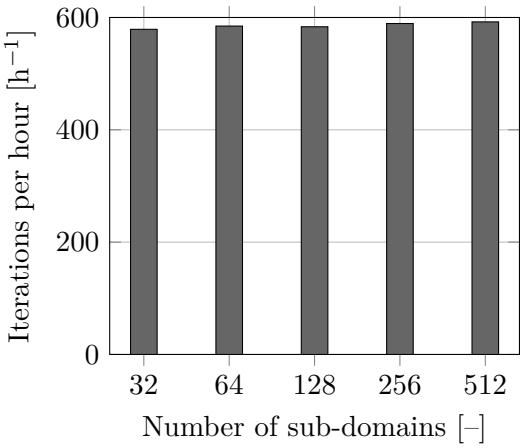


Figure 7.12: Iterations per hour: second-order weak scaling.

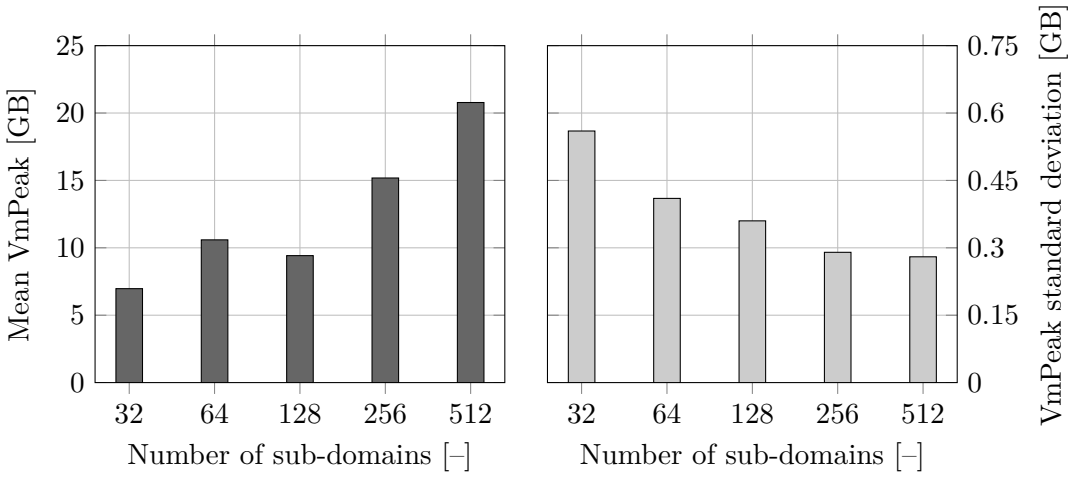


Figure 7.13: Mean peak virtual memory (VmPeak) and its standard deviation: second-order weak scaling.

7.4.2 Third-order case

Let us now analyze the third-order case. For this set of tests, the 512 sub-domains case was skipped: this simulation exceeds the maximum number of nodes, a user can normally request on the cluster.

For the computation time and the iteration count, Figures 7.14 and 7.15 depict the same behavior as the second-order case. The number of iterations per hour is roughly constant, and since the iteration count increases with the number of sub-domains, the total computation time follows this increase. Again, we can conclude that, the scaling of the computation time is limited only by the iteration count increase.

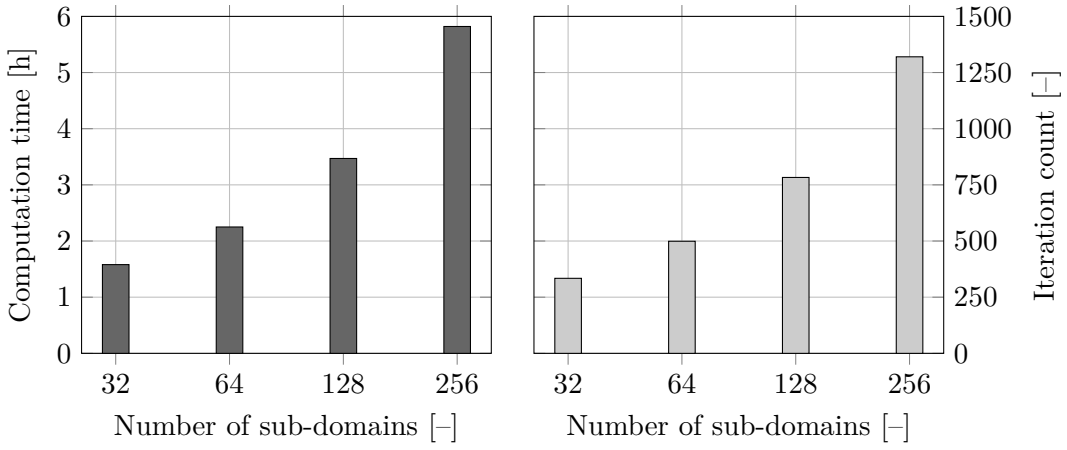


Figure 7.14: Computation time and iteration count: third-order weak scaling.

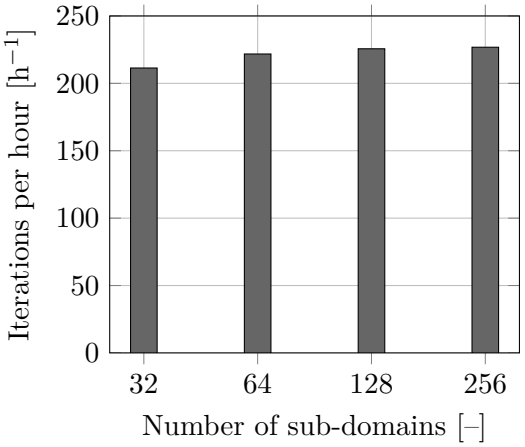


Figure 7.15: Iterations per hour: third-order weak scaling.

The same behavior, as the second-order case, is exhibited for the main peak virtual memory, as depicted in Figure 7.16: as the iteration count increases, the GMRES Krylov sub-space increases also, thus leading to an overall expansion of the memory requirements. However, by increasing the problem size by a factor 8, the memory usage increased by only a factor 1.65.

Again, the low standard deviation to the mean peak virtual memory suggests an excellent distribution across the processes.

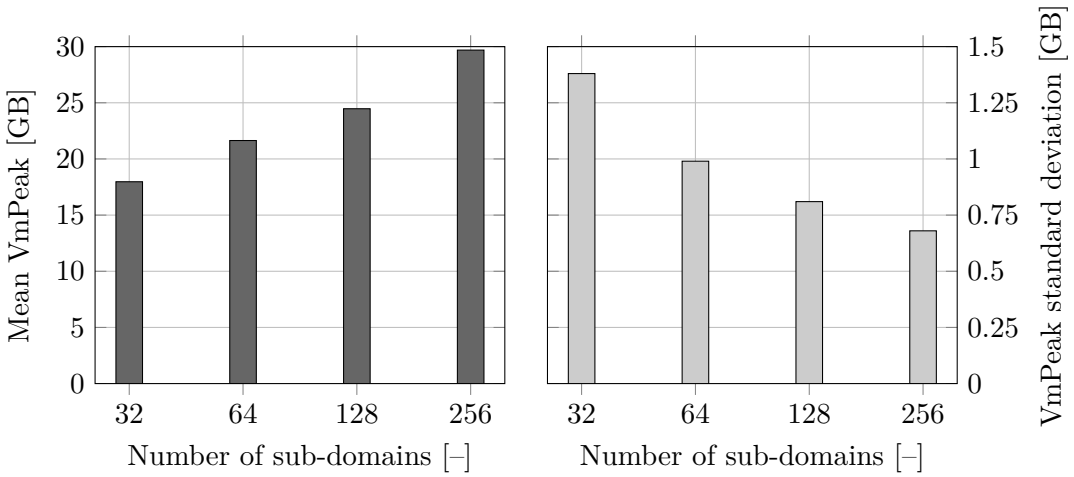


Figure 7.16: Mean peak virtual memory (VmPeak) and its standard deviation: third-order weak scaling.

7.4.3 Original system sizes

Last but not least, to give some orders of magnitude on the simulation sizes, we report in Figure 7.17 the number of unknowns of the original (*i.e.*, without the DDM) problems. The largest problem exhibits 41047026 unknowns for the second-order discretization, and 52696804 for the third-order case.

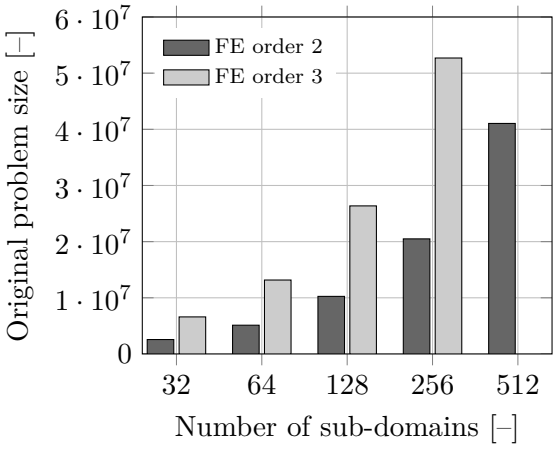


Figure 7.17: Number of unknowns of the original problems: weak scaling.

It is worth recalling that solving the 41047026 unknowns problem (*i.e.* a 76.8λ long structure) required 4 hours, 512 MPI processes and 20[GB] of memory per process. On the other

hand, the 52696804 unknowns problem (*i.e.* a 153.6λ long structure) required 6 hours, 256 MPI processes and 30[GB] of memory per process.

7.5 Conclusion

In this chapter, we carried out strong and weak scaling tests on a segmented waveguide. Since the DDM iteration count increases with the number of sub-domains (or, equivalently, the number of processes), the computation time must also grow, when the parallelism degree is increased. However, by considering the iteration number per hour, we found that:

1. it increases on strong scaling tests;
2. it remains constant on weak scaling tests.

This suggests that, the scaling of the computation time is limited by the iteration count increase, and not by the implementation. As already mentioned in chapter 6, preconditioners can be used to address this problem [129, 130].

Finally, by analyzing the memory usage of the strong and weak tests, we found that the mean peak virtual memory:

1. decreases on strong scaling tests;
2. increases slightly on weak scaling tests, because of the GMRES Krylov sub-space size increase (caused by a larger number of iterations).

Moreover, a low standard deviation to the mean peak virtual memory suggests an excellent memory distribution across the processes. It is worth recalling, that thanks to the Schwarz algorithm, we performed simulations up to 50 million unknowns.

Conclusion

"The Answer to the Great Question" [...] "Of Life, the Universe and Everything" [...] "Is" [...] "Forty-two," said Deep Thought, with infinite majesty and calm.

— D. Adams, *The Hitchhiker's Guide to the Galaxy.*

In this thesis, we have developed efficient methods for the treatment of large-scale time-harmonic wave simulations. To achieve this purpose, we first investigated the high-order finite element method. Thanks to its low dispersion property, it leads to accurate solutions of high-frequency wave problems, while keeping the number of unknowns low, relatively to first-order discretizations. Unfortunately, since this approach relies on the integration of many high-order polynomials, the assembly of the finite element matrix is computationally expensive, and can even exceed the time taken to solve the resulting linear system.

To compensate this drawback, efficient assembly procedure are needed. We proposed to extend the solution proposed in [64, 65, 83], for the high-order Lagrange discontinuous Galerkin method, to vector-valued and non-nodal problems. This approach consists in the computation of the numerical quadrature of the finite element terms, by using a computationally efficient matrix-matrix product. The performances of this technique was tested on various cases, and large speedups (sometimes up to 20) were found.

When handling high-order scalar bases, or vector bases, the finite element reference space must be oriented. Classically, this operation is performed on-the-fly during the assembly phase. Unfortunately, this approach is incompatible with the efficient assembly procedure, and an alternative approach was developed. The proposed solution relies on an *a priori* orientation of the mesh elements, by using a newly designed orientation dictionary structure. It is worth recalling that in the special case of Lagrange bases, this operation is quite simple, and is implemented by a swapping of the basis functions. However, this simple technique cannot be extended to more general bases.

With these new tools in hand, we performed highly accurate computations of the $\mathrm{TEM}_{9,0,0}$ mode, of an open resonator with an extremely high quality factor. By using high-order finite element discretizations, and high-order geometrical mesh elements, we analyzed the convergence of the eigenmode damping time. The simulations size were limited to roughly 10 million unknowns. Indeed, in the processing chain, an **LU** decomposition must be performed, which exhibits a bad memory scaling.

Since direct methods do not scale, and since iterative methods do not converge well (or not at all), alternative solutions must be found to solve the large linear systems, resulting

from the high-frequency problems discretization. Among the possible candidates, domain decomposition methods, that couple direct and iterative strategies, are promising. In this work, we focused on the optimized Schwarz algorithm, and we analyzed its performances when coupled with higher-order finite elements. By using numerical experiments, we observed that, at a given precision level, the iteration count is not impacted by a modification of the mesh density, and of the discretization order. Moreover, for the Padé-localized square-root operator, we noticed an iteration count decrease on simulations with higher accuracy.

We also considered the case when different discretization orders, for the sub-domain problems and the interface problem, are used in the domain decomposition. By considering a two dimensional waveguide test case, we experimentally found a convergence rate of $g+1$ for the Despres and Pade operators, and a rate of g for the OO2 operator, where g is the order used to discretize the interface problem. Let us note that we chose g lower than the order used for the sub-problems.

Finally, we carried out strong and weak scaling tests on a segmented waveguide. By using the optimized Schwarz algorithm, we were able to solve propagation problems, with up to 50 million unknowns. The considered problems were discretized using high-order bases, and curved mesh elements. In contrast with the resonator case, we were not able to treat larger problems, because we simply reached the maximum number of nodes, a user can normally request on the used cluster.

Perspectives

In the course of this thesis, we have identified many improvements and unanswered questions. In what follows, we discuss the main perspectives, sorted from the most short-term one to the most long-term one.

1. *Extension of the finite element library to discontinuous Galerkin formulations.*
   As already explained, the developed finite element assembler reuses the idea from the discontinuous Galerkin (DG) world. More precisely, we first perform a DG assembly, which is then recombined into a classical continuous finite element linear system. Thus, the assembly kernel is ready for DG formulations. However, the higher-level abstractions that use this kernel are not. Therefore, a first perspective is to extend the developed library, for the treatment of continuous and discontinuous formulations into a unified framework. With this tool, an easy treatment of mixed DG formulations [67], as well as coupled continuous-discontinuous formulations [2, 29], will be made possible.
2. *Removing unnecessary copies from the assembly kernel.*
   In the current implementation, for practical reasons, the finite element terms computed by the matrix-matrix product have to be copied into another memory segment. This copy limits the scaling of the implementation: for instance, if 12 threads are used for the assembly (instead of the 6 used in the tests of chapter 3), the efficiency of the new procedure drops (compared to the classical one). It is possible to remove those unnecessary copies, but at the cost of a change in the library class interfaces, thus leading to a non-trivial implementation rework. These modifications fit into our perspectives.

3. *Proper comparison of fully tweaked direct solvers and domain decomposition approaches.*
   In chapter 5, we found the limitations of the MUMPS solver at roughly 10 million unknowns. However, we used it out of the box, and did not tweak its many parameters. Moreover, we did not present timing comparison between direct and domain decomposition approaches. Therefore, the following pragmatic question remains unanswered: when is the use of a domain decomposition strategy more interesting than a state-of-the-art direct solution?

4. *Integrating the finite element library into a higher-level environment.*
   The library was developed to perform the assembly of finite element terms: *i.e.*, basically, to treat finite element spaces, quadrature rules, degrees of freedom, and so on. However, this is only a piece of a complete simulation environment. Thus, we also developed pre- and post-processing modules, that unfortunately lack flexibility. Therefore, we intend to integrate the finite element kernel into a more general environment [41, 128].

5. *Extension to other physics.*
   It this work, we focused on time-harmonic wave problems for acoustics and electromagnetism. However, the developed framework can be used in a larger set of disciplines. A (relatively) straightforward extension is the treatment of linear time-harmonic elastodynamic problems, with applications to geological prospecting for instance. Eventually, with a flexible high-level environment, and the ability to handle both continuous and discontinuous formulations, additional physics and more complex, larger-scale industrial problems will also be tackled.

6. *Rigorous error analysis of mixed order discretizations in the Schwarz algorithm.*
   When analyzing the impact of a discretization order decrease in the interface problem of the Schwarz algorithm, we only considered numerical experiments. On the two-dimensional case, we observed a convergence rate of $g + 1$ for the Despres and Pade operators, where g is the discretization order used for interface problem. This results is expected, since, intuitively, we presume that the smallest order will limit the convergence rate to $g + 1$. However, on the OO2 operator, and on the coarse three-dimensional analysis, we noticed a different behavior. To properly understand this, a rigorous error assessments must be carried out. Moreover, the impact on the convergence rate of the auxiliary spaces, used in the OO2 and Pade operators, must be assessed.

7. *Treatment of cross points and coarse grids in the Schwarz algorithm.*
   When we presented the Schwarz algorithm, we considered sliced decompositions, thus avoiding the presence of cross points. However, this limits significantly the range of applications of our implementation. The proper treatment of the cross points, in an optimized Schwarz context, still needs to be investigated. Recent developments in the area of coarse grids [28, 73] should also be integrated, in order to improve the strong and weak scaling of the method on realistic applications.

8. *Treatment of eigenvalue problems with domain decomposition methods.*
   As already showed, the Schwarz algorithm can treat large-scale simulations. However, it is not clear how this approach can be used for solving eigenvalue problems. As long-term perspective, we intend to study how domain decomposition methods can be extended to eigenvalue computations.

— `return 0;`

Appendix A

Finite element code

Write it, Cut it, Paste it, Save it,
Load it, Check it, Quick-Rewrite it.
— Daft Punk, *Technologic.*

In this appendix, we present the software architecture of the code developed during this thesis. After a brief overview of the main modules, each of them is analyzed in more details. This appendix concludes with convergence tests, which validate the implementation.

A.1 General overview

As already stated in the introduction of this thesis, a finite element C++ library has been developed from scratch, and is available on the subversion tree[1]:

`https://onelab.info/svn/gmsh/trunk`,

in the directory `./projects/small_fem/`. In order to handle the non finite element features, the developed code relies on the following third party libraries:

1. the OpenBLAS[2] [134] (version `0.2.13`) implementation of the BLAS (Basic Linear Algebra Subprograms) interface [18, 38, 39, 84], used to handle the elementary algebraic operations;
2. the direct solver MUMPS[3] [3, 4] (version `5.0.0`), used to compute the solution of the generated finite element linear systems;
3. the GMRES implementation of the PETSc library[4] [11, 12] (version `3.6.0`), used for the Krylov acceleration of the implemented Schwarz algorithm;
4. the SLEPc library[5] [62, 107] (version `3.6.0`), used to solve eigenvalue problems;

[1] Login: gmsh; password: gmsh.
[2] Available at: `http://www.openblas.net`.
[3] Available at: `http://mumps-solver.org`.
[4] Available at: `http://www.mcs.anl.gov/petsc`.
[5] Available at: `http://slepc.upv.es`.

5. the Gmsh library[6] [54], used to handle the mesh related features.

This code is divided into modules, each of them being a collection of classes collaborating in a common objective. In its current revision (22464), the developed software exhibits eight modules: `assembler`, `common`, `context`, `formulation`, `geometry`, `postprocessing`, `solver` and `functionspace`. The classes of each module are available in a directory (bearing the same name) located in `./projects/small_fem/`. In the following sections, more precise descriptions of these modules are proposed.

In this thesis, a high-level environment for describing a finite element problem was not developed. Instead, independent executable files, linked with the library, are written for each specific situation. The source codes of these files are located in the `simulation` folder. For instance, the source code `Haroche.cpp` implements the test case discussed in chapter 5, and the source code `BoubouchonsSomDdm.cpp` implements the test case of chapter 7.

A.2 The `geometry` module

Let us start by focusing on geometry related aspects of the code. A mesh element itself is represented by the **`MElement`** class of the Gmsh library. Moreover, the vertices, edges and faces of the an element are represented by the **`MVertex`**, **`MEdge`** and **`MFace`** classes of Gmsh. By exploiting these classes (and descendants), the `geometry` module offers useful higher level abstractions and containers.

A.2.1 A mesh

The first building block of the `geometry` module is the **`Mesh`** class. As its name suggests, it handles the mesh (by giving its file path at construction time), and it offers the following services:

1. for every **`MElement`**, **`MVertex`**, **`MEdge`** and **`MFace`** of the mesh, a **`Mesh`** object can return a unique positive integer identifying them;

2. a **`Mesh`** object can also return an ordered list of **`MElement`**s (see **`GroupOfElement`**), associated to a given physical group identifier[7], and possibly a given mesh partition identifier[8].

[6] Available at: `http://gmsh.info`.

[7] A physical group can be thought as a part (defined by the user) of the geometry; each part can be given an integer value, which is the physical group identifier; once the geometry meshed, it is possible to find the **`MElement`**s associated to a given physical group.

[8] The difference between the partition and the physical group is the following: a physical group is primarily an geometry based concept (it is defined without a mesh), while the partition is a mesh based concept; these two identifiers are complementary: it is possible to extract the **`MElement`**s associated to both a given physical group (that was defined at the geometry design stage), and a given partition (that was defined during the meshing stage).

A.2.2 Container for elements

The `GroupOfElement` class is an *ordered* `MElement` container. Its main property is the following: the contained `MElement`s are first sorted by geometrical types[9] and then by orientation tags (for a given type). In addition to this property, this class offers the following services:

1. it can give access to a vector[10] with the sorted `MElement`s;
2. it can return the number of `MElement`s (in the container) with a given geometrical type;
3. for a given geometrical type, it can return the number of `MElement`s (in the container) with the same orientation.

It is worth noticing that, thanks to the last two services and the sorting property, it is possible to find, in the `MElement` vector, the first and last indices of the `MElement`s sharing a common type and orientation.

A.2.3 Container for Jacobian matrices

To compute the finite element integrals, it is required to compute the Jacobian matrix for every mesh element and for every integration point. To handle all these matrices, two containers have been designed:

1. the `Jacobian` class, containing the Jacobian matrices evaluated at different integration points, for a given mesh element;
2. the `GroupOfJacobian` class, containing `Jacobian` objects, for every mesh element contained in a given `GroupOfElement`.

A.3 The `functionspace` module

As its name suggests, this module handles the function spaces, used for the finite element discretization.

A.3.1 Reference spaces

The first ingredient of this module is the `ReferenceSpace` interface, implementing the notions of oriented reference spaces, as presented in chapters 2 and 4. For a given mesh element type, it offers the following services:

1. returning the number of oriented reference spaces associated to this element type, and the reference space tag associated to a given mesh element;
2. returning the local vertex indices of the edges and faces for each possible oriented reference space;

[9] In the Gmsh library, each element type is given a unique positive integer value: `2` for lines, `3` for triangles, `4` for quadrangles, `5` for tetrahedra, `6` for pyramids, `7` for prisms and `8` for hexahedra.

[10] In this context, *vector* must be understood as a data structure [124], rather than a mathematical object.

3. returning the vector $\mathbf{r}$, mapping a vertex for the (u, v) space to the (a, b) space (see section 4.2.2), associated to a given mesh element;
4. computing the mapping of a point, inside a given mesh element, in the (x, y), (u, v) or (a, b) space into another space (see section 4.2.2);
5. computing the Jacobian matrix (and its determinant) of the mapping between the oriented reference space, of a given element, and its physical space.

As stated earlier, `ReferenceSpace` is an interface, and it depends on a mesh element type. An actual `ReferenceSpace` class implements then the above services for a particular element type. For instance `ReferenceSpaceQuad` implements a `ReferenceSpace` for quadrangular elements. The naming convention starts with `ReferenceSpace` followed by the element type (`Line`, `Tri`, `Quad`, `Tet`, `Pri`, `Pyr` or `Hex`)[11].

By the nature of a `ReferenceSpace`, it is sufficient to instantiate it only once for a given element type. Furthermore, it worth noticing that, since many element types can exist in a mesh, many `ReferenceSpace`s are needed. To handle this, the `ReferenceSpaceManager` class can be used. It is basically a factory and a container of `ReferenceSpace`s, and it offers the same services. Thus, for a given mesh element, it will determine on-the-fly its type, and sent the requested service to the appropriate `ReferenceSpace`. A `ReferenceSpaceManager` offers only static methods, and does not need to be instantiated.

As explained in chapter 4, oriented spaces are generated and handled using a permutation tree structure. This is implemented by the `PermutationTree` class.

Finally, let us note that it makes sens to offer those mappings and spaces services in an independent class, rather than in the class representing a mesh element itself. Indeed, this class usually comes from a mesh library (Gmsh in our case), and already represents the mesh element in a reference space (see section 2.3): the (u, v) space defined in section 4.2.2. Thus, it makes sens for the mesh element class to handle the different spaces needed for representing a *mesh element*, and to use another class to handle the *finite element* spaces.

A.3.2 Finite element bases

Now that we have everything we need to manage the finite element spaces, we can define the basis functions on every oriented reference space. This concept is represented by the `Basis` interface, and it basically offers the possibility to evaluate the basis functions (or their derivative) at a given set of points (in the oriented reference space) for a given mesh element or orientation tag.

As explained in section 2.2, it is a common choice to construct the basis functions with a local support. However, in some cases, it can be useful to use basis functions with a broader support, for instance to couple finite element formulations and circuit equations [42]. For this reason, an additional inheritance level has been introduced: the `BasisLocal` interface. This basically guarantees that, a classes implementing this interface is indeed representing a `Basis` with a local support, and defined on an oriented reference space. This has been introduce to enable global basis functions in future versions.

In order to handle basis functions spanning a finite-dimensional subset of H^1 or $H(\mathbf{curl})$,

[11] For line, triangle, quadrangle, tetrahedron, prism, pyramid and hexahedron.

two additional interfaces, inheriting from `BasisLocal`, have been introduced. A class representing an H^1 basis must implement the `BasisHierarchical0Form` interface, and a class representing an $H(\mathbf{curl})$ basis must implement the `BasisHierarchical1Form`[12] interface. As the names also suggest, classes inheriting from one of these interfaces must implement a hierarchical basis[13] (*e.g.*, as proposed by [136]).

The classes inheriting from these last interfaces are actually constructing the basis functions. The naming convention starts with `Basis`, then the element type, and finally the basis type (`0Form` for an H^1 basis, and `1Form` for an $H(\mathbf{curl})$ basis). Thus, for instance, `BasisQuad0Form` implements a hierarchical $H(\mathbf{curl})$ basis for quadrangular elements. In the actual implementation, H^1 bases are available for line, triangular, quadrangular, tetrahedral and hexahedral elements; and $H(\mathbf{curl})$ bases are available for line, triangular, quadrangular and tetrahedral elements.

To construct a `Basis` for every possible oriented reference space, the `ReferenceSpace` (associated to the considered element type) is used. Moreover, since (Legendre) polynomials are involved, a `Polynomial` and a `Legendre` classes have been developed to handle those aspects. Finally, in order to easily instantiate a `Basis`, the `BasisGenerator` class offers a set of static methods for instantiating these classes.

A.3.3 Function spaces

We now have everything we need to construct an actual function space, which is represented by the `FunctionSpace` interface. Basically, a `FunctionSpace` offers two services:

1. it can generate the degrees of freedoms (represented by the `Dof` class, as exposed in section A.5.1) associated to a given mesh element[14];
2. being given
   (a) a mesh element,
   (b) a set of coefficients a_i (associated to the degrees of freedom of the given mesh element),
   (c) a point (inside the element) in the physical space,

   a `FunctionSpace` can compute (in the physical space) the linear combination of the basis functions (or their derivative), associated to the mesh element, weighted by the coefficients a_i.

Regarding the last service, for a mesh element Ω_e and a $H^1(\Omega_e)$ basis, we formally have:

$$p(\mathbf{x}) = \sum_{i=0}^{N-1} a_i \, v^i(\mathbf{x}) \qquad v^i(\mathbf{x}) \in H^1(\Omega_e),$$

where $\mathbf{x}$ is a point inside Ω_e, $v^i(\mathbf{x})$ the basis function associated to the i^{th} (among N) degree of freedom of Ω_e, and where $p(\mathbf{x})$ is the result of the requested linear combination. Similarly,

[12] The 0-form and 1-form (or, more generally, k-form) notions come from the theory of differential forms, which offers a mathematical environment to treat multivariable calculus, independently of a coordinate system [10, 25, 121]; in our context, this naming convention allows us to classify basis functions, that are spanning a function space subset of: H^1 (0-forms), $H(\mathbf{curl})$ (1-forms) or $H(\text{div})$ (2-forms).

[13] We also introduced a `BasisLagrange` interface, to handle Lagrange basis, for testing purposes.

[14] In section 2.7, this feature was implemented by the `takeDof` function.

for an $H(\mathbf{curl}, \Omega_e)$ basis, we have:

$$\boldsymbol{p}(\mathbf{x}) = \sum_{i=0}^{N-1} a_i \, \boldsymbol{v}^i(\mathbf{x}) \qquad \boldsymbol{v}^i(\mathbf{x}) \in H(\mathbf{curl}, \Omega_e).$$

This can also be done with the derivatives of $v^i(\mathbf{x})$ or $\boldsymbol{v}^i(\mathbf{x})$.

Clearly, a `FunctionSpace` relies on one (or more) `Basis` to offer its services. It is worth noticing that by contrast to a `Basis`, which is defined on a *reference space*, a `FunctionSpace` can represent the basis functions in the *physical space* (or, at least, can compute their linear combination in the physical space). Thus, if we consider every mesh element of the domain Ω, the corresponding finite-dimensional function space $V^1(\Omega)$ (or $V(\mathbf{curl}, \Omega)$)[15] can be constructed.

As already stated, `FunctionSpace` is just an interface. Depending whether the function space is a subset of $H^1(\Omega)$ or $H(\mathbf{curl}, \Omega)$, one of the following class can be used:

1. `FunctionSpace0Form` for a finite-dimensional subset of $H^1(\Omega)$;
2. `FunctionSpace1Form` for a finite-dimensional subset of $H(\mathbf{curl}, \Omega)$.

Obviously, these two classes implement `FunctionSpace`.

A.4 The `formulation` module

The `formulation` module offers tools for constructing and combining FE elementary integrals, so that they can be eventually assembled in the finite element linear system matrix (or right hand side).

A.4.1 Finite element formulations

The first abstraction of the `formulation` module is the `Formulation` interface. It offers only two services:

1. telling whether this `Formulation` is a `FormulationBlock` or a `FormulationCoupled` (concepts that will be developed later in this text);
2. updating this `Formulation`.

Regarding the first service, we first need to define the `FormulationBlock` interface and the `FormulationCoupled` interface, both inheriting from `Formulation`. The `FormulationBlock` interface is a slightly more evolved `FeTerm` class, as introduced in section 2.7[16], and it offers the following services:

1. it can compute the elementary integral associated to a given mesh element, and a pair of its degrees of freedom (in the element local ordering), exactly as the `FeTerm::get` method does in section 2.7;

[15] Subset of $H^1(\Omega)$ (or $H(\mathbf{curl}, \Omega)$).

[16] Actually, the `FeTerm` class has never been implemented, and was used in this work as a pedagogical tool; in the developed code, the interface representing the actual `FeTerm` is the `FormulationBlock`; as explained in the text, it slightly extends the services offered by `FeTerm`

2. it can also compute the elementary integral associated to a single degree of freedom (in the element local ordering), that is a right hand side contribution in a finite element problem;
3. it can return the `GroupOfElement` (given at construction time) on which these integrals were defined;
4. it can return the `FunctionSpace` (given at construction time) used for the test functions;
5. it can also return the `FunctionSpace` (given at construction time) used for representing the unknown field.

In finite element problems, it is sometimes needed to define auxiliary problems, such as in the discretization of the Padé-localized square-root transmission condition in section 6.5. In other words, it can be necessary to assemble in the same linear system, finite element terms defined on different function spaces (for both the unknown field and the test functions). For this reason the `FormulationCoupled` interface is introduced. It is basically a container for `FormulationBlock`s (each of them being aware of their own `FunctionSpace`s). The only service a `FormulationCoupled` offers (in addition to the services of a `Formulation`), is to return a list with the `FormulationBlock`s it contains.

Let us now focus on the last service of a `Formulation`: the update. During the lifetime of an elementary finite element integral, its parameters can change: for instance, in a domain decomposition context, the right hand side of (6.39) changes from iteration to iteration. To account this, the update service is offered. For more flexibility, the update method does not take any argument. To control the update, an appropriate structure (taken from the `context` modules, or written from scratch) must be passed at the construction time of the `Formulation` object, so that the new data are available at the update time.

It is worth noticing, that `FormulationBlock` and `FormulationCoupled` are only interfaces. Thus, an actual `Formulation` must implement all the inherited services. For instance, `FormulationOSRCScalar` is a `FormulationCoupled`, implementing the FE formulation of the scalar Padé-localized square-root transmission condition (see section 6.5.1); and `FormulationSteadyWave` is a `FormulationBlock`, implementing the FE formulation of time-harmonic wave problems.

A.4.2 Finite element elementary terms

As detailed in chapter 3, an efficient FE system assembly can be obtained, by computing all the FE elementary integrals with a matrix-matrix product. This approach is made possible by the `Term` interface. It offers a single service: returning the FE elementary integral associated to a given mesh element, and a pair of its degrees of freedom (in the element local ordering); in the case of a right hand side term, one of the pair member is set to zero. It is worth recalling that in section 3.5, in order to implement the efficient assembly approach, we required that the elements have to be sorted with respect to their orientations, which is the case with `GroupOfElement` objects.

Classes inheriting the `Term` interface must implement a particular finite element term. In the developed code, an implementation for each of the five cases treated in section 3.3 is provided: `TermFieldField` (product of H^1 functions), `TermGradGrad` (product of $H(\mathbf{curl})$

functions), **TermCurlCurl** (product of $H(\text{div})$ functions), **TermProjectionField** (product of a known function by an H^1 function) and **TermProjectionGrad** (product of a known function by an $H(\mathbf{curl})$ function).

Obviously, these **Term**s can be used to implement a particular **Formulation**. In this case, the appropriate **Term**s will be instantiated at construction time. Finally, let us note that the quadrature laws needed to compute the elementary integrals (in a **Term** or not) are available in the **Quadrature** class, which is nothing but a wrapper for the quadrature laws provided by the Gmsh library.

A.5 The assembler module

As its name suggests, the **assembler** module is responsible of the finite element assembly.

A.5.1 Degrees of freedom

In the actual implementation, a degree of freedom is represented by a **Dof** object.Basically, a **Dof** is defined by a pair of integers, and it offers the following services:

1. returning the pair of integers defining a **Dof** object;
2. modifying the pair of integers defining a **Dof** object;
3. comparison (larger, equal, ...) with another **Dof** object.

Regarding the last service, two **Dof** objects are equal, if and only if they are defined with the same pair.

As explained previously, it is the **FunctionSpace** that will generate the degrees of freedom associated to every mesh element. Thus, in normal conditions, it is a **FunctionSpace** that will determine the pair of integers of its **Dof**s.

A.5.2 Mapping degrees of freedom

In section 2.7, we saw that the assembly procedure relies on a **getGlobalId** function. As a recall, it is responsible to associate a unique positive integer value to each DoF of the finite element system. This value is then used as an entry in the finite element matrix (or right hand side). In the developed code, this mapping between a degree of freedom and an integer value is the responsibility of the **DofManager** class.

A **DofManager** proceeds in two steps. Once instantiated, it is possible to add **Dof** objects in a **DofManager**. It is worth noticing that, if equal **Dof**s are inserted, only one instance is kept in the **DofManager**. During this first step, some **Dof**s can be marked as *fixed*. These degrees of freedom will not be associated to a unique integer value, however, it is possible to associate them with a scalar value. This mechanism is useful to impose Dirichlet boundary conditions, since the degrees of freedom associate to the boundary are no longer free (their value being imposed). Therefore, they are not associated with an entry in the linear system. Nevertheless, it is needed to store (and access) their imposed value: this is also handled by a **DofManager**.

In a second step, it is possible to request a **DofManager** to number all its non-fixed **Dof**s. This numbering starts with `0`, and ends with `N-1`, `N` being the total number of non-fixed **Dof**s. Once this numbering done, it becomes impossible to add new **Dof** objects in the considered **DofManager**. On the other hand, it is now possible, for a given **Dof**, to recover its associated integer.

It is worth noticing that, thanks to the comparison properties of **Dof** objects, efficient data structures can be used in the **DofManager** implementation.

A.5.3 Finite element system

It is now possible to assemble the system, associated to a finite element problem. In the developed framework, a finite element system must implement the **SystemAbstract** class. This system is defined thanks to a list of **Formulation**s, defining the elementary integrals to insert in the system matrix (or right hand side). If more than one elementary integral is inserted in a given position in the system (or right hand side), its contribution is added to the current value. A **SystemAbstract** offers the following services:

1. **Formulation**s can be inserted into a **SystemAbstract**;
2. **Dof**s of the finite element problem can be fixed to a given value (Dirichlet condition)[17];
3. the system, defined by the **Formulation**s and the fixed **Dof**s, can be assembled;
4. the assembled system can be solved;
5. the solution of the system can be extracted.

Obviously, a **SystemAbstract** object has to extract the **Dof**s involved in the system it defines. This is possible, since a **Formulation** knows its domain of definition (**GroupOfElement**) and its **FunctionSpace**s (for the unknown field and the test functions). Once all these **Dof**s extracted, a **SystemAbstract** can use a **DofManager** to associate to all these **Dof**s a unique number between `0` and `N-1` (`N` being the total number of non-fixed **Dof**s). This **DofManager** can also be used to fix some of these **Dof**s. Let us also note the **SystemHelper** class, which proposes a set of static methods to simplify the imposition of Dirichlet conditions. Finally, with the **FunctionSpace**s of each **Formulation** and the **DofManager**, the assemble procedure, as proposed in section 2.7, can be implemented.

In the developed code, three classes implementing a **SystemAbstract** are proposed: **System**, **SystemEigen** and **SystemPETSc**. A **System** and a **SystemPETSc** are both handling linear system. The unique difference is the solver used: the first uses MUMPS and the latter uses the GMRES of the PETSc library. A **SystemEigen** is handling eigenvalue problems thanks to the SLEPc library.

A.6 The `solver` module

The `solver` module offers bindings to external solvers (PETSc, SLEPc and MUMPS), as well as the home made DDM solver: **SolverDDM**. This latter implements the non-overlapping

[17] Therefore, strictly speaking, they are not degrees of freedom any more, and will be thus not assemble in the finite element system.

optimized Schwarz algorithm, as presented in chapter 6. To handle this task, it relies on the GMRES solver of the PETSc library to solve the DDM system (6.33):

$$(\mathcal{I} - \mathcal{A})\boldsymbol{g} = \boldsymbol{b}.$$

To implement this, the **SolverDDM** class exploits the **MATSHELL** matrix type of PETSc[18] library. Moreover, as we saw in section 6.4, applying the operator $\mathcal{A}$ to some vector, amounts to solve linear systems: this task is handled by **System** objects.

A.7 Other modules

In the previous sections, we presented the five most important modules. The remaining units are not part of the core system, and handle less important tasks. For this reason, only a brief overview is provided hereafter:

1. the `postpro` module offers a set of classes, for exporting finite element solutions to Gmsh post-processing views (**PView**);
2. the `context` module offers containers for updating **Formulation**s;
3. the `common` module offers some useful structures (*e.g.*, timers, exceptions, dots).

A.8 Example

Let us now present a working code, exploiting the developed library. Algorithm 10 presents a **main** function, solving an electromagnetic metallic waveguide problem. The waveguide is assumed to be infinite, and is thus truncated by a Silver-Müller radiation condition.

```
int main(void)
  Description
    Main function simulating a waveguide.

  Implementation (C++)
    /* Load Mesh                                                          */
    Mesh msh("mesh.msh");                                  // Load mesh

    /* Extract geometrical regions:                                       */
    /*  - infinity boundary, on which the radiation condition is imposed; */
    /*  - source boundary, on which the source signal is imposed;         */
    /*  - wall boundary, which is the metallic part of the waveguide;     */
    /*  - volume domain, which is the inner computational domain.         */
    GroupOfElement infinity(msh.getFromPhysical(4));          // Physical 4
    GroupOfElement source(msh.getFromPhysical(5));            // Physical 5
    GroupOfElement wall(msh.getFromPhysical(6));              // Physical 6
    GroupOfElement volume(msh.getFromPhysical(7));            // Physical 7
```

[18] See `http://www.mcs.anl.gov/petsc/petsc-3.6/docs/manualpages/Mat/MATSHELL.html`.

```
/* Function space:                                                    */
/*  - first define its domain of definition using a std::vector;      */
/*  - second define its finite element order.                         */
std::vector<const GroupOfElement *> domain(4);
domain[0] = &volume;
domain[1] = &source;
domain[2] = &wall;
domain[3] = &infinity;

int order = 3;                                  // Function space of order 3
FunctionSpace1Form fs(domain, order);

/* Formulations:                                                      */
/*  - wave is the Formulation for Maxwell's equations;                */
/*  - radiation is the Formulation for Silver-Müller condition;       */
int k = 20;                                                  // Wavenumber
FormulationSteadyWave<Complex> wave(volume, fs, k);
FormulationSilverMuller radiation(infinity, fs, k);      //  TE signal

/* System:                                                            */
/*  - first add Formulations;                                         */
/*  - then impose Dirichlet conditions.                               */
/* Function fZero(x,y,z) returns the three-dimensional vector 0,      */
/* and function fSource(x,y,z) returns a TE polarized source.         */
System<Complex> system;
system.addFormulation(wave);
system.addFormulation(radiation);

SystemHelper<Complex>::dirichlet(system, fs, wall, fZero);
SystemHelper<Complex>::dirichlet(system, fs, source, fSource);

/* Assemble, solve and write solution                                 */
system.assemble();
system.solve();

FEMSolution<Complex> solution;                    // From postpro module
system.getSolution(solution, fs, volume);
solution.write("solution");

return 0;
```

Algorithm 10: Main function exploiting the developed library.

A.9 Convergence tests

In this final section, the developed finite element framework is validated. To meet this objective, the L^2 projection of a known function is performed on a domain Ω. For scalar-valued case, the following weak problem is defined:

$$\begin{aligned} &\text{Find } v \in H^1(\Omega) \text{ such that, for every } v' \in H^1(\Omega) : \\ &\int_\Omega v \cdot v' \,\mathrm{d}\Omega = \int_\Omega f \cdot v' \,\mathrm{d}\Omega, \end{aligned} \tag{A.1}$$

where f is the known (scalar) function to project on Ω. In a vector-valued situation, this problem writes:

$$\begin{aligned} &\text{Find } \boldsymbol{v} \in H(\mathbf{curl}, \Omega) \text{ such that, for every } \boldsymbol{v'} \in H(\mathbf{curl}, \Omega) : \\ &\int_\Omega \boldsymbol{v} \cdot \boldsymbol{v'} \,\mathrm{d}\Omega = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v'} \,\mathrm{d}\Omega, \end{aligned} \tag{A.2}$$

where $\boldsymbol{f}$ is the known (vector) function to project on Ω.

The domain Ω is meshed with a given element type, and these two problems are solved by using the finite element method. In order to validate the implementation, the relative error between the known function and the FE solution is computed for different mesh densities and FE discretization orders. According to the theory, the slope of the error with respect to the mesh size shall be h^{p+1}, where h is the mesh size and p the FE discretization order.

In the following, the relative error is plotted for different element types. For the scalar case, this error is defined as:

$$e_S = \sqrt{\frac{\int_\Omega (f - v)^2 \,\mathrm{d}\Omega}{\int_\Omega f^2 \,\mathrm{d}\Omega}}, \tag{A.3}$$

where e_S is the error, f the known function and v the FE approximation. For the vector case, this error is defined as:

$$e_V = \sqrt{\sum_{i=0}^{D-1} e_i^2}, \tag{A.4}$$

where e_V is the error, D the dimentionality of the problem, and e_i the error computed by applying (A.3) on the i^{th} component.

The considered case are reported on Table A.1. Let us note that, since we did not implement a curl-confirming basis for hexahedral elements, only the scalar case is considered for this geometrical type. The structure of the coming subsections is the following. We first present the error plot for different finite element discretizations and mesh densities. Then, we show the convergence rates measured on the basis of the two finest meshes available. And finally, we conclude by assessing if the expected rate of h^{p+1} is obtained.

Domain	Mesh element type	Projected function
1D-Line	Line	$f(x) = \sin(10\,x)$ $\boldsymbol{f}(x) = [\sin(10\,x), 0, 0]^T$
2D-Square	Triangle	$f(x,y) = \sin(10\,x) + \sin(10\,y)$ $\boldsymbol{f}(x,y) = [\sin(10\,x), \sin(10\,y), 0]^T$
	Quadrangle	$f(x,y) = \sin(10\,x) + \sin(10\,y)$ $\boldsymbol{f}(x,y) = [\sin(10\,x), \sin(10\,y), 0]^T$
3D-Cube	Tetrahedron	$f(x,y,z) = \sin(10\,x) + \sin(10\,y) + \sin(10\,z)$ $\boldsymbol{f}(x,y,z) = [\sin(10\,x), \sin(10\,y), \sin(10\,z)]^T$
	Hexahedron	$f(x,y,z) = \sin(10\,x) + \sin(10\,y) + \sin(10\,z)$

Table A.1: Validation tests.

A.9.1 Line finite elements

Scalar case

Let us start with the scalar line case. The relative error between the known function and the FE solution is reported on Figure A.1. The convergence rates are available on Table A.2. Based on these results, we can directly notice that the expected convergence rate is found.

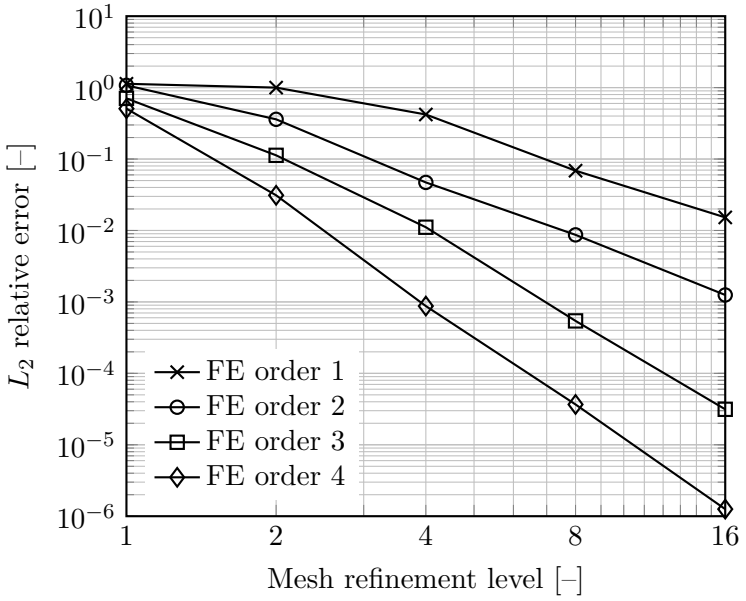


Figure A.1: Validation test: scalar line finite element (error plot).

FE order	Measured convergence rate	Expected convergence rate
1	2.2	2
2	2.8	3
3	4.1	4
4	4.9	5

Table A.2: Validation test: scalar line finite element (convergence rates).

Vector case

Now, let us consider the vector case. The relative error between the known function and the FE solution is reported on Figure A.2. The convergence rates are available on Table A.3. Based on these results, we can directly notice that the expected convergence rate is found.

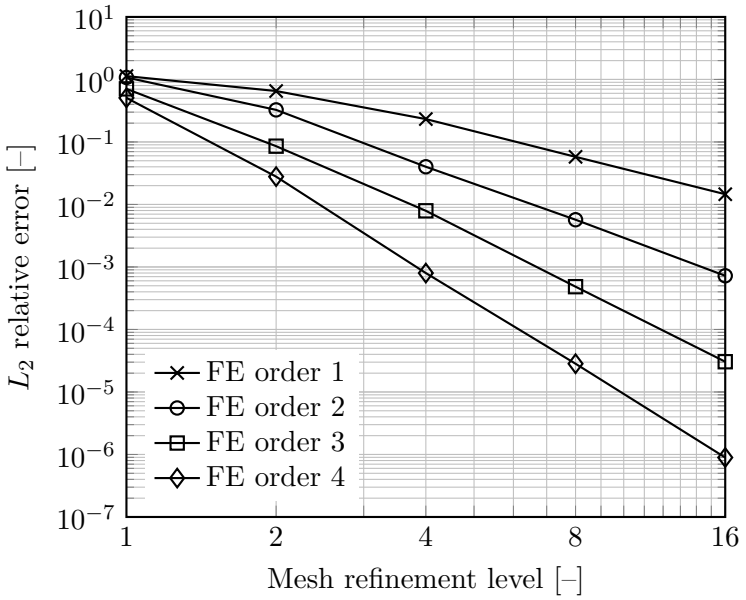


Figure A.2: Validation test: vector line finite element (error plot).

FE order	Measured convergence rate	Expected convergence rate
1	2.0	2
2	3.0	3
3	4.0	4
4	5.0	5

Table A.3: Validation test: vector line finite element (convergence rates).

A.9.2 Triangular finite elements

Scalar case

We now consider the scalar triangular case. The relative error between the known function and the FE solution is reported on Figure A.3. The convergence rates are available on Table A.4. Based on these results, we can directly notice that the expected convergence rate is found.

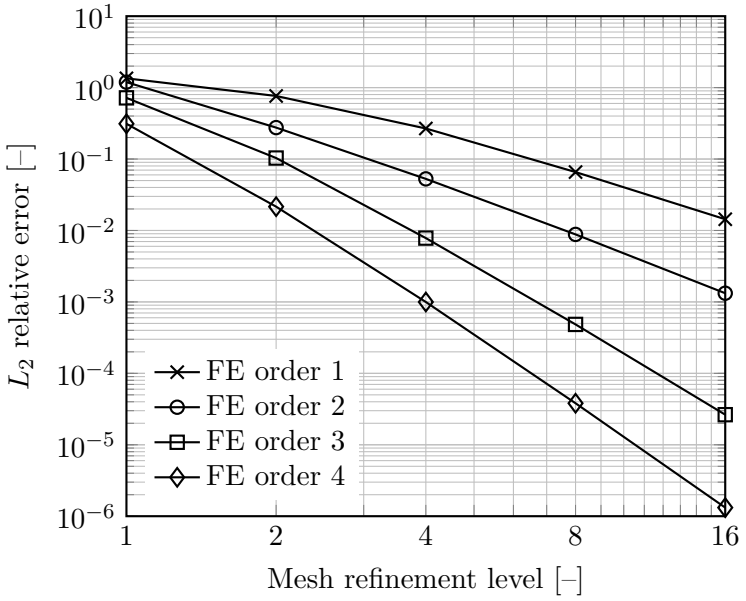


Figure A.3: Validation test: scalar triangular finite element (error plot).

FE order	Measured convergence rate	Expected convergence rate
1	2.2	2
2	2.7	3
3	4.2	4
4	4.9	5

Table A.4: Validation test: scalar triangular finite element (convergence rates).

Vector case

For the vector case, the relative error between the known function and the FE solution is reported on Figure A.4. The convergence rates are available on Table A.5. Based on these results, we can directly notice that the expected convergence rate is found.

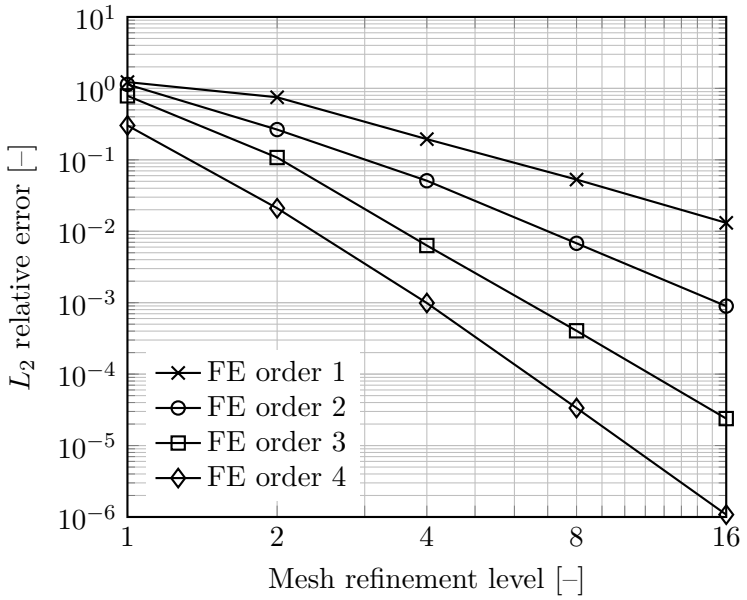


Figure A.4: Validation test: vector triangular finite element (error plot).

FE order	Measured convergence rate	Expected convergence rate
1	2.0	2
2	3.0	3
3	4.1	4
4	5.0	5

Table A.5: Validation test: vector triangular finite element (convergence rates).

A.9.3 Quadrangular finite elements

Scalar case

We now consider the scalar quadrangular case. The relative error between the known function and the FE solution is reported on Figure A.5. The convergence rates are available on Table A.6. Based on these results, we can directly notice that the expected convergence rate is found.

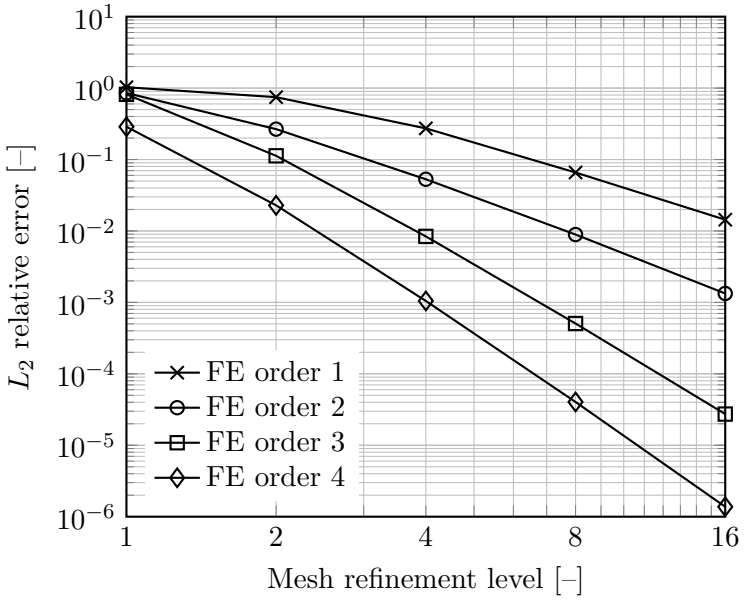


Figure A.5: Validation test: scalar quadrangular finite element (error plot).

FE order	Measured convergence rate	Expected convergence rate
1	2.2	2
2	2.7	3
3	4.2	4
4	4.9	5

Table A.6: Validation test: scalar quadrangular finite element (convergence rates).

Vector case

For the vector case, the relative error between the known function and the FE solution is reported on Figure A.6. The convergence rates are available on Table A.7. Based on these results, we can directly notice that the expected convergence rate is found.

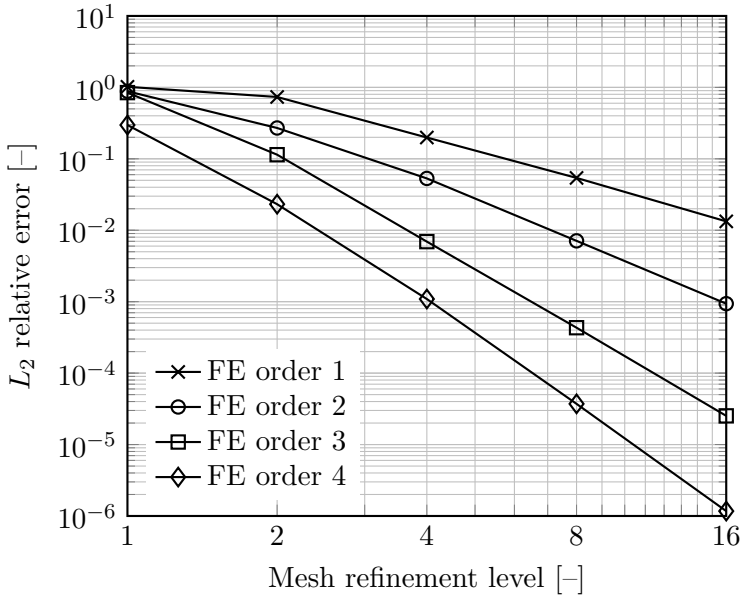


Figure A.6: Validation test: vector quadrangular finite element (error plot).

FE order	Measured convergence rate	Expected convergence rate
1	2.0	2
2	2.9	3
3	4.1	4
4	5.0	5

Table A.7: Validation test: vector quadrangular finite element (convergence rates).

A.9.4 Tetrahedral finite elements

Scalar case

We now consider the scalar tetrahedral case. The relative error between the known function and the FE solution is reported on Figure A.7. The convergence rates are available on Table A.8. Based on these results, we can directly notice that the expected convergence rate is found.

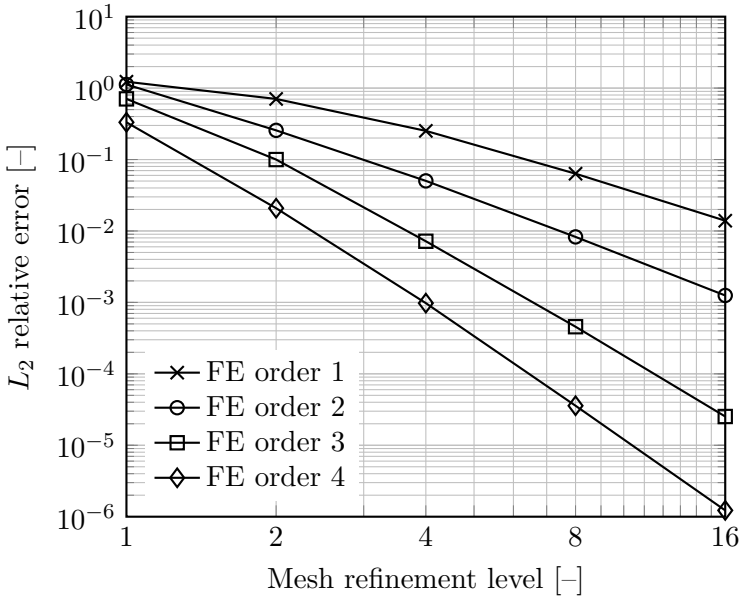


Figure A.7: Validation test: scalar tetrahedral finite element (error plot).

FE order	Measured convergence rate	Expected convergence rate
1	2.2	2
2	2.7	3
3	4.2	4
4	4.9	5

Table A.8: Validation test: scalar tetrahedral finite element (convergence rates).

Vector case

For the vector case, the relative error between the known function and the FE solution is reported on Figure A.8. The convergence rates are available on Table A.9. Based on these results, we can directly notice that the expected convergence rate is found.

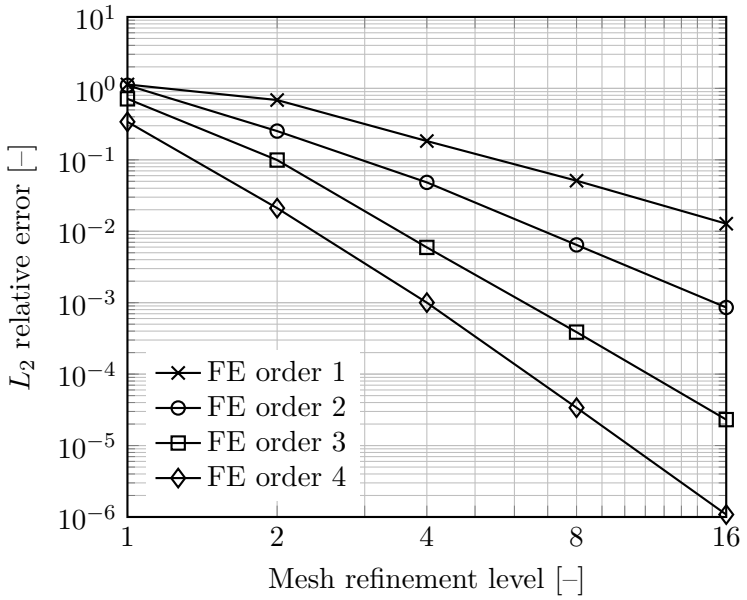


Figure A.8: Validation test: vector tetrahedral finite element (error plot).

FE order	Measured convergence rate	Expected convergence rate
1	2.0	2
2	2.9	3
3	4.1	4
4	5.0	5

Table A.9: Validation test: vector tetrahedral finite element (convergence rates).

A.9.5 Hexahedral finite elements

Scalar case

Finally, let us consider the scalar hexahedral case. The relative error between the known function and the FE solution is reported on Figure A.9. The convergence rates are available on Table A.10. Based on these results, we can directly notice that the expected convergence rate is found.

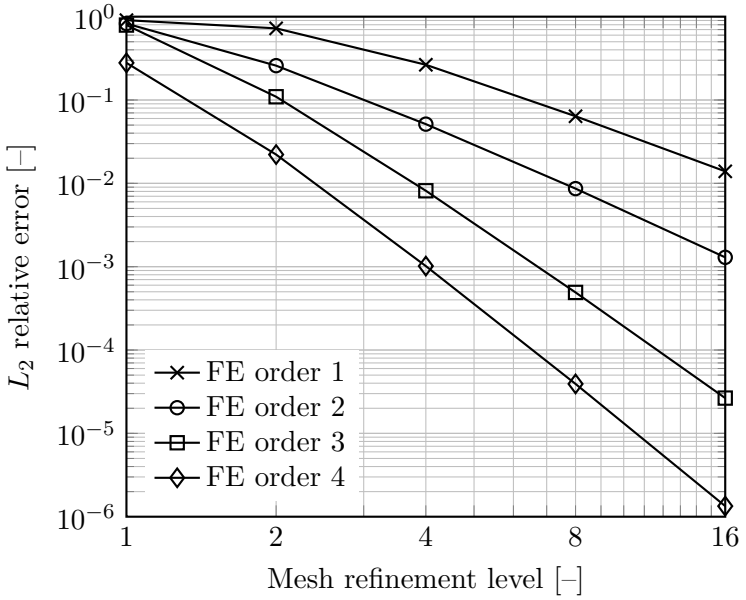


Figure A.9: Validation test: scalar hexahedral finite element (error plot).

FE order	Measured convergence rate	Expected convergence rate
1	2.2	2
2	2.7	3
3	4.2	4
4	4.9	5

Table A.10: Validation test: scalar hexahedral finite element (convergence rates).

Bibliography

[1] M. Ainsworth and J. Coyle, "Hierarchic finite element bases on unstructured tetrahedral meshes," *International Journal for Numerical Methods in Engineering*, vol. 58, no. 14, pp. 2103–2130, 2003.

[2] P. Alotto, A. Bertoni, I. Perugia, and D. Schotzau, "Efficient use of the local discontinuous Galerkin method for meshes sliding on a circular boundary," *IEEE Transactions on Magnetics*, vol. 38, no. 2, pp. 405–408, 2002.

[3] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001.

[4] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, "Hybrid scheduling for the parallel solution of linear systems," *Parallel Computing*, vol. 32, no. 2, pp. 136–156, 2006.

[5] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. Society for Industrial and Applied Mathematics, 1999.

[6] J. D. Anderson Jr, *A history of aerodynamics and its impact on flying machines*. Cambridge University Press, 1999.

[7] ——, *Fundamentals of aerodynamics*, 5th ed. McGraw-Hill, 2011.

[8] W. E. Arnoldi, "The principle of minimized iterations in the solution of the matrix eigenvalue problem," *Quarterly of Applied Mathematics*, vol. 9, no. 1, pp. 17–29, 1951.

[9] J. W. Arthur, "The evolution of Maxwell's equations from 1862 to the present day," *IEEE Antennas and Propagation Magazine*, vol. 55, no. 3, pp. 61–81, 2013.

[10] D. Bachman, *A Geometric Approach to Differential Forms*. Birkhäuser Basel, 2006.

[11] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, S. Zampini, and H. Zhang, "PETSc users manual," Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 3.6, 2015.

[12] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object oriented numerical software libraries," in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds., Birkhäuser Press, 1997, pp. 163–202.

[13] A. D. Baxevanis and B. F. F. Ouellette, *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*, 3rd ed. Wiley, 2004.

[14] A. Bendali and Y. Boubendir, "Non-overlapping domain decomposition method for a nodal finite element method," *Numerische Mathematik*, vol. 103, no. 4, pp. 515–537, 2006.

[15] J.-P. Bérenger, "A perfectly matched layer for the absorption of electromagnetic waves," *Journal of Computational Physics*, vol. 114, no. 2, pp. 185–200, 1994.

[16] ——, "Three-dimensional perfectly matched layer for the absorption of electromagnetic waves," *Journal of Computational Physics*, vol. 127, no. 2, pp. 363–379, 1996.

[17] A. Bermúdez, L. Hervella-Nieto, A. Prieto, and R. Rodríguez, "An exact bounded PML for the Helmholtz equation," *Comptes Rendus Mathematique*, vol. 339, no. 11, pp. 803–808, 2004.

[18] L. S. Blackford, J. Demmel, J. J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, "An updated set of Basic Linear Algebra subprograms (BLAS)," *ACM Transactions on Mathematical Software*, vol. 28, no. 2, pp. 135–151, 2002.

[19] A. Bossavit, *Computational Electromagnetism: Variational Formulations, Complementarity, Edge Elements*. Academic Press, 1998.

[20] Y. Boubendir, "An analysis of the BEM-FEM non-overlapping domain decomposition method for a scattering problem," *Journal of Computational and Applied Mathematics*, vol. 204, no. 2, pp. 282–291, 2007.

[21] Y. Boubendir, X. Antoine, and C. Geuzaine, "Quasi-optimal non-overlapping domain decomposition algorithm for the Helmholtz equation," *Journal of Computational Physics*, vol. 231, no. 2, pp. 262–280, 2012.

[22] Y. Boubendir, A. Bendali, and M. B. Fares, "Coupling of a non-overlapping domain decomposition method for a nodal finite element method with a boundary element method," *International Journal for Numerical Methods in Engineering*, vol. 73, no. 11, pp. 1624–1650, 2008.

[23] R. M. Bozorth, *Ferromagnetism*. Wiley-IEEE Press, 1993.

[24] D. S. Britt, S. V. Tsynkov, and E. Turkel, "A high-order numerical method for the Helmholtz equation with nonstandard boundary conditions," *SIAM Journal on Scientific Computing*, vol. 35, no. 5, A2255–A2292, 2013.

[25] H. Cartan, *Formes differentielles*. Hermann, 1967.

[26] C. Chevalier and F. Pellegrini, "PT-Scotch: A tool for efficient parallel graph ordering," *Parallel Computing*, vol. 34, no. 6-8, pp. 318–331, 2008.

[27] P. G. Ciarlet, *The finite element method for elliptic problems*. North-Holland Publishing, 1978.

[28] L. Conen, V. Dolean, R. Krause, and F. Nataf, "A coarse space for heterogeneous Helmholtz problems based on the Dirichlet-to-Neumann operator," *Journal of Computational and Applied Mathematics*, vol. 271, pp. 83–99, 2014.

[29] C. Dawson and J. Proft, "Coupling of continuous and discontinuous galerkin methods for transport problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, no. 29-30, pp. 3213–3231, 2002.

[30] V. Debierre, G. Demésy, T. Durt, A. Nicolet, B. Vial, and F. Zolla, "Absorption in quantum electrodynamic cavities in terms of a quantum jump operator," *Physical Review A*, vol. 90, no. 3, 033806, 2014.

[31] L. Demkowicz, P. Monk, L. Vardapetyan, and W. Rachowicz, "de Rham diagram for hp finite element spaces," *Computers & Mathematics with Applications*, vol. 39, no. 7-8, pp. 29–38, 2000.

[32] B. Després, "Méthodes de décomposition de domaine pour les problèmes de propagation d'ondes en régime harmonique. le théorème de Borg pour l'equation de Hill vectorielle," PhD thesis, Université de Paris IX, France, 1991.

[33] B. Després, P. Joly, and J. E. Roberts, "A domain decomposition method for the harmonic Maxwell equations," in *Iterative methods in linear algebra (Brussels, 1991)*, North-Holland, 1992, pp. 475–484.

[34] V. Dolean, M. J. Gander, and L. Gerardo-Giorda, "Optimized Schwarz methods for Maxwell's equations," *SIAM Journal on Scientific Computing*, vol. 31, no. 3, pp. 2193–2213, 2009.

[35] V. Dolean, M. J. Gander, S. Lanteri, J.-F. Lee, and Z. Peng, "Effective transmission conditions for domain decomposition methods applied to the time-harmonic curl-curl Maxwell's equations," *Journal of Computational Physics*, vol. 280, pp. 232–247, 2015.

[36] V. Dolean, P. Jolivet, and F. Nataf, "An introduction to domain decomposition methods: Algorithms, theory and parallel implementation," Lecture, France, 2015, [Online]. Available: `https://hal.archives-ouvertes.fr/cel-01100932`.

[37] B. Donderici and F. L. Teixeira, "Conformal perfectly matched layer for the mixed finite element time-domain method," *IEEE Transactions on Antennas and Propagation*, vol. 56, no. 4, pp. 1017–1026, 2008.

[38] J. J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, "A set of level 3 Basic Linear Algebra Subprograms," *ACM Transactions on Mathematical Software*, vol. 16, no. 1, pp. 1–17, 1990.

[39] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, "An extended set of FORTRAN Basic Linear Algebra Subprograms," *ACM Transactions on Mathematical Software*, vol. 14, no. 1, pp. 1–17, 1988.

[40] U. Drepper, *What every programmer should know about memory*, 2007. [Online]. Available: `http://people.redhat.com/drepper/cpumemory.pdf`.

[41] P. Dular, C. Geuzaine, F. Henrotte, and W. Legros, "A general environment for the treatment of discrete problems and its application to the finite element method," *IEEE Transactions on Magnetics*, vol. 34, no. 5, pp. 3395–3398, 1998.

[42] P. Dular, C. Geuzaine, and W. Legros, "A natural method for coupling magnetodynamic H-formulations and circuit equations," *IEEE Transactions on Magnetics*, vol. 35, no. 3, pp. 1626–1629, 1999.

[43] V. Eijkhout, E. Chow, and R. Van de Geijn, *Introduction to high performance scientific computing*, 2015. [Online]. Available: `http://pages.tacc.utexas.edu/~eijkhout/istc/istc.html`.

[44] M. El Bouajaji, X. Antoine, and C. Geuzaine, "Approximate local magnetic-to-electric surface operators for time-harmonic Maxwell's equations," *Journal of Computational Physics*, vol. 279, pp. 241–260, 2014.

[45] M. El Bouajaji, B. Thierry, X. Antoine, and C. Geuzaine, "A quasi-optimal domain decomposition algorithm for the time-harmonic Maxwell's equations," *Journal of Computational Physics*, vol. 294, pp. 38–57, 2015.

[46] N. Engheta, W. D. Murphy, V. Rokhlin, and M. S. Vassiliou, "The fast multipole method (FMM) for electromagnetic scattering problems," *IEEE Transactions on Antennas and Propagation*, vol. 40, no. 6, pp. 634–641, 1992.

[47] T. Ericsson and A. Ruhe, "The spectral transformation Lanczos method for the numerical solution of large sparse generalized symmetric eigenvalue problems," *Mathematics of Computation*, vol. 35, no. 152, pp. 1251–1268, 1980.

[48] Y. A. Erlangga, "Advances in iterative methods and preconditioners for the Helmholtz equation," *Archives of Computational Methods in Engineering*, vol. 15, no. 1, pp. 37–66, 2008.

[49] O. G. Ernst and M. J. Gander, "Why it is difficult to solve Helmholtz problems with classical iterative methods," in *Numerical Analysis of Multiscale Problems*, ser. Lecture Notes in Computational Science and Engineering, I. G. Graham, T. Y. Hou, O. Lakkis, and R. Scheichl, Eds., vol. 83, Springer Berlin Heidelberg, 2012, pp. 325–363.

[50] C. Farhat and F.-X. Roux, "A method of finite element tearing and interconnecting and its parallel solution algorithm," *International Journal for Numerical Methods in Engineering*, vol. 32, no. 6, pp. 1205–1227, 1991.

[51] M. J. Gander, "Schwarz methods over the course of time," *Electronic Transactions on Numerical Analysis*, vol. 31, pp. 228–255, 2008.

[52] M. J. Gander and F. Kwok, "Best Robin parameters for optimized Schwarz methods at cross points," *SIAM Journal on Scientific Computing*, vol. 34, no. 4, A1849–A1879, 2012.

[53] M. J. Gander, F. Magoulès, and F. Nataf, "Optimized Schwarz methods without overlap for the Helmholtz equation," *SIAM Journal on Scientific Computing*, vol. 24, no. 1, pp. 38–60, 2002.

[54] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities," *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, pp. 1309–1331, 2009.

[55] D. Givoli, "Computational absorbing boundaries," in *Computational Acoustics of Noise Propagation in Fluids - Finite and Boundary Element Methods*, S. Marburg and B. Nolte, Eds., Springer Berlin Heidelberg, 2008, pp. 145–166.

[56] S. Gleyzes, S. Kuhr, C. Guerlin, J. Bernu, S. Deleglise, U. B. Hoff, M. Brune, J.-M. Raimond, and S. Haroche, "Quantum jumps of light recording the birth and death of a photon in a cavity," *Nature*, vol. 446, no. 7133, pp. 297–300, 2007.

[57] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Johns Hopkins University Press, 1996.

[58] I. G. Graham, M. Löhndorf, J. M. Melenk, and E. A. Spence, "When is the error in the h-BEM for solving the Helmholtz equation bounded independently of k?" *BIT Numerical Mathematics*, vol. 55, no. 1, pp. 171–214, 2015.

[59] A. Greenbaum, *Iterative Methods for Solving Linear Systems*. Society for Industrial and Applied Mathematics, 1997.

[60] W. Hackbusch, *Hierarchical matrices: Algorithms and analysis*. Springer-Verlag, 2015.

[61] S. Haroche, "Vie et mort d'un photon: une autre manière de voir," *La Lettre du Collège de France*, no. 20, pp. 8–9, 2007.

[62] V. Hernandez, J. E. Roman, and V. Vidal, "SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems," *ACM Transactions on Mathematical Software*, vol. 31, no. 3, pp. 351–362, 2005.

[63] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409–436, 1952.

[64] K. Hillewaert, "Development of the discontinuous Galerkin method for high-resolution, large scale CFD and acoustics in industrial geometries," PhD thesis, Université catholique de Louvain, Belgique, 2013.

[65] ——, "Exploiting data locality in the DGM discretisation for optimal efficiency," in *ADIGMA - A European Initiative on the Development of Adaptive Higher-Order Variational Methods for Aerospace Applications*, ser. Notes on Numerical Fluid Mechanics and Multidisciplinary Design, N. Kroll, H. Bieler, H. Deconinck, V. Couaillier, H. Ven, and K. Sørensen, Eds., vol. 113, Springer Berlin Heidelberg, 2010, pp. 11–23.

[66] C. Hirsch, *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics*, 2nd ed. Butterworth-Heinemann, 2007.

[67] P. Houston, I. Perugia, and D. Schotzau, "Mixed discontinuous Galerkin approximation of the Maxwell operator," *SIAM Journal on Numerical Analysis*, vol. 42, no. 1, pp. 434–459, 2004.

[68] F. Q. Hu, "On absorbing boundary conditions for linearized Euler equations by a perfectly matched layer," *Journal of Computational Physics*, vol. 129, no. 1, pp. 201–219, 1996.

[69] F. Ihlenburg and I. Babuška, "Finite element solution of the Helmholtz equation with high wave number part I: The h-version of the FEM," *Computers & Mathematics with Applications*, vol. 30, no. 9, pp. 9–37, 1995.

[70] ——, "Finite element solution of the Helmholtz equation with high wave number part II: The h-p version of the FEM," *SIAM Journal on Numerical Analysis*, vol. 34, no. 1, pp. 315–358, 1997.

[71] Intel Corporation, *Reference manual for Intel Math Kernel Library*, 2014.

[72] A. Johnen, J.-F. Remacle, and C. Geuzaine, "Geometrical validity of curvilinear finite elements," *Journal of Computational Physics*, vol. 233, pp. 359–372, 2013.

[73] P. Jolivet, "Méthodes de décomposition de domaine: Application au calcul haute performance," PhD thesis, Université de Grenoble, France, 2014.

[74] B. Kågström, P. Ling, and C. Van Loan, "GEMM-based level 3 BLAS: High-performance model implementations and performance evaluation benchmark," vol. 24, no. 3, pp. 268–302, 1998.

[75] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[76] D. B. Kirk and W.-m. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, 2nd ed. Morgan Kaufmann, 2012.

[77] S. Kuhr, S. Gleyzes, C. Guerlin, J. Bernu, U. B. Hoff, S. Deléglise, S. Osnaghi, M. Brune, J.-M. Raimond, S. Haroche, E. Jacques, P. Bosland, and B. Visentin, "Ultrahigh finesse Fabry-Pérot superconducting resonator," *Applied Physics Letters*, vol. 90, no. 16, 164101, 2007.

[78] P. K. Kundu, I. M. Cohen, and D. R. Dowling, *Fluid Mechanics*, 5th ed. Academic Press, 2011.

[79] Kungliga Vetenskapsakademien, "Measuring and manipulating indivudual quantum systems," 2012.

[80] ——, "Particle control in a quantum world," 2012.

[81] ——, "Pressmeddelande: Nobelpriset i fysik 2012," 2012.

[82] A. Kuzmin, M. Luisier, and O. Schenk, "Fast methods for computing selected elements of the greens function in massively parallel nanoelectronic device simulations," in *Euro-Par 2013 Parallel Processing*, ser. Lecture Notes in Computer Science, F. Wolf, B. Mohr, and D. an Mey, Eds., vol. 8097, Springer Berlin Heidelberg, 2013, pp. 533–544.

[83] J. Lambrechts, "Finite element methods for coastal flows: Application to the great barrier reef," PhD thesis, Université catholique de Louvain, Belgique, 2011.

[84] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic Linear Algebra Subprograms for FORTRAN usage," *ACM Transactions on Mathematical Software*, vol. 5, no. 3, pp. 308–325, 1979.

[85] E. G. Lewars, *Computational Chemistry*. Springer Netherlands, 2011.

[86] J.-Y. L'Excellent, "Multifrontal methods: Parallelism, memory usage and numerical aspects," Habilitation à diriger des recherches, Ecole normale supérieure de lyon, France, 2012.

[87] X. S. Li, "An overview of SuperLU: Algorithms, implementation, and user interface," *ACM Transactions on Mathematical Software*, vol. 31, no. 3, pp. 302–325, 2005.

[88] P.-L. Lions, "On the Schwarz alternating method I," in *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, Eds., Society for Industrial and Applied Mathematics, 1988, pp. 1–42.

[89] ——, "On the Schwarz alternating method III: A variant for nonoverlapping subdomains," in *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, Eds., Society for Industrial and Applied Mathematics, 1990, pp. 202–223.

[90] S. H. Lui, "Domain decomposition methods for eigenvalue problems," *Journal of Computational and Applied Mathematics*, vol. 117, no. 1, pp. 17–34, 2000.

[91] J.-C. Luo, "A domain decomposition method for eingenvalue problems," in *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, D. E. Keyes, T. F. Chan, G. Meurant, J. S. Scroggs, and R. G. Voig, Eds., Society for Industrial and Applied Mathematics, 1992, pp. 306–321.

[92] N. Marsic, "Implémentation et analyse des performances d'algorithmes de calcul scientifique sur GPU," Master's thesis, Université de Liège, Belgique, 2011.

[93] N. Marsic and C. Geuzaine, "Efficient finite element assembly of high order Whitney forms," *IET Science, Measurement & Technology*, vol. 9, pp. 204–210, 2 2015.

[94] N. Marsic, C. Waltz, J.-F. Lee, and C. Geuzaine, "Domain decomposition methods for time-harmonic electromagnetic waves with high order Whitney forms," *IEEE Transactions on Magnetics*, in press.

[95] A. Modave, "Absorbing layers for wave-like time-dependent problems: Design, discretization and optimization," PhD thesis, Université de Liège, Belgique, 2013.

[96] A. Moiola and E. A. Spence, "Is the Helmholtz equation really sign-indefinite?" *SIAM Review*, vol. 56, no. 2, pp. 274–312, 2014.

[97] P. Monk, *Finite Element Methods for Maxwell's Equations*. Oxford University Press, 2003.

[98] M. H. Nayfeh and M. K. Brussel, *Electricity and magnetism*. Wiley, 1985.

[99] J.-C. Nédélec, *Acoustic and Electromagnetic Equations: Integral Representations for Harmonic Problems*. Springer-Verlag, 2001.

[100] ——, "Mixed finite elements in R^3," *Numerische Mathematik*, vol. 35, pp. 315–341, 1980.

[101] A. Nicolet, G. Demesy, F. Zolla, and B. Vial, "Quasi-modal analysis of segmented waveguides," in *IEEE Conference on Antenna Measurements Applications (CAMA)*, 2014, pp. 1–4.

[102] H. Němec, P. Kužel, J.-L. Coutaz, and J. Čtyroký, "Transmission properties and band structure of a segmented dielectric waveguide for the terahertz range," *Optics Communications*, vol. 273, no. 1, pp. 99–104, 2007.

[103] Z. Peng and J.-F. Lee, "Non-conformal domain decomposition method with second-order transmission conditions for time-harmonic electromagnetics," *Journal of Computational Physics*, vol. 229, no. 16, pp. 5615–5629, 2010.

[104] V. Rawat and J.-F. Lee, "Nonoverlapping domain decomposition with second order transmission condition for the time-harmonic Maxwell's equations," *SIAM Journal on Scientific Computing*, vol. 32, no. 6, pp. 3584–3603, 2010.

[105] D. D. Reynolds, *Engineering Principles of Acoustics: Noise and Vibration Control*. Allyn and Bacon, 1981.

[106] V. Rokhlin, "Rapid solution of integral equations of classical potential theory," *Journal of Computational Physics*, vol. 60, no. 2, pp. 187–207, 1985.

[107] J. E. Roman, C. Campos, E. Romero, and A. Tomás, "SLEPc users manual," Departament de Sistemes Informàtics i Computació, Universitat Politècnica de València, Tech. Rep. DSIC-II/24/02 - Revision 3.6, 2015.

[108] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Society for Industrial and Applied Mathematics, 2003.

[109] ——, *Numerical Methods for Large Eigenvalue Problems*, 2nd ed. Society for Industrial and Applied Mathematics, 2011.

[110] Y. Saad and M. H. Schultz, "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[111] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley, 2010.

[112] O. Schenk, M. Bollhöfer, and R. A. Römer, "On large-scale diagonalization techniques for the Anderson model of localization," *SIAM Review*, vol. 50, no. 1, pp. 91–112, 2008.

[113] O. Schenk, A. Wächter, and M. Hagemann, "Matching-based preprocessing algorithms to the solution of saddle-point problems in large-scale nonconvex interior-point optimization," *Computational Optimization and Applications*, vol. 36, no. 2-3, pp. 321–341, 2007.

[114] J. H. Schmid, P. Cheben, P. J. Bock, R. Halir, J. Lapointe, S. Janz, A. Delâge, A. Densmore, J.-M. Fedeli, T. J. Hall, B. Lamontagne, R. Ma, I. Molina-Fernandez, and D.-X. Xu, "Refractive index engineering with subwavelength gratings in silicon microphotonic waveguides," *IEEE Photonics Journal*, vol. 3, no. 3, pp. 597–607, 2011.

[115] F. Schmidt-Kaler, "Quantum physics: Total surveillance," *Nature*, vol. 446, no. 7133, pp. 275–276, 2007.

[116] E. Schrödinger, "Die gegenwärtige Situation in der Quantenmechanik," *Naturwissenschaften*, vol. 23, no. 48, pp. 807–812, 1935.

[117] L. Schwartz, *Théorie des distributions*. Hermann, 1966.

[118] H. A. Schwarz, "Über einen Grenzübergang durch alternirendes Verfahren," *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, vol. 15, pp. 272–286, 1870.

[119] L. C. Shen and J. A. Kong, *Applied Electromagnetism*. PWS Publishing, 1995.

[120] G. L. G. Sleijpen and H. A. Van der Vorst, "A Jacobi–Davidson iteration method for linear eigenvalue problems," *SIAM Review*, vol. 42, no. 2, pp. 267–293, 2000.

[121] M. Spivak, *Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus*. W. A. Benjamin, 1965.

[122] G. W. Stewart, "A Krylov–Schur algorithm for large eigenproblems," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 3, pp. 601–614, 2002.

[123] ——, "Addendum to 'a Krylov–Schur algorithm for large eigenproblems'," *SIAM Journal on Matrix Analysis and Applications*, vol. 24, no. 2, pp. 599–601, 2002.

[124] B. Stroustrup, *Programming: Principles and Practice Using C++*. Addison-Wesley, 2008.

[125] A. S. Tanenbaum, *Modern Operating Systems*, 3rd ed. Prentice Hall Press, 2009.

[126] F. L. Teixeira and W. C. Chew, "Analytical derivation of a conformal perfectly matched absorber for electromagnetic waves," *Microwave and Optical Technology Letters*, vol. 17, no. 4, pp. 231–236, 1998.

[127] ——, "Complex space approach to perfectly matched layers: A review and some new developments," *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, vol. 13, no. 5, pp. 441–455, 2000.

[128] B. Thierry, A. Vion, S. Tournier, M. El Bouajaji, D. Colignon, N. Marsic, X. Antoine, and C. Geuzaine, "GetDDM: An open framework for testing optimized Schwarz methods for time-harmonic wave problems," *Computer Physics Communications*, submitted.

[129] A. Vion, "Multi-domain approaches for the solution of high-frequency time-harmonic propagation problems," PhD thesis, Université de Liège, Belgique, 2014.

[130] A. Vion and C. Geuzaine, "Double sweep preconditioner for optimized Schwarz methods applied to the Helmholtz problem," *Journal of Computational Physics*, vol. 266, pp. 171–190, 2014.

[131] P. Šolín, K. Segeth, and I. Doležel, *Higher-order finite element methods*. Chapman & Hall/CRC, 2003.

[132] S. A. Ward and R. H. Halstead, *Computation Structures*. MIT Press, 1990.

[133] R. C. Whaley and A. Petitet, "Minimizing development and maintenance costs in supporting persistently optimized BLAS," *Software: Practice and Experience*, vol. 35, no. 2, pp. 101–121, 2005.

[134] Z. Xianyi, W. Qian, and Z. Yunquan, "Model-driven level 3 BLAS performance optimization on Loongson 3A processor," in *IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS)*, 2012, pp. 684–691.

[135] K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Transactions on Antennas and Propagation*, vol. 14, no. 3, pp. 302–307, 1966.

[136] S. Zaglmayr, "High order finite element methods for electromagnetic field computation," PhD thesis, Johannes Kepler Universität, Österreich, 2006.

[137] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu, *The Finite Element Method: Its Basis and Fundamentals*, 7th ed. Butterworth-Heinemann, 2013.

[138] O. C. Zienkiewicz, R. L. Taylor, and D. Fox, *The Finite Element Method for Solid and Structural Mechanics*, 7th ed. Butterworth-Heinemann, 2014.