Universitat Autònoma
de Barcelona

Application of the edge-based finite element method for fusion plasma simulations

Physics Department, Universitat Autònoma de Barcelona

Fusion Group, CASE Department, Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC-CNS)

Bachelor's thesis for the degree in Physics

Marc Fuster Rullan

Supervised by:
Shimpei Futatani and Daniel Campos

Abstract

Fusion is a clean energy source which is a promising future nuclear energy resource. One of the conditions of the nuclear fusion on Earth is to confine very high temperature ionized particles forming a plasma with magnetic field lines. However, the magnetically confined plasma is not an equilibrium system which leads to a complicated dynamics. Its spatio-temporal dynamics is not easy to model. **The goal of the work is to asses the feasibility of the development of a useful computational tool for fusion applications based on edge elements, which is a preferable numerical scheme for electromagnetic physics including plasma physics**. The development uses the infrastructures of the Parallel Edge-based Tool for Geophysical Electromagnetic Modelling (PETGEM) which is a Python HPC scalable tool based on edge finite element method. This code is used to solve the so called *Marine Controlled-Source Electromagnetic method* (CSEM) which is an important technique for reducing ambiguities in data interpretation for hydrocarbon exploration. Finite element method (FEM) and both its nodal and edge version are explained in this thesis along with a comparison of both methods and the reasons why the edge elements are more suitable for electromagnetic problems. The indexing and formulation of edge FEM is explained as well as the assembly and the solver used in PETGEM. This work contains a tutorial on how to use PETGEM which is the result of the learning process of the multiple simulations carried out. The mesh generation and refinement is achieved using Gmsh software. The implementation and the benchmark of different physical initial profiles are achieved through parallel simulations in Marenostrum supercomputer.

Acknowledgments

I wish to express my sincere appreciation to those who have contributed to this thesis and supported me in one way or the other during the development of this thesis.

I am deeply thankful to both of my advisors Dr. Shimpei Futatani (Physics Department, UPC and formely at Fusion Group at BSC) and Dr. Daniel Campos (Statistical Physics Department, UAB) for giving me support and guidance. Moreover, I want to thank ICREA Professor Dr. Mervi Mantsinen for accepting me in his team, the Fusion Group in the department Computer Applications in Science & Engineering department (CASE) of the Barcelona Supercomputing Center-Centro Nacional de Supercomputación (BSC-CNS). Finally, my gratitude goes to Dr. Octavio Castillo Reyes, the creator of PETGEM code, for all the support that he has provided along this thesis.

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

Communication of this work

- Marc Fuster et al. Application of the edge based finite element method for fusion plasma simulations, Poster presentation at 5^{th} BSC Severo Ochoa Doctoral Symposium $25^{th}$of April, 2018, Barcelona, Spain.

Contents

1 Introduction

1.1 Why the computational study of physics is required?

Physics is based on formulating mathematical models to describe physical phenomena. These models often rely on solving systems of differential equations, large sums or integrals, derivatives, ... In general, solving analytically the mathematical model for a particular system is not possible. This can occur when the solution does not have a closed-form expression, or is too complicated to obtain it. In such cases, numerical approximations are used to solve the model. The approximations are usually obtained by simple mathematical operations repeated many times. Computers have the ability to repeat those operations in a short time. Those approximations can produce errors from the exact solution. However, those numerical errors can be reduced to the level that they become irrelevant. The reduction of the errors can be both achieved by improving the numerical routines (theoretically more difficult) or reducing the numerical step (requiring more computational time). For example, in a numerical integration, one can use the Trapezoidal rule [1] with a small step that will require large computational times but an easy implementation of the method. On the other hand, one can apply Romberg's integration [2] with a higher step that will require less computational time but larger implementation. Numerical methods are not only important but essential for most research in physics and many others disciplines. Moreover, computational physics allow physicists to realize "numerical experiments". Those numerical experiments or simulations are extremely powerful, for example, one can modify parameters of an experiment which it would be much more difficult and expensive to do it in real experiments. As an example in plasma physics, one can simulate different geometries and determine which one causes fewer instabilities. Building many geometries would be both expensive and slow. Another important application of computational physics is weather prediction. Predicting the trajectory of hurricane Irma on September 2017 allowed the population of the affected zones to get prepared for the hurricane or even escape it. Computer simulations are also essential for topics such as aerodynamics, particle physics, space rockets, climate change prediction, traffic prediction.

1.2 Nuclear Fusion

The human population in the world is increasing remarkably. Accordingly, the energy consumption is expected to increase as shown in Fig. 1. However, current energy sources, such as fossil fuels or coal, are limited. One alternative to fossil fuels is nuclear fission. However, fission is not preferable for the environment as it generates radioactive products which are harmful and dangerous for the population. Other alternatives are solar and wind energy. Yet, those clean sources only produce energy in some time intervals and the energy obtained cannot be stored because high capacity batteries are not yet available.

One of the best candidates in a middle term is nuclear fusion. The nuclear fusion is considered as a promising future energy resource for two reasons. First, it does not emit any harmful residual such as radioactive isotopes or greenhouse gases. Second, there is no problem with the fuel source as it is two hydrogen isotopes. Fusion must be distinguished from fission. Nuclear fission is the decay of a heavy atomic nucleus into two lighter fragments. In a thermonuclear fusion reaction, nuclei of light atoms are combined to create a new element of a slightly less mass than the sum of the initial atoms. The missing mass is converted into mass according to Einstein's formula $E = \triangle mc^2$. In order to achieve

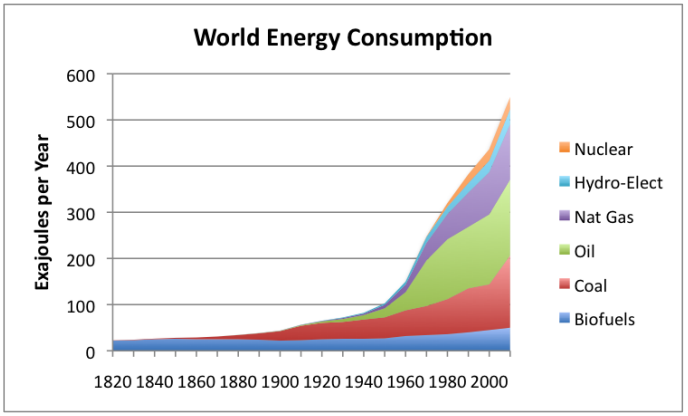

Figure 1: Global energy consumption over recent history [3].

the nuclear fusion as an energy source, the most attractive reaction involves the fusion of deuterium (D) and tritium (T) ions as shown in Fig. 2. This reaction releases an energetic neutron with a kinetic energy of 14.1 MeV and a Helium ion or alpha particle with a kinetic energy of 3.5 MeV. The sum of both kinetic energies is 17.6 MeV. Alpha particles will be in charge of maintaining the reaction.

$$^{2}_{1}\mathrm{H} + ^{3}_{1}\mathrm{H} \rightarrow ^{4}_{2}\mathrm{He} + ^{1}_{0}\mathrm{n} + 17.6MeV$$

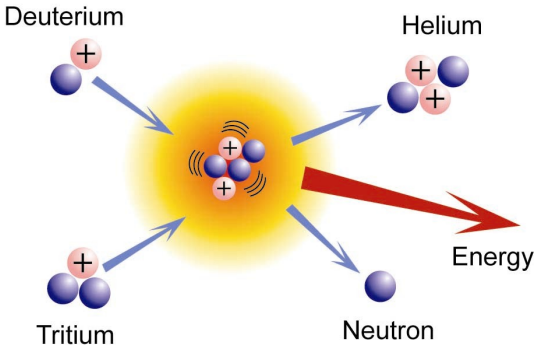

Figure 2: Deuterium tritium reaction [4]

Despite the many advantages of fusion over fission, it has been proven to be difficult to achieve nuclear fusion. In order to start the reaction, one needs to achieve the plasma state of the fuel at a very high temperature and density. However, it is not easy to confine the fuel in such state. Both experiments and simulations must be carried out in order to understand how to best confine the plasma.

Temperature is not the only key factor but also a suitable confinement and density are required to achieve fusion. An important concept in the field of nuclear fusion is the Lawson Criteria. It defines the condition between the plasma electron density n_e, the energy confinement time τ_E, and the ignition plasma temperature T. Lawson's triple product represents a power balance in thermonuclear reactors in order reach a self- sustained state in which no input of energy is required. For Deuterium-Tritium reactions, and temperatures of the order of $T = 14KeV$, the triple product becomes:

$$n_e \tau_E T \geq 3 \cdot 10^{21} KeVs/m^3 \tag{1}$$

1.3 Plasma

It is often said that the plasma is a cloud of electrons, protons, and neutrons. Those electrons have been excited from their respective molecules and atoms due to high thermal excitation. However, not all ionized gas can be called a plasma. A definition of a plasma is proposed by Chen [5]:

A plasma is a quasi-neutral gas of charged and neutral particles which exhibits collective behavior.

The meaning of "collective behavior" means that plasmas have more interaction than regular liquids and gases. This arises from the fact that when plasma particles move they create electromagnetic fields that modify the dynamics of the plasma particles. The plasmas interact among themselves more than liquids and gases. The "quasineutrality" means that even the net charge of the plasma is zero as in most matter, particles are ionized and ions and electrons are separated and can have different behavior. As the general rule, one can take $n_{electrons} \simeq n_{protons}$ but one cannot neglect electromagnetic fields created by small differences in the density of electrons and ions.

The criteria for an ionized gas to be plasma can be summarized in three mathematical conditions [5]:

- $\mathbf{\lambda_D \ll L}$
  Where L is the dimension of the overall plasma and λ_D is the Debye length $\lambda_D = \left(\frac{\epsilon_0 KT}{e^2 n}\right)^2$. These condition emerges from the "Debye shielding", the attenuation of any potential present in the plasma. The Debye shielding has been observed experimentally.

- $\mathbf{N_D = n\frac{4}{3}\pi\lambda_D \gg 1}$
  Debye shielding is only valid if there are enough particles in the plasma. One can compute the number of particles in a "Debye sphere" (N_D) that is a sphere of Debye length radius. The number of particles is obtained by multiplying the density of particles (n) by the volume of the sphere $\frac{4}{3}\pi\lambda_D$. This quantity must be much larger than 1.

- $\mathbf{w \cdot \tau > 1}$
  This conditions are still fulfilled by jet exhaustion's particles which interact more with normal air particles. One requires more electromagnetic interactions than ordinary hydrodynamic interactions. Therefore, $w \cdot \tau > 1$ where τ is the mean time between collisions with neutral atoms and w is the frequency of typical plasma oscillations.

Plasmas appear in many applications besides nuclear fusion:

- **Illumination** Plasma is used in illumination in some devices. For example, gas-discharge lamps use plasma in order to illuminate the environment. Another example is plasma displays, they use small cells containing electrically charged ionized gases, which are plasmas.

- **Propulsion** Plasma can be used as ion propulsion [6]. Due to their large charge-mass ratio, ions can be accelerated by electric fields to large velocities. Throwing ions in the opposite direction will accelerate rockets. The large accelerations achieved need

less fuel than normal combustion. Therefore, plasma propulsion fuel is more efficient than usual combustion.

- **Medicine** Plasma can be used in medicine for sterilization of implants or surgical instruments. It is also used to modify biomaterials [7]
- **Industry** Plasmas are used in semiconductors devices fabrication and industry. Some processes that use plasma are plasma activation, plasma etching, plasma cleaning, ... [8].

1.4 Fusion Reactors

There are many different devices to study fusion. A summary of them is given in Fig. 3. The most used category is the magnetic confinement. In the magnetic confinement, the high-temperature fusion plasma is confined by strong magnetic fields in order to avoid touching the reactor wall. One can classify magnetic confinement systems in open-end systems and closed systems. In open systems, plasma does not repeat the same path, while on closed systems plasma does repeat the same path over and over. One can compare this separation to linear accelerators and synchrotrons in particle physics. Here we will focus mainly on tokamak and stellarators.

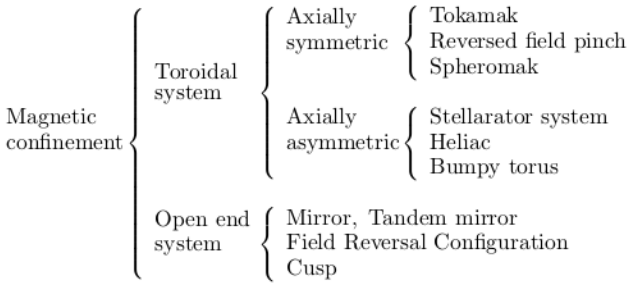

Figure 3: Fusion devices classification [9]

The tokamak is the most common magnetic confinement option for fusion worldwide. This design was proposed in 1950 by Soviet physicists Igor Tamn and Andrei Sakharov inspired by the idea of Oleg Lavrentiev [10] and comes from a Russian acronym that stands for "toroidal chamber with magnetic coils". The tokamak and stellarator configurations are shown in Fig. 4. The $\mathbf{E}\times\mathbf{B}$ drift shifts all particles to the external part of the toroid since $v_{E\times B} = \frac{\mathbf{E}\times\mathbf{B}}{B^2} = -\frac{E_z}{B}\hat{e}_R$. On the other hand, curvature drift separate ions and electrons as in the Fig. 5. The solution to the charge separation problem is giving a small poloidal component to the magnetic field. Furthermore, the poloidal cross-section of tokamaks are not exactly circular, it has some ellipsity and some triangularitiy to improve the plasma stability.

The second main option to achieve controlled nuclear fusion is the stellarator. It was invented on 1951 by Lyman Spitzer (Princeton). The main difference between a stellarator and a tokamak is the stellarator's more complex shape as illustrated in Fig. 4. The twisting paths were proposed in order to cancel out instabilities seen in tokamaks. In the following years, stellarators were constructed and demonstrated poor performance due to a problem called "pump out". Since the 1960s tokamak showed higher performance than

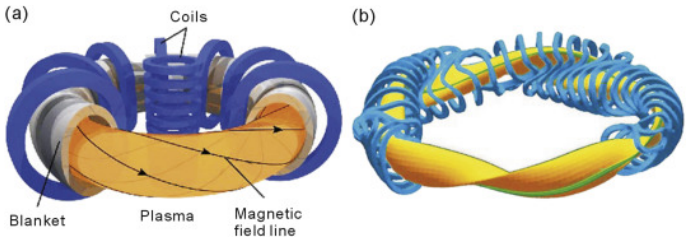

Figure 4: [11] Schematic picture of a) tokamak and b) stellarator

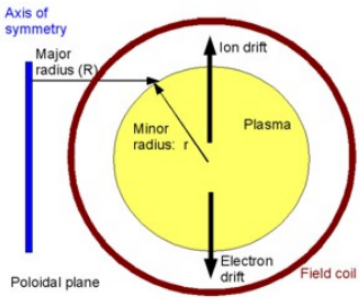

Figure 5: Poloidal cut of a tokamak [12]

stellarators. However, by the 1990s tokamaks were proved to have similar problems than stellarators. The main stellarator project nowadays is Wendelstein 7-X in Germany.

ITER and JET projects are based on tokamak. ITER is an international nuclear fusion research project [13]. ITER's objective is to demonstrate the scientific and technological feasibility of fusion energy for peaceful purposes. ITER is being constructed in Saint-Paul-lès-Durance (France) and will have the world's largest tokamak nuclear fusion reactor. This project is funded worldwide by European Union, India, Japan, China, Russia, South Korea, and the United States. ITER reactor is aimed to make the transition from the experimental fusion to the full-scale fusion power stations. It is expected to produce 500 MW with an input power of 50 MW ($Q = \frac{P_{output}}{P_{input}} = 10$). Some ITER objectives are to produce a steady-state plasma with a Q greater than 5, to maintain a fusion pulse up to 8 minutes and to ignite a self-sustained plasma. The reactor will be able to host $840m^3$ surpassing the volume of the largest present-day device, the Joint European Torus (JET) by a factor of 10. After ITER the next international big project will be DEMO (DEMOnstration Power Station) that is intended to lie somewhere between those of ITER and a "first of a kind" commercial station.

The situation when $Q = 1$ is called "breakeven". That situation implies that the power gains and the looses balance each other. The ideal situation is when $Q = \infty$ (ignition), that means disconnecting the power input $P_{input} = 0$ and still maintaining the reaction by just adding more fuel. While this situation has not been achieved in 2017, there is plenty of research on the relevance of the alpha particles in maintaining the reaction. While the alpha particles may have the duty to achieve ignition, the neutrons produced have the duty to produce the output energy. The 14.1 MeV neutrons will be isotropically distributed and absorbed by the reactors' walls that will heat water located in a vessel surrounding the reactor. Therefore, water will boil and will move a turbine producing electricity.

1.5 Magnetohydrodynamics (MHD)

Magnetohydrodynamics (MHD) is the study of the magnetic and electric properties of conducting fluids. The field of MHD was started by Hannes Alfvén for which he received the Physics Nobel prize in 1970. MHD is a mathematical model used to describe the plasma. While there are some extensions of MHD, this work will focus on the simplest model, "Ideal MHD". This simplification relies on the assumption that the fluid has so little resistivity that can be considered a perfect conductor. Moreover, since plasma is modeled as a fluid, some physics as kinetic effects are ignored. Yet, although the model is simple, it describes most basic properties of tokamak plasmas. The "Ideal MHD" is a system of seven differential equations given by "continuity, momentum and energy equations" together with "low-frequency Maxwell's equations".

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{2}$$

$$\rho \frac{d\mathbf{v}}{dt} = \mathbf{J} \times \mathbf{B} - \nabla p \tag{3}$$

$$\frac{d}{dt}\left(\frac{p}{\rho^{\gamma}}\right) = 0 \tag{4}$$

$$\mathbf{E} + \mathbf{v} \times \mathbf{B} = 0 \tag{5}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{6}$$

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{J} \tag{7}$$

$$\nabla \cdot \mathbf{B} = 0, \tag{8}$$

where ρ is the density, $\mathbf{v}$ is the fluid velocity, p is the pressure, $\mathbf{E}$ and $\mathbf{B}$ are the electric and magnetic field, $\mathbf{J}$ is the current density and γ is the "heat capacity ratio". Since the plasma is ionized, it can be considered ideal and monoatomic ($\gamma = 5/3$). Furthermore, Eqs. 3 and 4 use the "convective derivative" ($d/dt = \partial/\partial t + \mathbf{v} \cdot \nabla$). It is important to remark that MHD is a mixture between Fluid Mechanics (Eqs. 2-4) and Electromagnetism (Eqs. 5-8). The main concept behind MHD is that the magnetic fields can create currents in the plasmas. Those fields polarize the fluid and change the magnetic field. MHD is not only used in fusion but also in geophysics where it is used in the Earth's core to study the Earth's magnetic field and in astrophysics, where it is used, among other examples, to study the jet's propulsion system and the stars.

1.6 State of the art and motivation of the work

There are many different fusion simulation codes based on different numerical schemes. However, there is not any complete code which solves all the physics involved in a fusion reactor. Some examples of fusion codes are: JOREK [14] which solves three dimensional non-linear MHD model, PION [15] code which solves Fokker-Planck equation to study the

heating by Ion Cyclotron Resonance Frequency Waves (ICRF), ASCOT [16] is a Monte-carlo code which also studies ICRF and EUTERPE [17] uses particle in cell scheme to study plasma instabilities.

The thesis project focuses on an edge finite element method (EFEM) which is not a common approach among the fusion community. The geophysics code PETGEM [18], which will be explained in section 3, is not only based on EFEM but also solves Maxwell's equations which are part of MHD equations, the long-term objective of this project. Moreover, PETGEM is written in Python, a programming language with a large growth in the recent years. PETGEM is High Performance Computing (HPC) and parallel. This HPC capability is essential to study plasma since it is turbulent and requires high density meshes which increase significantly the computational time required. Therefore these simulations are required to run in large supercomputers and PETGEM has been proved to be very efficient in Marenostrum supercomputer.

Therefore, PETGEM shows a good potential as an infrastructure to start this new research line. This project assesses the feasibility of the implementation of the edge based finite element method code PETGEM for fusion applications.

1.7 Structure of this thesis

This thesis is organized as follows. Chapter 1 gives the introduction to plasma physics and states the importance of fusion research. Chapter 2 explains the numerical scheme of this thesis. Finite element method both its nodal and edge version are explained along with a comparison of both methods and the reasons why the edge elements should be used for fusion problems. Chapter 3 introduces PETGEM code along with instructions on how to run the code. Those instructions are the result of the multiple simulations carried out using this code. Chapter 4 summarizes the results of this new research line to apply edge finite element method to fusion problems. Finally, chapter 5 contains the conclusions of this thesis and chapter 6 is a description of the future work to do in this new research line.

The thesis work consists of following four steps: to understand and learn EFEM theory, to understand the workflow and the scripts of PETGEM code, to check the capability to implement different initial profiles and to change the equations that PETGEM solves and introducing the time integration of the equations as summarized in this chapter:
Learn EFEM $\rightarrow$ Understanding PETGEM $\rightarrow$ Check initial profile $\rightarrow$ Change equations

2 Finite Element Method (FEM)

The Finite element method is a common approach to solve numerically differential equations in complex geometries, it is the approach used in PETGEM. In this chapter, the FEM method is explained starting from the nodal elements, then moving to the edge elements and remarking why edge elements should be used for electromagnetic problems. The indexing, the formulation and the assembly of the edge finite element method are explained in this chapter. However, firstly an explanation of the Finite Difference Method is given in order to better understand the Finite Element method.

2.1 Space discretization

Computer can not treat continuous space, they only can handle discrete space. Therefore, in order to use numerical methods, space must be discretized. There are different discretizations methods used to solve partial differential equations. Some examples are Finite Element Method (FEM), Finite Difference Method (FDM), Finite Volume Method (FVM), Spectral Method.

The Finite Difference Method (FDM) is the simplest numerical technique used to solve differential equations, especially partial differential equations. It is based on Taylor's expansion. By dividing space in homogeneously distributed by a separation h and expanding a function at a certain point x_0, one has:

$$f(x_0 + h) = f(x_0) + f'(x_0)h + f''(x_0)h^2/2 + ... \,, \tag{9}$$

neglecting $O(h^2)$ and higher orders, one can easily get:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}. \tag{10}$$

Repeating the same procedure with the second derivative, one finally obtains.

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}. \tag{11}$$

These differential equations are solved by replacing the derivatives with the equations 10 and 11. FDM has an error of $O(h)$ over the first derivative and a $O(h^2)$ over the second derivative.

2.2 Main concepts of the Finite Element Method

The main idea behind FEM is to divide the whole domain into elements or subdomains which have associated simple equations that approximate the equations to solve. While FDM divides space homogeneously, FEM has a specific division of space to that leads to better representations of complex geometries, better capture of locals effects, etc. The elements can have different geometries and can be used for one-dimensional (1D), two-dimensional (2D) and three-dimensional (3D) problems. Some examples of 2D problems are rectangles and triangles while some 3D examples are the rectangular prism, the squared pyramid and the triangular pyramid (Tetrahedron). In the work, only tetrahedrons will be used because they are the only ones supported by PETGEM. PETGEM focuses on tetrahedrons because they are the easiest to scale-up to very large domains or arbitrary shape. Figure 6 shows a comparison between FDM's and FEM's discretizations.

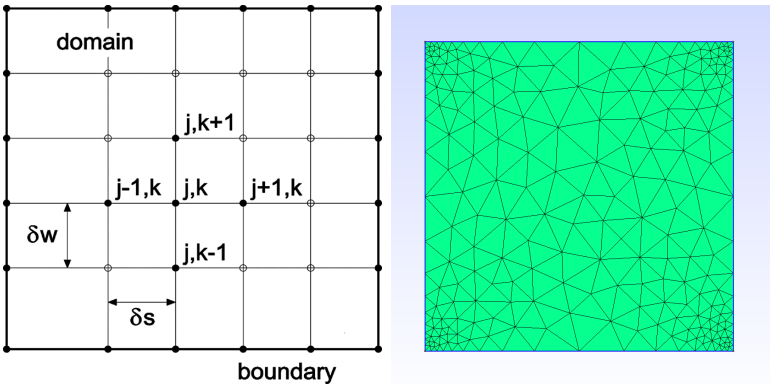

Figure 6: FDM example on the left part [19]. On the right part the same example but with triangular mesh used in FEM.

Figure 6 shows that the corners have more density of elements in order to achieve more accuracy. Increasing the density of elements is called *Refinement*. The refinement produces two main advantages for FEM: The first one is that it allows more accuracy in the target areas. The second advantage is the reduction of the computational time by decreasing the density of elements in the areas where the function has a low changing rate. These methods are based on an interpolation function whose coefficients are obtained from solving a matrix equation $A \cdot x = b$.

FEM is based on two main approaches: the nodal based element and the edge based element. The main idea in both nodal and edge elements is to solve a matrix equation $A \cdot x = b$ and then interpolate the result at any point using interpolation functions (often called basis function). The construction of the matrix A and the array b is a complex task that will be explained in the following subsections. The solution x are the coefficients of the interpolation functions and in FEM theory are called degrees of freedom (DOFs). The main difference between both methods is that in the nodal elements the DOFs are assigned to the nodes while on the edge elements are assigned to the edges. FEM can be used different geometries but along this work, only triangles for 2D and tetrahedra for 3D will be used, as shown in Fig. 7. The interpolation functions are scalar for nodal elements and vectorial for edge elements. Therefore, the output of the nodal elements will be scalar while the edges' output will be vectorial. Since this thesis focuses on solving electromagnetic problems, the output of nodal elements will be the electric potential (ϕ) while the edges' output will be the electric field ($\mathbf{E}$). Finally, due to the fact that edges elements are built over nodal elements, the nodal elements will be explained first.

2.3 Nodal elements

The main idea of nodal elements is to obtain the coefficients ϕ_i^e that will allow obtaining the electric potential at any point using the interpolation functions. In order to obtain the DOFs (ϕ_i^e) one must solve an equation $A \cdot \phi = b$. The building of A and b will be explained later. First of all, the 2D problem will be explained and then the 3D.

Two dimensional nodal:
The 2D interpolation function is given by the following equation. Due to the fact that nodal elements have the DOFs in the nodes, the interpolation requires a sum from 1 to 3,

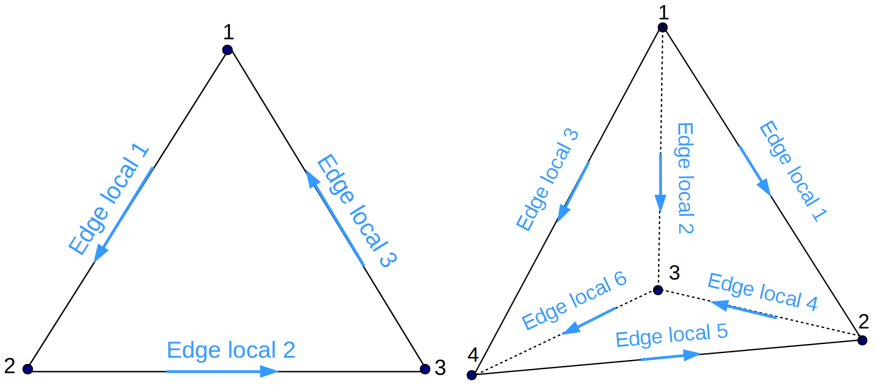

Figure 7: On the left side, the 2D geometry, the triangle. On the right side, the 3D geometry, the tetrahedron.

each number corresponding to one node:

$$\phi^e(x,y) = \sum_{i=1}^{3} W_i^e(x,y)\phi_i^e, \tag{12}$$

where $W_i^e(x,y)$ are the interpolation functions and follow:

$$W_i^e(x,y) = \frac{1}{2\triangle^e}(a_j^e + b_j^e x + c_j^e y) \qquad j = 1,2,3 \ , \tag{13}$$

where a_j^e, b_j^e and c_j^e are parameters given by (x_i^e, y_i^e) which are the location of the node i for each triangle e. Moreover $\triangle^e$ is the surface of the triangle. Those coefficients are given by:

$$\begin{array}{lll} a_1^e = x_2^e y_3^e - y_2^e x_3^e & b_1^e = y_2^e - y_3^e & c_1^e = x_3^e - x_2^e \\ a_2^e = x_3^e y_1^e - y_3^e x_1^e & b_2^e = y_3^e - y_1^e & c_2^e = x_1^e - x_3^e \\ a_3^e = x_1^e y_2^e - y_1^e x_2^e & b_3^e = y_1^e - y_2^e & c_3^e = x_2^e - x_1^e \end{array} \tag{14}$$

$$\triangle^e = \frac{1}{2}(b_1^e c_2^e - b_2^e c_1^e)$$

Three dimensional nodal: As in the two-dimensional case, the 3D interpolation function is given by the following equation. As seen in Fig. 7, tetrahedron has 4 nodes and 6 edges. Therefore, this sum is from 1 to 4 each for every node:

$$\phi^e(x,y,z) = \sum_{i=1}^{4} W_i^e(x,y,z)\phi_i^e \ , \tag{15}$$

where the interpolation function $W_i^e(x,y,z)$ is:

$$W_i^e(x,y,z) = \frac{1}{6V^e}\left(a_i^e + b_i^e x + c_i^e y + d_i^e z\right) \ . \tag{16}$$

In the same way, as in the 2D case, V^e is the volume of the tetrahedron. This volume and the coefficients of the interpolation function are obtained from the nodes of the tetrahedron

using:

$$a_i^e = \begin{vmatrix} x_{i+1}^e & x_{i+2}^e & x_{i+3}^e \\ y_{i+1}^e & y_{i+2}^e & y_{i+3}^e \\ z_{i+1}^e & z_{i+2}^e & z_{i+3}^e \end{vmatrix}, \tag{17}$$

$$b_i^e = \begin{vmatrix} 1 & 1 & 1 \\ y_{i+1}^e & y_{i+2}^e & y_{i+3}^e \\ z_{i+1}^e & z_{i+2}^e & z_{i+3}^e \end{vmatrix}, \tag{18}$$

$$c_i^e = \begin{vmatrix} 1 & 1 & 1 \\ x_{i+1}^e & x_{i+2}^e & x_{i+3}^e \\ z_{i+1}^e & z_{i+2}^e & z_{i+3}^e \end{vmatrix}, \tag{19}$$

$$d_i^e = \begin{vmatrix} 1 & 1 & 1 \\ x_{i+1}^e & x_{i+2}^e & x_{i+3}^e \\ y_{i+1}^e & y_{i+2}^e & y_{i+3}^e \end{vmatrix}, \tag{20}$$

$$V_i^e = \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1^e & x_2^e & x_3^e & x_4^e \\ y_1^e & y_2^e & y_3^e & y_4^e \\ z_1^e & z_2^e & z_3^e & z_4^e \end{vmatrix}, \tag{21}$$

where $i_{4+k} = i_k$ and (x_i^e, y_i^e, z_i^e) give the position of the node i of the element e.

2.4 Edge elements

Similarly with the nodal elements, the main idea of edge elements is to obtain the coefficients E_i^e that will allow to obtain the electric field at any point using the interpolation functions. In order to obtain the DOFs (E_i^e) one must solve an equations $A \cdot E = b$. The building of A and b will be explained later. As before, the 2D case will be explained first and the 3D thereafter.

Two dimensional edge:
Once the system is solved and the DOFs are obtained, one can recover the electric field at any point of the domain using the same interpolation function. Since the triangle has 3 edges, the sum is from 1 to 3 one corresponding to each edge.

$$\mathbf{E^e}(x, y) = \sum_{j=1}^{3} \mathbf{N_j^e}(x, y) E_j^e \ . \tag{22}$$

It is important to remark that for edge elements, the interpolation functions are now completely different from the nodal. In fact, the interpolation functions for edge elements $(\mathbf{N_j^e})$ are computed using nodal interpolation functions (W_j^e):

$$\mathbf{N}_1^e = \left(W_1^e \vec{\nabla} W_2^e - W_2^e \vec{\nabla} W_1^e \right) l_1^e \tag{23}$$

$$\mathbf{N}_2^e = \left(W_2^e \vec{\nabla} W_3^e - W_3^e \vec{\nabla} W_2^e \right) l_2^e \tag{24}$$

$$\mathbf{N}_3^e = \left(W_3^e \vec{\nabla} W_1^e - W_1^e \vec{\nabla} W_3^e \right) l_3^e. \tag{25}$$

Here W_i^e correspond to the basis functions of 2D nodal and l_i^e is the length of the edge i of the element e.

Three dimensional edge:
In this case, the tetrahedron has 6 edges. Therefore, the interpolation function becomes a sum from 1 to 6 each one corresponding to one of the edges:

$$\mathbf{E^e}(x,y) = \sum_{j=1}^{6} \mathbf{N_j^e}(x,y) E_j^e, \tag{26}$$

where $\mathbf{N_j^e}$ are given by Eqs. 27-32 and E_j^e are the DOFs obtained by solving $A \cdot E = b$.

$$\mathbf{N}_1^e = \left(W_1^e \vec{\nabla} W_2^e - W_2^e \vec{\nabla} W_1^e\right) l_1^e \tag{27}$$

$$\mathbf{N}_2^e = \left(W_1^e \vec{\nabla} W_3^e - W_3^e \vec{\nabla} W_1^e\right) l_2^e \tag{28}$$

$$\mathbf{N}_3^e = \left(W_1^e \vec{\nabla} W_4^e - W_4^e \vec{\nabla} W_1^e\right) l_3^e \tag{29}$$

$$\mathbf{N}_4^e = \left(W_2^e \vec{\nabla} W_3^e - W_3^e \vec{\nabla} W_2^e\right) l_4^e \tag{30}$$

$$\mathbf{N}_5^e = \left(W_4^e \vec{\nabla} W_2^e - W_2^e \vec{\nabla} W_4^e\right) l_5^e \tag{31}$$

$$\mathbf{N}_6^e = \left(W_3^e \vec{\nabla} W_4^e - W_4^e \vec{\nabla} W_3^e\right) l_6^e \tag{32}$$

2.5 Comparison between nodal and edge elements

The two key points of the comparison rely on the discontinuities of the EM fields and on the necessity to do the gradient of the electric potential in order to obtain the electric field. From the integral form of Maxwell equation's, one can derive the interface conditions for the electromagnetic fields. The final expressions are:

$$\mathbf{n}_{12} \times (\mathbf{E}_2 - \mathbf{E}_1) = 0 \qquad (\mathbf{D}_2 - \mathbf{D}_1) \cdot \mathbf{n}_{12} = \rho_s \tag{33}$$

$$\mathbf{n}_{12} \times (\mathbf{H}_2 - \mathbf{H}_1) = \mathbf{j}_s \qquad (\mathbf{B}_2 - \mathbf{B}_1) \cdot \mathbf{n}_{12} = 0, \tag{34}$$

where ρ_s and $\mathbf{j}_s$ are the surface charge density and the surface current density of the interface, $\mathbf{n}_{12}$ is a vector perpendicular to the surface that has its origin in media 1 and its end in media 2. Let $\perp$ be parallel to $\mathbf{n}_{12}$ (and perpendicular to the interface) and $\parallel$ perpendicular to $\mathbf{n}_{12}$ (and contained in the interface). If $\rho_s = 0$ and $\mathbf{j}_s = 0$, all components of the EM fields are continuous. However, if $\rho_s \neq 0$ and $\mathbf{j}_s \neq 0$, Eqs. 33 and 34 show that $D_\perp$ and $H_\parallel$ are not continuous. Therefore, $E_\perp$ and $B_\parallel$ are not continuous also.

To sum up, when electromagnetic fields face an interface, tangential component of E ($E_\parallel$) and normal component of B ($B_\perp$) are always continuous. However, $E_\perp$ and $B_\parallel$ are not continuous when there is a density charge and a current density, respectively, on the interface. When using nodal elements, all components are automatically continuous functions. When solving discontinuous fields, one must add a penalization term to be able to use nodal elements method. However, edge elements support discontinuities by construction and, therefore, are a better option [20].

The second key point in the comparison is that the nodal elements output the electric potential. Most of the times, the variable needed is the electric field. Therefore, when using nodal elements one must do the $-\nabla\phi$ in order to obtain the $\mathbf{E}$. Since the output ϕ is a set of value and not a function, the gradient must be computed numerically. Therefore, further errors are introduced in the solution. To sum up, for EM problems edge elements are preferred over nodal elements.

2.6 Indexing

A good indexing rule is needed in order to keep control of the edges properties such as the position and the element which the edges belong to. In this section, the indexing used will be first explained in detail in 2D and thereafter extended to 3D problems. For both cases, the indexing rule is based on having both a global and a local index.

Two Dimensions:

When a Mesh is created all nodes are numbered from 1 to the total number of nodes. In order to keep track of the edges, a global and a local rule are set:

- **Global Rule:** Edges go from the lower possible node index to the second possible lower node index.
- **Local Rule:** Local edges follow this table rule:

Table 1: Local indexing for 2D triangular elements.

$Edge_i$	$node_{initial}$	$node_{end}$
1	1	2
2	2	3
3	3	1

The best way to understand this rule is using an example. Figure 8 contains a very simple case.

The global rule will be explained first. Starting from the node with index 1, it is connected to the nodes 2, 3 and 4. Therefore, edge 1 goes from node 1 to node 2, edge 2 goes from node 1 to node 3, ... When all connections beginning from node 1 are finished, comes node 2. Node 2 is connected to node 1 and node 4. However, the connection from node 1 and 2 has been already made. Therefore, edge 4 goes from node 2 to node 4 and so on. Therefore, the global indexing of this easy case is shown in Table 2.

Table 2: Global indexing for the easy example.

Global index	1	2	3	4	5
Initial node	1	1	1	2	3
End node	2	3	4	4	4

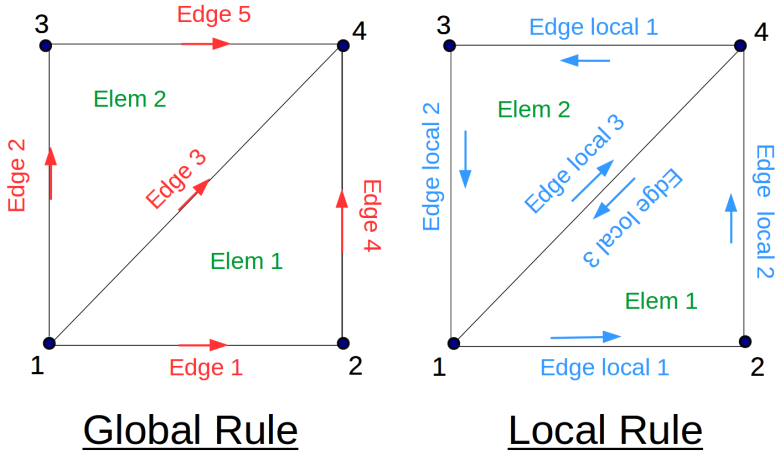

Figure 8: Simple example to understand indexing. In green the element number, in black the node index. The left box contains in red the indexes of the Global rule while the right one contains the indexes of the Local rule in blue colour.

As happened with the nodes indexing, the creation of the mesh outputs a matrix where each column contains the nodes' index of each element. In PETGEM this matrix is called elemsN and an example of this matrix for this easy system is given in Eq. 35

$$\text{elemsN} = \begin{array}{c} \\ \text{position 1} \\ \text{position 2} \\ \text{position 3} \end{array} \begin{array}{c} \begin{array}{cc} \text{Elem1} & \text{Elem 2} \end{array} \\ \begin{pmatrix} 1 & 4 \\ 2 & 3 \\ 4 & 1 \end{pmatrix} \end{array} \tag{35}$$

The explanation of the local indexing will start with the second element since the first one can be confusing due to the fact that node 1 and node 2 are located in position 1 and 2. The second element contains node 4, 3 and 1. According to Table 1, edge 1 goes from the 1st position to the 2^{nd}. Therefore, edge 1 of element 2 goes from node 4 to node 3. Edge 2 of element 2 goes from position 2 to position 3 (from node 3 to node 1) and finally, edge 3 of element 2 goes from position 3 to position 1 (from node 1 to node 4). The same algorithm is applied to the 1st element and can be checked in Fig. 8.

Three Dimensions:

For the 3D case, the global rule is the same as in 2D. That means that it uses the node's index given when meshing and order them from the lowest possible to the second lowest possible.

On the other hand, the local rule changes due to the fact that now there are 4 nodes per tetrahedron instead of the 3 of the triangle. The local rule used is given in Table 3. This local indexing rule can be observed in Fig. 9 with a single element example.

2.7 Formulation of the problem. A and b formulas

The formulation of FEM requires a large amount of time and it is out of the scope of this thesis. Here, the equations to be solved are introduced and the interested reader is referred to chapter 8 of [21] for details on the formulation. Only the edge elements formulation will be explained.

Table 3: Local indexing for 3D tetrahedra elements.

$Edge_i$	$node_{initial}$	$node_{end}$
1	1	2
2	1	3
3	1	4
4	2	3
5	4	2
6	3	4

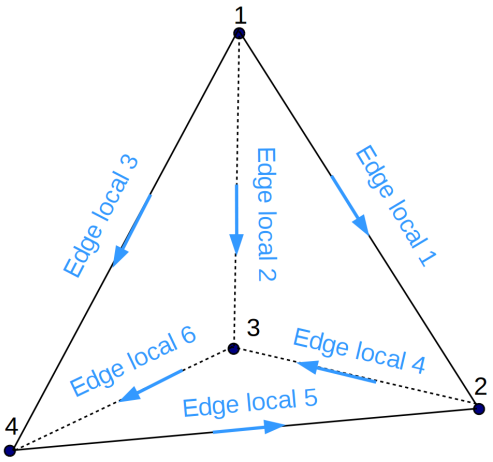

Figure 9: Local indexing rule example in a single tetrahedron.

The best way to understand the formulation is by an example. The example used will be the FEM formulation of the electromagnetic differential equation that PETGEM solves which is:

$$(\nabla \times \nabla \times + iw\mu\sigma)\,\mathbf{E}_s = -iw\mu\,\triangle\sigma\,\mathbf{E}_p \tag{36}$$

The derivation of this formula and the interpretation of its parameters will be given in Section 3). The FEM formulation of this differential equation is given by:

$$\underbrace{(K^e_{jk} + iw\mu\sigma M^e_{jk})}_{A}\underbrace{\{E^e_k\}}_{x} = \underbrace{-iw\mu\triangle\sigma\{R^e_k\}}_{b}, \tag{37}$$

where $\{E_{sk}\}$ is the vector of the DOFs, $\{R^e_k\}$ is a vector that is integrated numerically, and K^e_{jk} and M^e_{jk} are called the Stiffness and the Mass matrix respectively. They have different forms for the 2D and the 3D case as discussed in the following two paragraphs.

Two dimensional formulation:

In 2 dimensions, the stiffness and mass matrices are given by:

$$\begin{aligned} K^e_{ij} &= \iint_{w_e} (\nabla \times \mathbf{N^e_i}) \cdot (\nabla \times \mathbf{N^e_j})\, dS \\ M^e_{ij} &= \iint_{w_e} \mathbf{N^e_i} \cdot \mathbf{N^e_j}\, dS. \end{aligned} \tag{38}$$

It is very important to note that the FEM formulation integrates the operators of the differential equation all over the domain. These two integrals can be computed analytically for rectangular and triangular elements. However, they need to be computed numerically for quadrilateral elements. For the triangular discretizations, these two integrals are computed by:

$$K_{ij}^e = \frac{l_i^e l_j^e}{\triangle^e} \tag{39}$$

$$\begin{aligned}
M_{11}^e &= \frac{(l_1^e)^2}{24\triangle^e}(m_{22} - m_{12} + m_{11}) \\
M_{22}^e &= \frac{(l_2^e)^2}{24\triangle^e}(m_{33} - m_{23} + m_{22}) \\
M_{33}^e &= \frac{(l_3^e)^2}{24\triangle^e}(m_{11} - m_{13} + m_{33}) \\
M_{21}^e = M_{12}^e &= \frac{l_1^e l_2^e}{48\triangle^e}(m_{23} - m_{22} - 2m_{13} + m_{12}) \\
M_{31}^e = M_{13}^e &= \frac{l_1^e l_3^e}{48\triangle^e}(m_{21} - m_{11} - 2m_{23} + m_{13}) \\
M_{32}^e = M_{23}^e &= \frac{l_2^e l_3^e}{48\triangle^e}(m_{31} - m_{33} - 2m_{12} + m_{23}) \ ,
\end{aligned} \tag{40}$$

where $m_{ij} = b_i^e b_j^e + c_i^e c_j^e$. It is important to remark that these matrices are obtained for each element e. Since the discretization is triangular, there are three edges per element and therefore three DOFs per element. Consequently, the matrices are 3x3. The global matrix A is $n \,\mathrm{x}\, n$ where n is the total number of DOFs in the whole mesh. The computation of the global matrix A is not trivial since the positions of the elements' matrices must be allocated in the global matrix. The discussion of the building of the matrix A and the vector b will be given in the subsection 2.8.

Three dimensional formulation

Equivalent to the 2D case, in 3D the stiffness and mass matrices are given by:

$$\begin{aligned}
K_{ij}^e &= \iiint_{w_e} (\nabla \times \mathbf{N_i^e}) \cdot (\nabla \times \mathbf{N_j^e}) \, dV \\
M_{ij}^e &= \iiint_{w_e} \mathbf{N_i^e} \cdot \mathbf{N_j^e} \, dV.
\end{aligned} \tag{41}$$

For the tetrahedral elements that will be used in this thesis, the stiffness matrix is given by:

$$\begin{aligned}
K_{ij}^e = \frac{4 l_i^e l_j^e V^e}{(6V^e)^4}[&(c_{i1}^e d_{i2}^e - c_{i2}^e d_{i1}^e)(c_{j1}^e d_{j2}^e - c_{j2}^e d_{j1}^e) \\
&(d_{i1}^e b_{i2}^e - d_{i2}^e b_{i1}^e)(d_{j1}^e b_{j2}^e - d_{j2}^e b_{j1}^e)(b_{i1}^e c_{i2}^e - b_{i2}^e c_{i1}^e)(b_{j1}^e c_{j2}^e - b_{j2}^e c_{j1}^e)].
\end{aligned} \tag{42}$$

The mass matrix for tetrahedral elements is now 6x6 since there are six edges (DOFs) per element. There is no general formula to describe all 36 terms of the mass matrix but it could be defined with 21 formulas. Those 21 formulas to describe M_{ij}^e will not be written in this thesis because it is out of the scope but they can be found in page 301 of [21].

2.8 Assembly of A and b

This subsection explains the assembly of the global matrix A_{ij} from the Stiffness K^e_{ij} and Mass M^e_{ij} matrices for each element; and the building of global array b from the numerical integration of R^e_k. Both the building of A and b rely on the indexing rules that have been set before. For a better understanding, the explanation will be started with the building of b. Moreover, all explanations will be only focused on the 2D case for its simplicity. The 3D case can be extracted from the 2D.

b array

It is common to use an initial profile when solving a differential equation in FEM. This is the case of PETGEM's main equation:

$$(\nabla \times \nabla \times + iw\mu(\sigma - \sigma_p)) \underbrace{\mathbf{E}_s}_{\text{Result of the computation}} = -iw\mu\,(\sigma - \sigma_p) \underbrace{\mathbf{E}_p}_{\text{Initial Profile}}$$

The initial profile is always contained in the b array. As K^e_{ij} and M^e_{ij} are the operators integrated over the whole domain, R^e_k of Eq. 36 is the integral over the whole domain of $\mathbf{E}_p$. In fact, R^e_k is given by:

$$R^e_k = \iint_{w^e} \mathbf{N^e_k} \cdot \mathbf{E}_p. \tag{43}$$

This integral can sometimes be computed analytically depending on the shape of $\mathbf{E_p}$. However, for a general purpose, this integral will be done numerically using Gaussian quadrature [22]. The main idea behind Gaussian quadrature is to compute an integral of a function as a sum of this function evaluated in some points multiplied by some weights. The choice of points to use and their weights rely on the shape of the integral domain. In this 2D case, the integrals are over triangles. The repository [22] provides a large set of Gaussian points locations ($\mathbf{r}_i$), it's weights ($weight(i)$) and the formula to transform the locations from a unit triangle to any arbitrary triangle. Introducing the Gaussian quadrature approximation, R^e_k becomes:

$$R^e_k = \iint_{w^e} \mathbf{N^e_k} \cdot \mathbf{E}_p \simeq \sum_{i=1}^{\#\text{ Guass Points}} weight(i)\, \mathbf{N^e_k}(\mathbf{r}_i) \cdot \mathbf{E}_p(\mathbf{r}_i) \tag{44}$$

Once these computations are finished, R^e_k is obtained where e is the index of the element and $k = 1, 2, 3$ is the index of the DOF (edge). Following the specific problem that is being solved, it is defined $b^e_k \equiv -iw\mu\,(\sigma - \sigma_p) R^e_k$.

Now, b^e_k can be computed for all e and k. However this elements must be allocated in the global array b. In order to get a better understanding of the allocation process, the simple example used in the indexing section (Fig. 8) will be used as well. The allocation process depends on the local indexing and, also, on a variable not mentioned yet called EdgesN. This variable and elemsN (Eq. 35) are part obtained on the Preprocessing of the Mesh as it will be explained later. The information needed to allocate this specific problem is contained in Fig. 10:

This figure exemplifies the allocation process of b^e_k in the global array b. It is important to remark that since the mesh is based on 2 elements and 5 edges, the global array b will

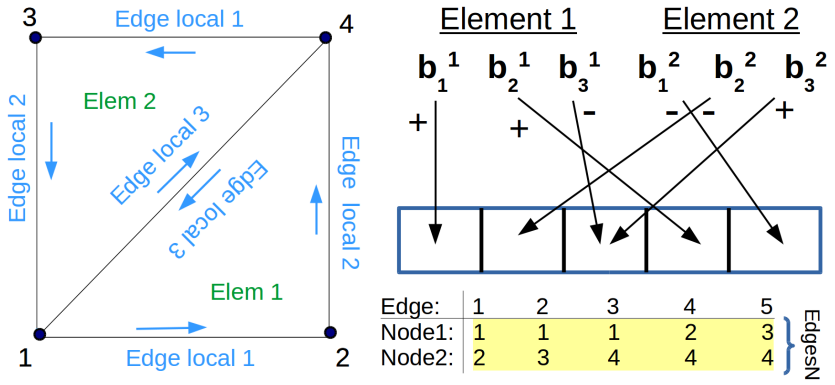

Figure 10: On the left side, the local indexing of the easy example. On the right side, the allocation process with the matrix edgesN highlighted in yellow.

have 5 components. This is a general rule, the length of b is the number of DOFs. Starting from the 1^{st} element, the left figure shows that the first edge goes from node 1 to node 2. Looking on the edgesN matrix, from 1 to 2 corresponds to the global edge 1. Therefore, b_1^1 goes with a + sign to the first position of the global array b. The + sign is due to the fact that both the local and the global indexing go from 1 to 2. If one went from 1 to 2 and the other from 2 to 1 it would correspond a - sign. The second edge of the first element goes locally from 2 to 4 and it corresponds in the matrix edgesN to the 4^{th} position. Therefore, b_2^1 goes to the 4^{th} position of b with a + sign. The 3^{rd} edge of the 1^{st} element goes from node 4 to node 1. The 3^{rd} position of edgesN goes from 1 to 4. In this case, one is from 1 to 4 and the other is from 4 to 1. Therefore, b_3^1 goes to the the 3^{rd} position of b with a - sign. Once the first element is finished, it is time for the second and last element. Its 1^{st} edge goes from 4 to 3 and the 5^{th} position of edgesN goes from 3 to 4. Therefore, b_1^2 goes to the 5^{th} position with a - sign. The second edge goes from 3 to 1 that corresponds to the second position of edgesN that is from 1 to 3. Therefore, b_2^2 goes to the 2^{nd} position with a - sign. Finally, edge 3 goes from 1 to 4 as the 3^{rd} position of edgesN. Therefore, b_3^2 is added with a + sign to the $-b_3^1$ that was already in the 3^{rd} position of edgesN.

Matrix A

To sum up, the equation that follows the stiffness matrix (K_{ij}^e) for 2D is in Eq. 39 and in 3D in Eq. 41. Moreover, the mass matrix (M_{ij}^e) follows in 2D Eq. 40 and in 3D is given in page 301 of [21]. Once K_{ij}^e and M_{ij}^e can be computed for both 2D and 3D, A_{ij}^e is obtained using Eq. 45:

$$A_{ij}^e = K_{ij}^e + iw\mu(\sigma - \sigma_p)M_{ij}^e \tag{45}$$

Once A_{ij}^e is known for every element, it is time to allocate this matrices in the global matrix A that is a squared matrix with dimension number of edges (DOFs). As happened with the assembly of b, the signs of the elements must be taken into account. In the assembly of A is easier to include the signs just before the allocation and to do so it is required the function Signs given in Eq. 46

$$S_i^e = \frac{node_{i2}^e - node_{i1}^e}{|node_{i2}^e - node_{i1}^e|} \tag{46}$$

where i is the edge index that joins $node_{i1}^e$ with $node_{i1}^e$ of the element e. Afterwards, a_{ij}^e

is defined as the element that will be allocated containing the signs analysis that is given by:

$$a_{ij}^{e} = A_{ij}^{e} S_i^e S_j^e \tag{47}$$

In order to allocate a_{ij}^{e}, a matrix called connectivity (will be shorted to "elemsE") is needed. This matrix specifies which edges belong to each element and it is very similar to the matrix elemsN that was introduced before in Eq. 35. The computation of "elemsE" is out of the scope of this thesis but for the two-triangles example, this matrix is:

$$\text{elemsE} = \begin{array}{c} \\ \text{position 1} \\ \text{position 2} \\ \text{position 3} \end{array} \begin{array}{c} \begin{array}{cc} \text{Elem1} & \text{Elem 2} \end{array} \\ \begin{pmatrix} 1 & 5 \\ 4 & 2 \\ 3 & 3 \end{pmatrix} \end{array} \tag{48}$$

Firstly, the general rule will be given and then it will be exemplified using the two-triangles example. Defining the column e of the matrix "elemsE" as (k, l, m), then the coefficient a_{ij}^{e} is allocated following the rule:

$$\begin{array}{ccc} a_{11} \rightarrow (k,k) & a_{12} \rightarrow (k,l) & a_{11} \rightarrow (k,m) \\ a_{21} \rightarrow (l,k) & a_{22} \rightarrow (l,l) & a_{23} \rightarrow (l,m) \\ a_{31} \rightarrow (m,k) & a_{32} \rightarrow (m,l) & a_{33} \rightarrow (m,m) \end{array}$$

Assuming that for this example the matrices a^1 and a^2 for each element are given by:

$$a^1 = \begin{pmatrix} \lambda_{11} & \lambda_{12} & \lambda_{13} \\ \lambda_{21} & \lambda_{22} & \lambda_{23} \\ \lambda_{31} & \lambda_{32} & \lambda_{33} \end{pmatrix} \quad a^2 = \begin{pmatrix} \Lambda_{11} & \Lambda_{12} & \Lambda_{13} \\ \Lambda_{21} & \Lambda_{22} & \Lambda_{23} \\ \Lambda_{31} & \Lambda_{32} & \Lambda_{33} \end{pmatrix} \tag{49}$$

The column 1 of "elemsE" is $(1, 4, 3)$ and the second column is $(5, 2, 3)$. Therefore, the allocation of this problem is the following one:

$$A = \begin{pmatrix} \lambda_{11} & 0 & \lambda_{13} & \lambda_{12} & 0 \\ 0 & \Lambda_{22} & \Lambda_{23} & 0 & \Lambda_{21} \\ \lambda_{31} & \Lambda_{32} & \lambda_{33} + \Lambda_{33} & \lambda_{32} & \Lambda_{31} \\ \lambda_{21} & 0 & \lambda_{23} & \lambda_{22} & 0 \\ 0 & \Lambda_{12} & \Lambda_{13} & 0 & \Lambda_{11} \end{pmatrix} \tag{50}$$

Knowing A and b, one can solve the equation $A \cdot x = b$ by inverting the matrix A: $x = A^{-1}b$. The most important requirement is that the determinant of a matrix must be different to zero in order to be able to invert that matrix ($det(A) \neq 0$). There are many different methods to numerically invert a matrix and which one to use depends on the matrix itself. However, this topic is out of the scope of the thesis and no further information will be addressed.

3 PETGEM

3.1 Brief description of PETGEM

The Parallel Edge-based Tool for Geophysical Electromagnetic Modelling (PETGEM) is a Python HPC scalable tool based on the finite element method. This code contains more than 8000 lines with many different modules such as the preprocessing, the assembly, the solver, the postprocessing, ... PETGEM has been developed as open-source (under GPLv3 license) at the Department of Computer Applications in Science & Engineering (CASE) of the Barcelona Supercomputing Center (BSC). PETGEM solves the marine Controlled-Source Electromagnetic method (CSEM) which is an important technique for reducing ambiguities in data interpretation for hydrocarbon exploration. In order to solve CSEM, one must solve Maxwell's equations 51 where the harmonic time dependence e^{-iwt} is omitted.

$$\nabla \times \mathbf{E} = iw\mu_0 \mathbf{H} \qquad \nabla \times \mathbf{H} = \mathbf{J}_s + \sigma \mathbf{E}\,, \tag{51}$$

where w is the frequency, σ the conductivity and $\mathbf{J}_s$ the distribution of source current which is considered a punctual source. Using a perturbation approach, the total electric field is written as a sum of a primary field (the known term) and a secondary field (the part that will be computed) $\mathbf{E} = \mathbf{E}_s + \mathbf{E}_p$. For a general layered Earth model, E_p is computed using a Hankel transform filters [23].

The same procedure can be applied to electric conductivity. Following the common geophysics notation, $\sigma = \sigma_s + \triangle\sigma$, where $\triangle\sigma$ is the electrical conductivity of the area where the focus is located. Therefore, σ_s will change along the sediments. On the focus zone it will be 0 and on the other zones will be given by $\sigma_s = \sigma_{real} - \triangle\sigma$. One can get the final equation to solve:

$$\nabla \times \nabla \times \mathbf{E}_s + iw\mu_0\sigma\mathbf{E}_s = -iw\mu_0 \triangle\sigma\ \mathbf{E}_p \tag{52}$$

PETGEM is an HPC code due to the fact that edge elements offer a good scalability and it is exploited through the Python Package Petsc4py [24].

3.2 How to use PETGEM

In order to install PETGEM, one must follow the guide [25]. On the following paragraphs, a guide to understand PETGEM is given. Those instructions are the result of the learning process of the multiple simulations carried out using this code. The tutorial is divided in: preprocessing, kernel and visualization of the output.

3.2.1 Preprocessing

In order to run the preprocessing, one needs to prepare a file called "preprocessingParams.py". An example of this file is found below:

```
preprocessing = {
    # ---------- Mesh file ----------
    'MESH_FILE': '/home/marc/Dropbox/BSC/gmsh/cube/cube.msh',

    # ---------- Material conductivities ----------
    'MATERIAL_CONDUCTIVITIES': [1.0],
```

```
    # ---------- Receivers position file ----------
    'RECEIVERS_FILE': '/home/marc/Dropbox/BSC/gmsh/cube/cube.txt',

    # ---------- Path for Output ----------
    'OUT_DIR': '/home/marc/petgem-0.30.43/PreprocessingOutput/',
}
```

The four elements needed are now introduced. The first one is the mesh file which has already been explained how to create it. The second element is the material conductivities. This part of the code is important to solve geophysical problems where there are different layers with different conductivities but for this thesis, the plasma will be considered to have a homogeneous conductivity of 1.0 from now on. The third option is the Receivers file which must be a txt containing the X, Y and Z coordinates of each element separated by a tab. The following example contains just 3 receivers:

```
0.000       0.000       0.000
1.000       0.000       0.000
0.000       1.000       0.000
```

On appendix C there are two Matlab algorithms to create homogeneous distributions of receivers for both a cube and a cylinder. Finally, the last variable needed is the output directory where there will be stored the code output's.

Once the file "preprocessingParams.py" is ready, it is time to execute the file "run_ preprocessing.py". To do so, one must execute on the computer's console:

```
python3 /...../run_preprocessing.py /...../preprocessingParams.py
```

where /...../ refers to specify the folder where those files are.

3.2.2 Kernel

The file "kernel.py" also requires of two auxiliary parameters files called "pets.ops" and "modelParams.py". The first one contains parameters of the solution which are out of the scope of this thesis but further information can be found on [26]. The second file is very similar to "preprocessingParams.py". It contains variables that will be used in the computations. Those variables can be separated into two groups: the ones that define E_p and the ones related to the mesh. The elements in the second subgroup are the output of the preprocessing. Finally, to run the kernel one must write on the console:

```
mpirun -n X python3 /...../kernel.py -options_file /...../petsc.opts
    /...../modelParams.py
```

where /...../ refers to specify the folder where those files are and X the numbers of cores where the job will be parallelized.

3.2.3 Visualization

PETGEM does not give the visualization by itself. However, it generates output files with the electric responses at receivers positions available in the formats ASCII, PETSc and Matlab. Appendix C contains a simple Matlab script created in this thesis work in order to view the output of PETGEM.

4 Results

The thesis work consists of following four steps: to understand and learn EFEM theory, to understand the workflow and the scripts of PETGEM code, to check the capability to implement different initial profiles to PETGEM and to change the equations that PETGEM solves and introducing the time integration of the equations as summarized in this chapter:

Learn EFEM → Understanding PETGEM → Check initial profile → Change equations

The most time demanding part of this thesis is understanding and being able to modify PETGEM because it is an HPC code with more than 8000 lines and many different modules such as the preprocessing, the assembly, the solver, the postprocessing, ... That is why this current thesis has stopped at the changing equations steps, where some research has been done but none remarkable result of changing the equations has been achieved. However, there has been many important results and achievements that will be described in the following lines.

4.1 FEM Mesh Creation

The finite element method mesh must be created using Gmsh [27]. This software is open-source and it can be used both with its graphic interface or via script-console. A 2D example of a rectangular mesh generation using Gmsh is given in the following lines.

```
lc=0.1; //radius length of vertex

//Point(i) = {x, y, z, radius of the vertex};
Point(1) = {0, 0, 0, lc};
Point(2) = {1, 0, 0, lc} ;
Point(3) = {1, 3, 0, lc} ;
Point(4) = {0, 3, 0, lc} ;

//Line(k) = {i,j}; From point i to j.
Line(1) = {1,2} ;
Line(2) = {3,2} ;
Line(3) = {3,4} ;
Line(4) = {4,1} ;

//Line Loop Closes the area.
Line Loop(1) = {4,1,-2,3} ;
//minus since line2 is 3-2 and not 2-3
Plane Surface(1) = {1} ;
```

This code only creates the geometry to be divided in elements. To create the mesh click on Modules>Mesh>Define>2D (or 3D for those geometries). By default, this configuration creates meshes with triangular, hexahedra, pyramid, ... elements. In order to make the mesh only with a specific type of elements, one must go to Tools>Options>Mesh>elements and unselect all except the one wanted.

As it has been mentioned before, the refinement process is an important step to optimize this method. Gmsh documentation provides a detailed tutorial on different strategies to refine a mesh [28]. Appendix A provides an example of refinements over a point, a line

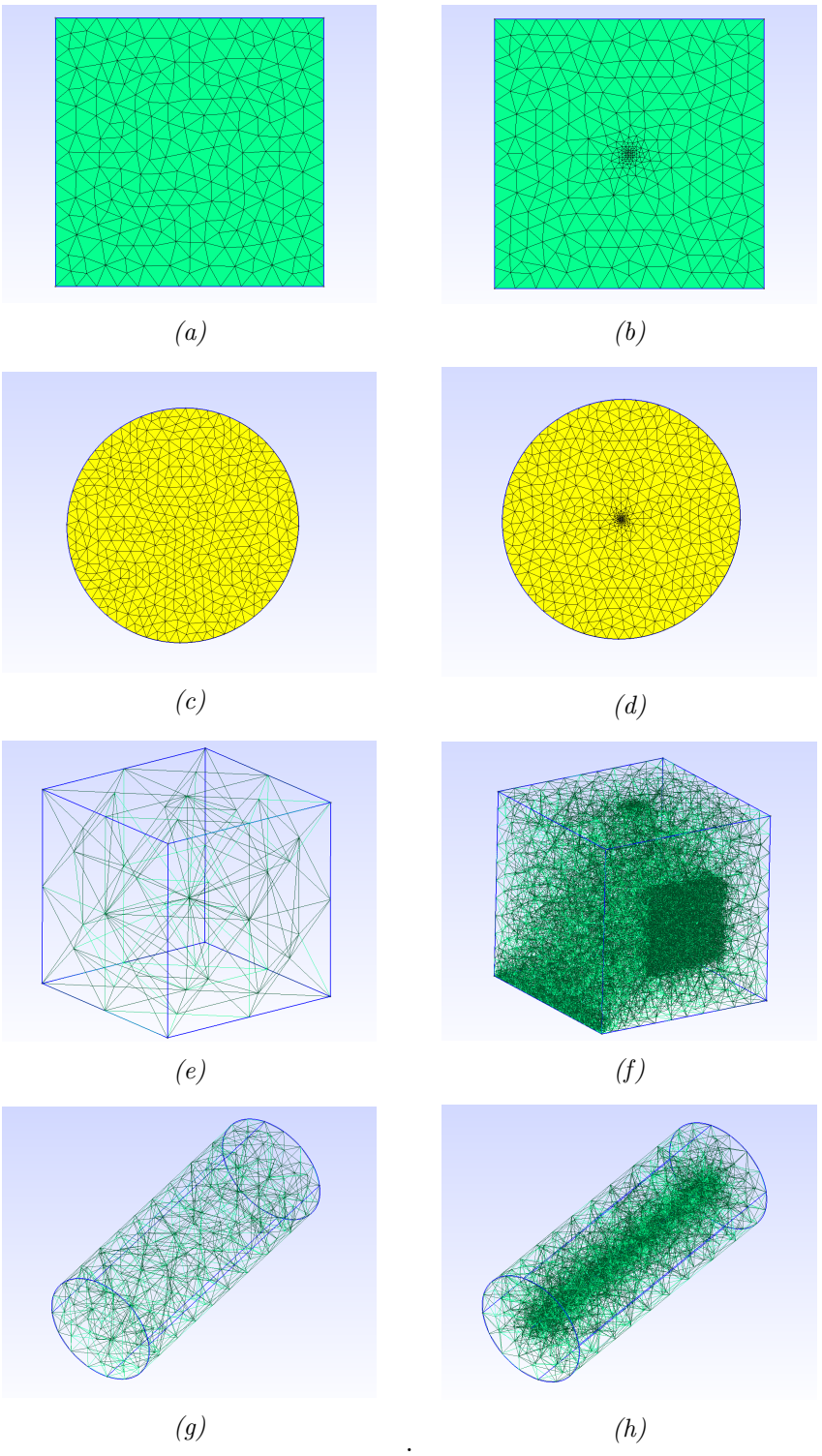

Figure 11: Set of different geometries obtained using Gmsh. The left panels are the homogeneous meshes while the right ones are the refined meshes.

and a box. On https://github.com/marcfusterr/Gmsh_Scripts there are uploaded different codes to generate 2D geometries (circle and square) and 3D geometries (cylinder and cube) developed in this thesis work. All codes have both a homogeneous and a refined mesh. Figure 11 shows two cases of the distribution of the mesh density; homogeneous and

inhomogeneous in the simulation volume. They are plotted from PETGEM output data using the Gmsh graphic interface. The work contributes to the visualization of the data using Paraview [29] so that the GUI analysis can be made directly from the visualization. The capability of the arbitral choice of the mesh density concentration cases the simulations according to the physics problems to be solved.

4.2 Implementation of initial profile

Before studying the feasibility of applying PETGEM to plasma physics, the ability to implement initial profiles to PETGEM must be checked. This subsection contains a benchmark of the implementation of an initial profile and also explains how to get the velocity field from the velocity potential.

4.2.1 Benchmark

The implementation of an initial profile is not a trivial task in PETGEM. That is because PETGEM always uses the same initial profile: the electric field created from CSEM antenna using the Hankel filters mentioned before. Therefore, the code is not prepared for the initial profile to be changed. Changing the initial profile in PETGEM requires introducing the function to be implemented in both the "postprocessing.py" and "assembler.py". Moreover, the choice of an initial profile must go with a choice of a good set of receivers in order to correctly appreciate the output.

The initial profile $\vec{u} = [-2sin(y), 2sin(x), 0]$ has been selected as a benchmark to check if the initial profiles are well implemented as shown in Fig. 12. The flow of the vector field of $\vec{u}$ shows the rotative dynamics of the fluid.

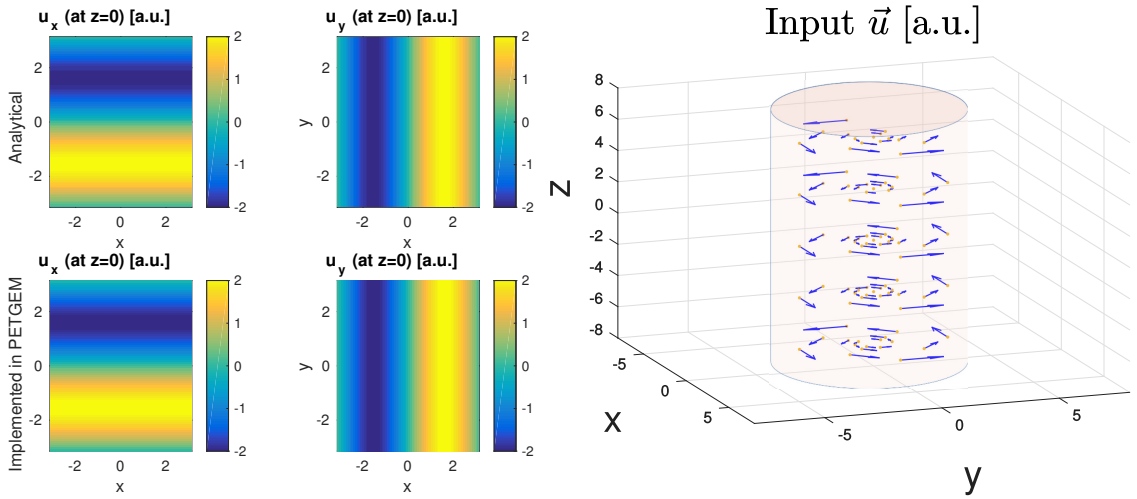

Figure 12: On the left side: Top panels show the analytical function $\vec{u}$. Bottom panels show the implemented function obtained using PETGEM's output data. On the right side: alternative view of the initial profile $\vec{u} = [-2sin(y), 2sin(x), 0]$ from PETGEM's output data.

4.2.2 Initial electrical field for magnetically confined plasma.

When the electric field is conservative, the electric field is determined by the gradient of the electrostatic potential, i.e. $\vec{E} = -\nabla\phi$. Therefore, a script has been created that automatically obtains $\vec{E}$ from any analytical ϕ by computing the gradient analytically

using the Python package Sympy [30]. The cylindrical geometry is used in mirror plasma confinement devices such as SLPM [31], PANTA [32], etc. The plasma confined in the cylindrical geometry is a useful approach to study the plasma instabilities. The density of the simulation grid can be arbitrarily chosen in the simulation domain. The location of the mesh concentration can be chosen depending on the physics problem to be aimed to study. Figure 13 shows the electric field obtained from $\phi = exp(-(x/1.5)^2 - (y/1.5)^2))$ using the automatic gradient script.

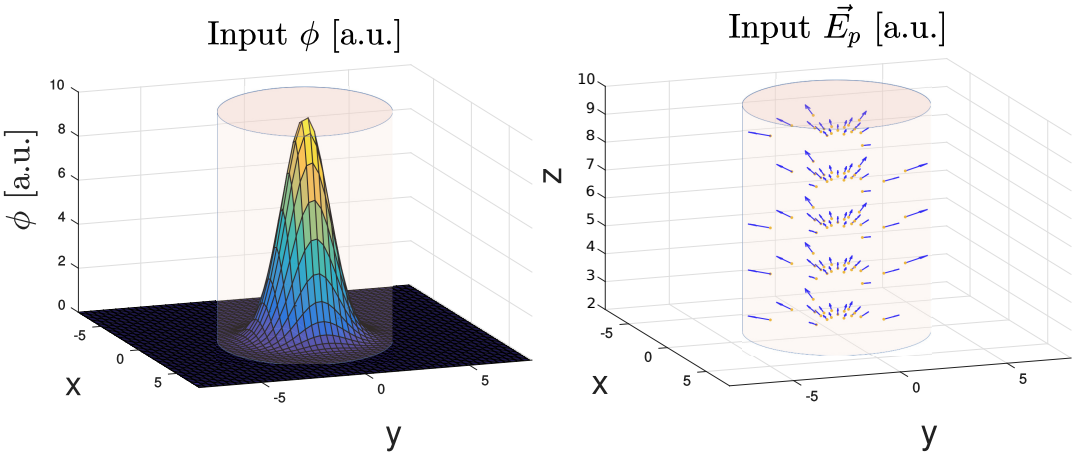

Figure 13: On the right hand, output of the automatic gradient script obtained from the electric potential $\phi = exp(-(x/1.5)^2 - (y/1.5)^2)$ which is plotted on the left part.

4.3 Steady state plasma

The full wave equation in a magnetically confined plasma with an antenna current as a boundary condition is given by Eq. 53:

$$\begin{aligned} \nabla \times \nabla \times \mathbf{E} - \frac{w^2}{c^2}\mathbf{E} &= \frac{4\pi w i}{c^2}\left(\mathbf{J^p} + \mathbf{J^a}\right) \\ \mathbf{J^p}(\mathbf{r}) &= \int d\mathbf{r}'\sigma(\mathbf{r},\mathbf{r}')\mathbf{E}(\mathbf{r}') \end{aligned} \tag{53}$$

where $\mathbf{J^a}$ is the current of the antenna and σ is the conductivity tensor. This equation has the same shape as the one that PETGEM solves. However, this equation is the wave equation with a boundary condition corresponding to the current density introduced by the antenna while Eq. 52 is the perturbation solution of a wave propagating in the earth. Figure 14 shows the initial profile of the current density $\mathbf{J} = \mathbf{J^p} + \mathbf{J^a} = \hat{\mathbf{z}}/f(r)$ along with it's electrical response, where $f(r) = (1 + (r/a)^{2\Lambda})^{1+1/\Lambda}$ and $a = 0.25$m is the radius of the cylinder and $\Lambda = 4$ [33]. The implemented current profile is a reasonable approach to compute the initial profile of the current density and the electric field for fusion plasmas [34]. The time variation of the current density will be implemented in the future in order to investigate the interactions between the effect of the antenna and the plasma response. The output $\vec{E}$ which is solved from $\vec{J}$ can be obtained by PETGEM calculation. However the initial profile implemented here is time-independent quantity, and it does not have a function of the coefficient of the time-frequency. The PETGEM output $\vec{E}$ shows zero (or numerical noise). It is as expected and it confirms the implementation and the PETGEM calculation have been performed correctly without generating any unexpected errors.

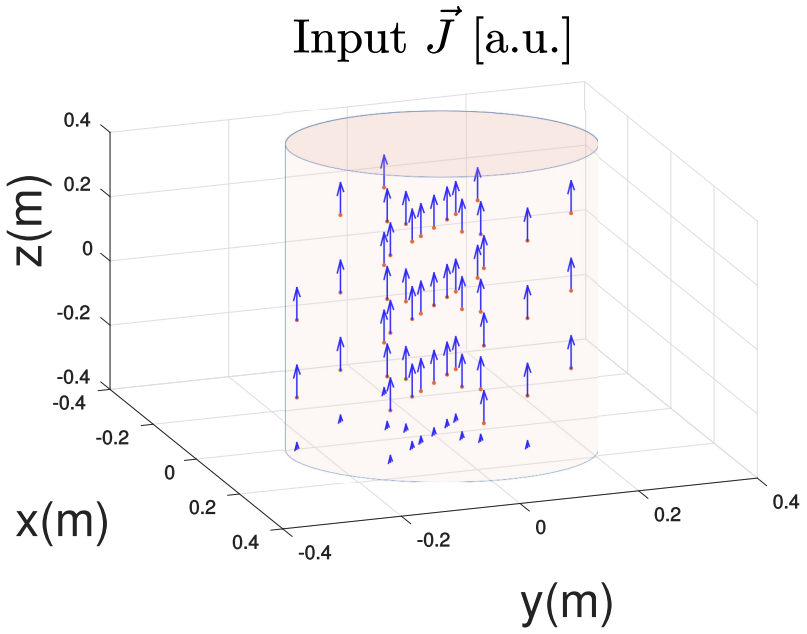

Figure 14: Initial profile of the antenna current density to be solved by the full wave equation.

5 Conclusions

The study of the assessment of this new research line of the EFEM application to fusion plasma physics has been successfully carried out. Due to the complex dynamics of the plasma, high performance simulations are required. Therefore, this work demonstrates the capability to perform and modify a large parallel code in the supercomputer Marenostrum. The edge finite element method theory has been studied and has been found to be more useful than the nodal finite element method for electromagnetic problems and therefore, for fusion problems.

Moreover, this work demonstrates the generation of the simulation mesh for finite element method for fusion plasma in the different geometries such as squares, circles, cubes and cylinders. The location and the density of the mesh concentration can be arbitrarily adopted according to the physics problem which is aimed to investigate in a user-friendly manner using the codes that have been uploaded to the Github link https://github.com/marcfusterr/Gmsh_Scripts. The implementation of the initial field of any quantities such as electric field, and the application of the reasonable current profile which is specifically aimed for the fusion plasma research have been carried out.

6 Future Work

The future steps of the work are to go beyond the calculation of the steady state plasma and implement the time integration of the plasma dynamics. The long-term objective is to solve the magnetohydrodynamic (MHD) which is the combination of the electromagnetism system i.e. Maxwell's equations and the fluid system i.e. Navier-Stokes equation. The primitive approach is to develop the fluid modeling part considering an incompressible ($\nabla \cdot \mathbf{u} = 0$), diffusive model: $\frac{\partial \mathbf{u}}{\partial t} = -\nu \nabla \times \nabla \times \mathbf{u}$ which is ongoing to be implemented in PETGEM. Once the diffusive model is finished, Navier-Stokes equation should be implemented and, afterward, MHD equations. The development team of PETGEM is currently introducing a more complex numerical scheme, the higher order elements. This scheme consists of polynomial interpolation functions rather than linear interpolation functions. Therefore, an important future step is to adapt all the work done to the new PETGEM scheme.

References

[1] I.P. Mysovskikh. Trapezium formula. https://www.encyclopediaofmath.org/index.php/Trapezium_formula.

[2] I.P. Mysovskikh. Romberg method. https://www.encyclopediaofmath.org/index.php/Romberg_method.

[3] http://ourfiniteworld.com/2012/03/12/world-energy-consumption-since-1820/.

[4] http://nupex.eu/index.php?g=textcontent/nuclearenergy/nuclearfusion&lang=en.

[5] F.F. Chen. *Introduction to Plasma Physics and Controlled Fusion*. Number v. 1 in Introduction to Plasma Physics and Controlled Fusion. Springer, 1984.

[6] E. Stuhlinger. *Ion propulsion for space flight*. McGraw-Hill New York, 1964.

[7] I-H. Loh and M.S. Sheu. Plasma surface modification in biomedical applications. *MRS Online Proceedings Library Archive*, 414, 1995.

[8] V.L. Bonch-Bruevich and Sh.M. Kogan. The theory of electron plasma in semiconductors. *Soviet Phys.-Solid State*, 1, 1960.

[9] K. Miyamoto. *Plasma physics for nuclear fusion*. MIT Press, 1989.

[10] B.D. Bondarenko. Role played by o a lavrent'ev in the formulation of the problem and the initiation of research into controlled nuclear fusion in the ussr. *Physics-Uspekhi*, 44(8):844, 2001.

[11] http://www.sciencedirect.com/science/article/pii/S2468080X1630032.

[12] E. Hill. Physics of fusion power lecture 6: Conserved quantities / mirror device / tokamak. http://slideplayer.com/slide/4645450/.

[13] Iter webpage. https://www.iter.org/.

[14] G. Huijsmans and O. Czarny. Mhd stability in x-point geometry: Simulation of elms. 47:659, 06 2007.

[15] L.G. Eriksson et al. Comparison of time dependent simulations with experiments in ion cyclotron heated plasmas. *Nuclear Fusion*, 33(7):1037, 1993.

[16] T. Kurki-Suonio et al. Ascot simulations of fast ion power loads to the plasma-facing components in iter. *Nuclear Fusion*, 49(9):095001, 2009.

[17] R. Kleiber and R. Hatzky. Euterpe gyrokinetic code. http://fusionwiki.ciemat.es/wiki/EUTERPE.

[18] O. Castillo-Reyes. Parallel edge-based tool for geophysical electromagnetic modelling (petgem). http://petgem.bsc.es/.

[19] R. Nicoletti. Comparison between a meshless method and the finite difference method for solving the reynolds equation in finite bearings. 2013;135(4):044501-044501-9. doi:10.1115/1.4024752. *ASME. J. Tribol.*

[20] G. Mur. Edge elements, their advantages and their disadvantages. *IEEE Transactions on Magnetics, 30 (5)*, 30, 09 1994.

[21] J.M. Jin. *The finite element method in electromagnetics*. John Wiley & Sons, 2015.

[22] Repository of quadrature rules for triangles. https://people.sc.fsu.edu/~jburkardt/datasets/quadrature_rules_tri/quadrature_rules_tri.html.

[23] F.N. Kong. Hankel transform filters for dipole antenna radiation in a conductive medium. *Geophysical Prospecting*, 55(1):83–89, 1960.

[24] L.D. Dalcin et. al. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124 – 1139, 2011. New Computational Methods and Software Tools.

[25] O. Castillo-Reyes. Petgem installation guide. https://pypi.org/project/petgem/.

[26] S. Balay et al. PETSc Web page. http://www.mcs.anl.gov/petsc, 2017.

[27] C. Geuzaine and J-F. Remacle. Gmsh. http://gmsh.info/.

[28] C. Geuzaine. Gmsh refining tutorial. https://gitlab.onelab.info/gmsh/gmsh/blob/master/tutorial/t10.geo.

[29] A. Henderson. Paraview guide, a parallel visualization application. *Kitware Inc.*, 2007.

[30] A. Meurer. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[31] F. Castellanos et al. Parallel flows and turbulence in a linear plasma machine. *Plasma Physics and Controlled Fusion*, 47(11):2067, 2005.

[32] S. Inagaki et al. A concept of cross-ferroic plasma turbulence. *Scientific Reports*, 6:22189, 02 2016.

[33] X. Shan and D. Montgomery. On the role of the hartmann number in magnetohydrodynamic activity. *Plasma Physics and Controlled Fusion*, 35(5):619, 1993.

[34] S. Futatani J. Morales and W.J.T. Bos. Dynamic equilibria and magnetohydrodynamic instabilities in toroidal plasmas with non-uniform transport coefficients. *Physics of Plasmas*, 22:052503.

A 3D Mesh creation and Refinement

This code summarizes how to create geometries (points, lines, surfaces and volumes) and how to create an inhomogeneous mesh by refining over points, lines, surfaces and volumes. With those basic geometries, one can adapt them to create different objects.

```
lc=0.1; //radius length of vertex

// lower surface
Point(1) = {0, 0, 0, lc};
Point(2) = {1, 0, 0, lc} ;
Point(3) = {0, 1, 0, lc} ;
Point(4) = {1, 1, 0, lc} ;

Line(1) = {1,2} ;
Line(2) = {2,4} ;
Line(3) = {4,3} ;
Line(4) = {3,1} ;

Line Loop(1) = {1,2,3,4} ;
// minus since line2 is 3-2 and not 2-3
Plane Surface(1) = {1} ;

//uper surface
Point(5) = {0, 0, 1, lc};
Point(6) = {1, 0, 1, lc} ;
Point(7) = {0, 1, 1, lc} ;
Point(8) = {1, 1, 1, lc} ;

Line(5) = {5,6} ;
Line(6) = {6,8} ;
Line(7) = {8,7} ;
Line(8) = {7,5} ;

Line Loop(2) = {5,6,7,8} ;
// minus since line2 is 3-2 and not 2-3
Plane Surface(2) = {2} ;


//join both surface

Line(9) = {1,5} ;
Line(10) = {2,6} ;
Line(11) = {3,7} ;
Line(12) = {4,8} ;

Line Loop(3) = {9,5,-10,-1}; Plane Surface(3) = {3};
Line Loop(4) = {2,12,-6,-10}; Plane Surface(4) = {4};
Line Loop(5) = {3,11,-7,-12}; Plane Surface(5) = {5};
Line Loop(6) = {-4,11,8,-9}; Plane Surface(6) = {6};

Surface Loop(100) = {1,2,3,4,5,6};
Volume(101) = {100};
Physical Volume ("Seds", 1) = {101}; // This line is necessary for
// PETGEM in order to understand each layer.
```

```
// ###################################################################
// #              Refine a Point and or a line                       #
// ###################################################################

// points to increase density
Point (11) = {0.5,1,0.5,lc};

Field[1] = Attractor;
//list of the points where to have a higher density
Field[1].NodesList = {11};
//list of lines where to have a higher density
Field[1].EdgesList = {9};

Field[2] = Threshold;
Field[2].IField = 1;
Field[2].LcMin = lc / 10.;
Field[2].LcMax = lc;
Field[2].DistMin = lc / 4.;
Field[2].DistMax = 2.*lc;

// ###################################################################
// #          Refine a rectangle and or a solid box                  #
// ###################################################################

Field[3] = Box;
Field[3].VIn = lc / 10;
Field[3].VOut = lc;
Field[3].XMin = 0.25;
Field[3].XMax = 0.75;
Field[3].YMin = 0.25;
Field[3].YMax = 0.75;
Field[3].ZMin = 1.;
Field[3].ZMax = 1.;

//by setting Min=Max in 1 dimension, we get a rectangle
//if for all dimensions Min different Max, one gets
//a solid cube

// ###################################################################
// #                     Join all Fields                             #
// ###################################################################


// Use minimum of all the fields as the background field
Field[11] = Min;
Field[11].FieldsList = {2,3};
Background Field = 11;
```

B Algorithm's to create a homogeneous distribution of receivers

The first algorithm is used to create an homogeneous distribution of receivers in a unitary box from -0.4 to 0.4 in all directions. To scale the receivers just change the variable

ScaleFactor.

```
ScaleFactor=100;
xtarget=[0.1:0.1:0.9];
ytarget=[0.1*ones(1,9)];
for irepeat=1:8
    xtarget=[xtarget,0.1:0.1:0.9];
    ytarget=[ytarget,0.1*(irepeat+1)*ones(1,9)];
end
[a,lengthxtarget]=size(xtarget);
xtarget2=xtarget;
ytarget2=ytarget;
ztarget2=0.1*ones(1,lengthxtarget);
for irepeat=1:8
xtarget2=[xtarget2,xtarget];
ytarget2=[ytarget2,ytarget];
ztarget2=[ztarget2,0.1*(irepeat+1)*ones(1,lengthxtarget)];
end
[a,lengthxtarget2]=size(xtarget2);
receivers = zeros(lengthxtarget2, 3);
receivers(:,1)=xtarget2-0.5;
receivers(:,2)=ytarget2-0.5;
receivers(:,3)=ztarget2-0.5;
receivers=ScaleFactor*receivers;
dlmwrite('/home/marc/Dropbox/BSC/gmsh/cube/cube.txt',...
  receivers,'delimiter','\t','precision','%.6e')
```

The second one is equivalent but for a cylinder. "R" changes the radius, "L" the length and "n" the number of layers of receivers in the Z direction

```
R=23/100;
L=96/100;
n=5;
xtargetAux=R*[1,0.5,0.5,0,0,0,-0.5,-0.5,-1,0.25,0.25,-0.25,...
  -0.25,0.15,0.15,-0.15,-0.15,0,0,0.25,-0.25];
ytargetAux=R*[0,0.5,-0.5,1,0,-1,0.5,-0.5,0,0.25,-0.25,0.25,...
  -0.25,0.15,-0.15,0.15,-0.15,0.25,-0.25,0,0];
[nothing,lengthAux]=size(xtargetAux);
zdiv=linspace(0,L,n);
zAux=ones(1,lengthAux);
xtarget=xtargetAux;
ytarget=ytargetAux;
ztarget=zeros(1,lengthAux);
for i=1:n-1
 xtarget=[xtarget,xtargetAux];
 ytarget=[ytarget,ytargetAux];
 ztarget=[ztarget,zAux*zdiv(i+1)];
end
[a,lengthtargets]=size(xtarget);
receivers = zeros(lengthtargets, 3);
receivers(:,1)=xtarget;
receivers(:,2)=ytarget;
receivers(:,3)=ztarget-L/2;
dlmwrite('/home/marc/Dropbox/BSC/gmsh/cylinder/cylinder.txt',...
  receivers,'delimiter','\t','precision','%.6e')
```

C Visualization Matlab script

```
load('/home/marc/petgem-0.30.48/examples/Output/Matlab/Ep.mat');
load('/home/marc/petgem-0.30.48/examples/Output/Matlab/Es.mat');
load('/home/marc/petgem-0.30.48/examples/Output/Matlab/Et.mat');
receivers=load('/home/marc/petgem-0.30.48/PreprocessingOutput/receiversPETGEM.txt');
figure
quiver3(receivers(:,1), receivers(:,2), receivers(:,3),
    real(Ep(:,1)),real(Ep(:,2)),real(Ep(:,3)))
hold on
scatter3(receivers(:,1), receivers(:,2), receivers(:,3),'.')
hold off
```