UNIVERSITÀ DEGLI STUDI DI PADOVA

(IN COLLABORATION WITH GREENWICH UNIVERSITY)

DEPARTMENT OF INDUSTRIAL ENGINEERING
MASTER DEGREE COURSE IN ELECTRICAL ENGINEERING

ELECTROMAGNETIC SEPARATION OF IMPURITIES IN MOLTEN SILICON

SUPERVISOR:
Dr. Michele Forzan

GRADUATE:
Giorgio Zorzi

CO-SUPERVISOR (at Greenwich University):
Dr. Valdis Bojarevics

AY 2013/2014

SOMMARIO

Lo scopo di questa tesi è lo studio del metodo per la separazione delle impurità nel silicio fuso attraverso l'applicazione di un campo elettromagnetico. Questo approccio sfrutta la differenza di conduttività che esiste tra le impurità e lo stesso silicio fuso. L'analisi è fatta usando il metodo degli elementi finiti con implementazione in MATLAB, concentrandosi sui casi in cui la frequenza della corrente di alimentazione dell'induttore è di 1 kHz e 11 kHz. Risultati precedenti, ottenuti usando metodi spettrali, sono serviti per comparare e validare i risultati ottenuti.

È stato inoltre portato a termine uno studio termico sui conduttori componenti l'induttore, sempre utilizzando il metodo degli elementi finiti implementato in MATLAB.

I risultati confermano che l'uso di un campo elettromagnetico è un metodo valido per la separazione delle impurità nel silicio fuso.

ABSTRACT

The aim of this work is the study of the method for the separation of impurities in the molten silicon by the application of an electromagnetic (EM) field. This approach takes advantage of the differences in electrical conductivity of the impurities and the molten silicon. The analysis is performed using finite element method (FEM) with the particular simulations implemented in the MATLAB environment. The focus has been taken to the 1 kHz and 11 kHz power supply cases. Previous results using spectral methods were used to compare and validate the MATLAB FE solutions.

Additionally, a thermal study of the conductor load has been carried out using FEM method in the MATLAB environment.

The results confirm that using the electromagnetic field is a valid approach for the EM separation of impurities from the molten silicon.

RINGRAZIAMENTI

Un ringraziamento va al mio relatore, il Prof. Michele Forzan, che, oltre a farmi scoprire il mondo del riscaldamento ad induzione, mi ha aiutato per la stesura di questa tesi.

Un altro grande ringraziamento va al supervisor presso Greenwich University, il Prof. Valdis Bojarevics, il quale, anche quando sommerso dal lavoro ha sempre trovato un momento da dedicarmi. Questa tesi è anche frutto della sua gentilezza, disponibilità e dei suoi preziosi consigli.

Un ringraziamento particolare va a mia madre Maria Teresa e mio padre Giacomo, che mi hanno sempre sostenuto durante il mio percorso, non solo accademico, ma anche di crescita personale. Devo sicuramente a loro quello che sono oggi.

Ringrazio anche mia sorella Angela e suo marito Marco per il loro aiuto e per aver messo al mondo i miei meravigliosi nipotini Gabriele e Filippo.

Un grande ringraziamento va ad Andrea, un caro amico che nonostante sostenga l'inutilità dello studio, si è sempre fatto in quattro nel momento del bisogno.

Nei miei ringraziamenti, ovviamente, non poteva non mancare Lei, la fonte della mia ispirazione giornaliera, la bussola della mia esistenza, la mia gioia più grande, Lia. Grazie per esserci sempre e per credere in me più di quanto faccia io.

Ringrazio e chiedo scusa a tutti gli altri amici che non ho citato in queste poche righe, ma che rimarranno sempre nel mio cuore.

CONTENTS

LIST OF FIGURES

List of Abbreviations and Acronyms

SYMBOL	DESCRIPTION
CFF	Ceramic Foam Filters
CPV	Concentration Photovoltaic
c-Si	Crystalline Silicon
DC	Direct Current
EPM	Electromagnetic Processing of Materials
FEM	Finite Element Method
GUI	Graphical User Interface
GW	Giga Watt (10E9 W)
HCPV	High Concentration Photovoltaic
LCPV	Low Concentration Photovoltaic
ODE	Ordinary Differential Equations
OPV	Organic Photovoltaic
PDE	Partial Differential Equation
ppb	Parts per billion (1/10E9)
ppm	Parts per million (1/10E6)
ppt	Parts per trillion (1/10E12)
PV	Photovoltaic
TF	Thin Film

LIST OF APPENDICES

1 Introduction

1.1 Energy outlook

The non-sustainable nature of fossil fuels leads naturally to a challenge to find a new source of energy. For this reason renewable energies research is growing fast. The world's cumulative PV (photovoltaic) capacity surpassed the impressive 100-GW installed, achieving over 102 GW in 2012 (See Figure 1-1) [1].

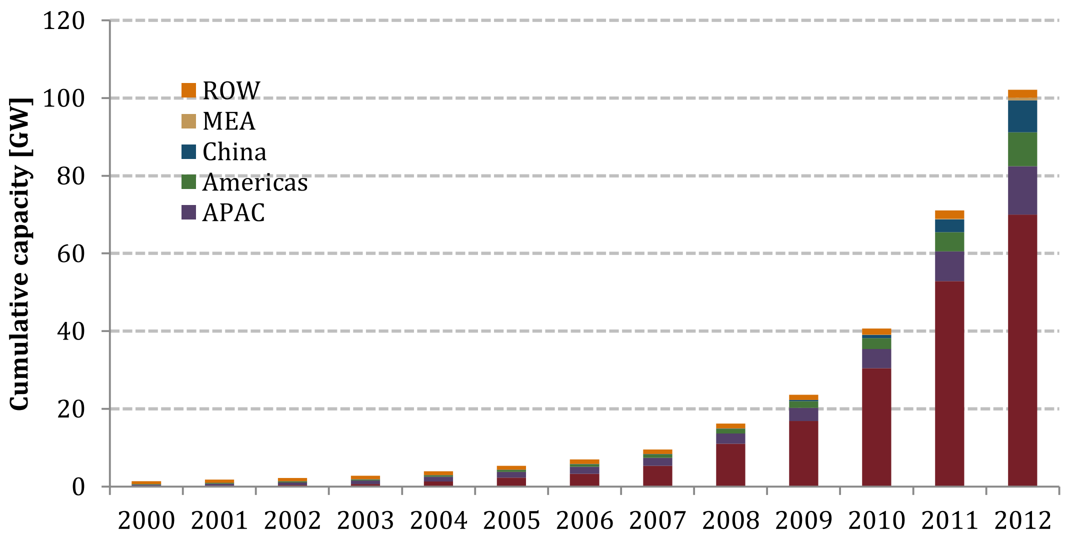

Figure 1-1: Evolution of global cumulative renewable installed capacity 2000-2012 [MW]

It is known that the PV is capable of producing as much annual electrical energy as 16 coal power plants or nuclear reactors of 1 GW each. Every year these PV installations save more than 53 million tons of CO2. PV remains, after hydro and wind power, the third most important renewable energy source in terms of globally installed capacity [2].

In the scenario 2012-2035 from IEA (International Energy Agency), the largest addition of renewable energy will be made by wind (~1250 GW) followed by PV (~750 GW) and hydro (~740 GW).

The PV market is variable but it is strongly lead by the c-Si (crystalline silicon) technology which is expected to maintain its market share at 80% for the next 20 years (See Figure 1-2) [1].

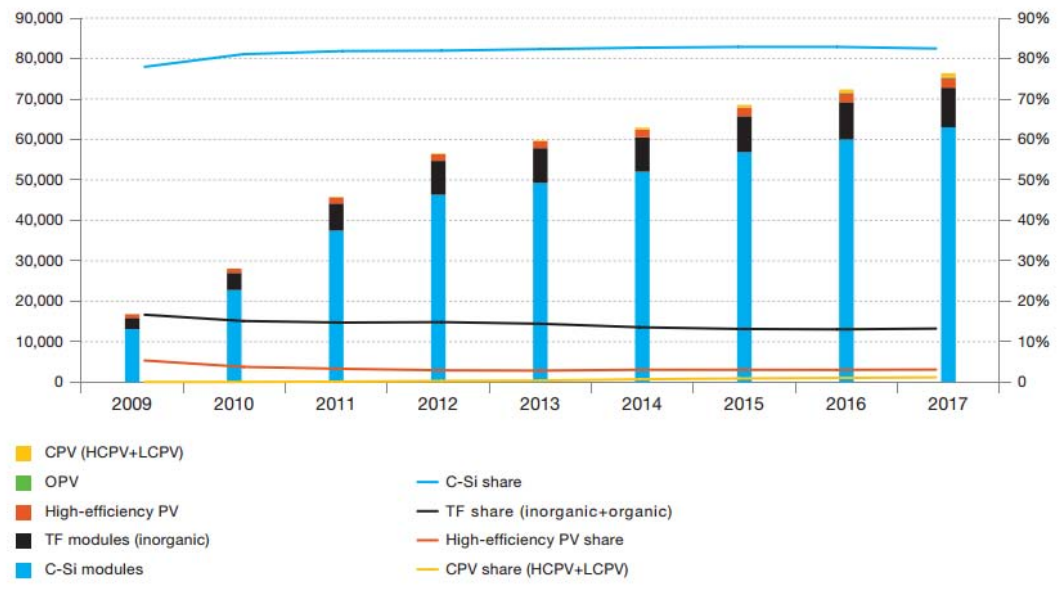

Figure 1-2: PV modules production capacity until 2017 (MW; %)

For this reason, a lot of investment have been done in this market in attempts to find new improved production methods and recycling processes for the PV silicon.

1.2 Production of Silicon

The most common way of producing metallurgic grade silicon is the carbothermic reduction of quartz to obtain silicon with typical purity of 98.5%. The production process involves submerged electric arc furnaces where a crucible is filled with quartz and carbon materials. Silicon is released in the carbothermic reduction of silica according to the overall reaction:

$$SiO_2\ (solid) + 2C(solid) = Si(liquid) + 2CO(gas) \tag{1.1}$$

The purity of metallurgic grade silicon is not sufficient for direct use, so a further refining is necessary. Nowadays there are three reliable methods for the industrial production of the next purity stage material called poly-silicon:

- *Siemens process:* the most popular process developed in the late 1950s is based on the thermal decomposition of trichlorosilane at 1100 °C on a heated silicon rod or filament placed inside a deposition chamber.
- *Carbide Komatsu Process:* is a recent process (1980s) in which the trichlorosilane has been replaced by monosilane SiH_4, but the principle of decomposition on a heated silicon rod inside a closed deposition chamber is maintained.
- *Ethyl Corporation process:* this third process has been developed in 1980s to 1990s and also making use of monosilane SiH_4. The heated silicon rod in the closed reaction chamber has been replaced by a fluidized bed of heated silicon particles. The particles act as seeds on which

SiH4 is continuously decomposed to larger granules of hyper pure silicon. Unlike the two first mentioned this process is a continuous one.

These processes are expensive and produce a hyper-pure material, with impurities in the order of a ppb (parts per billion), ppt (parts per trillion) [3]. Whereas this high level of purity is a standard for the semiconductor industry, for the production of solar cell is not necessary. This is why new methods for the purification of the metallurgic grade silicon have been investigated.

1.3 Electromagnetic processes

Electromagnetic processes are widely used nowadays for electromagnetic treatments of materials. These methods are preferred more than traditional treatment because of their better performance, and often because these are the only possible ways to operate. Some examples of these treatments are:

- *Improvement of surface quality of cast steel*: the improvement of the surface quality of continuously cast steel is possible directly at the continuous casting stage, which will provide a large amount of energy saving due to the elimination of the process of reheating slabs and the process of surface treatment. An alternative process for improving the surface quality has been found by use of an alternate magnetic field imposed from the coils outside of the mold. The reduction of the contact pressure between the mold and the molten metal due to the magnetic pressure is the main crucial factor to obtain a good surface quality in this process.
- *Induction cold crucible*: the cold crucible is composed of a water cooled segmented crucible containing a charge to be melted, surrounded by an induction coil. The charge is levitated in the crucible using the magnetic pressure. This process enables melting of the charge without contamination from the crucible under a controlled inert gas atmosphere. The cold crucible is indispensable for melting metals with high melting points and chemically reactive properties [4].
- *Bimetallic slab*: in order to produce a bimetallic slab in continuous casting process, a DC (direct current) magnetic field, which has the function to dump the fluid motion, has been applied [5].
- *Electrolytic aluminium production*: for the production of aluminum, the electrolysis process is used to free the metal from the CO_2 [6].

These are only few of the possible application of the electromagnetic processes of materials.

2 Methods for the Separation of Impurities

Here, a list for the most common separation methods with a short explanation will be given.

2.1 Sedimentation

The gravity or centrifugal sedimentation processes are widely used for water treatment, but they find application also in the purification of molten metals. The disadvantage is that these methods are limited to inclusion sizes larger than 100 μm.

2.2 Directional solidification

A reliable method for producing the multi-crystalline silicon for the PV industry is the directional solidification. In this method, the molten silicon is solidified in controlled conditions to achieve a planar solidification front with a well defined interface between the solid and the liquid phases. Since the solubility of major impurities is higher in the liquid than in the solid phase, directional solidification works as a purification process where the impurities are retained in the liquid phase.

The impurity content in solid phase C_S during the crystallization is given by Scheil equation [7]:

$$C_S = k_e\, C_0\, (1 - f_s)^{k_e - 1} \tag{2.1}$$

Where:

C_0 is the initial content coefficient of impurities in the silicon

f_s is the solidified fraction

k_e is the effective segregation coefficient

2.3 Filtration of inclusions with ceramic foam filters

Filtration in metallurgy is a process of removing inclusions by forcing molten metal through a porous material.

In CFF (Ceramic Foam Filters), there are two mechanisms of filtration:

- Cake filtration: the bigger particles in the melt are retained on the surface of the filter where they build up, forming an increasing thicker cake. This layer act as a filter for the incoming particles.
- Deep-bed filtration: In this case, the separation is effected through the particle deposition throughout the entire depth of the filter.

For this reasons, cake filtration is also known as surface filtration, whereas bed filtration is also known as depth filtration.

CFF are commonly used in industry for filtering a variety of molten metals, including aluminium, iron, magnesium and copper. There are no industrial uses of CFF for the purification of silicon, but the experiments point out as a possible way to operate [8].

2.4 Electromagnetic separation

In the electromagnetic separation, an EM field is applied in order to induce currents that, interacting with the magnetic field, create forces in the silicon. The conductivity of the molten silicon at room temperature is low, but for the molten silicon is high enough (approximately the same conductivity of the steel) to induce a significant amount of current in the melt. The Lorentz force (See Chap 3.1.2) acts on the melt pushing it accordingly. The result is that the impurities are pushed in the opposite direction (See Figure 2-1). In this figure it can be seen that the current in the melt avoids the not conductive particle. The magnetic field **B** is applied in the direction out of the plane. The resultant Lorentz force is indicated with **f** expressed as:

$$\boldsymbol{f} = \boldsymbol{j} \times \boldsymbol{B} \tag{2.2}$$

The force **f** acting on the particle is sometimes called the Archimedes electromagnetic force.

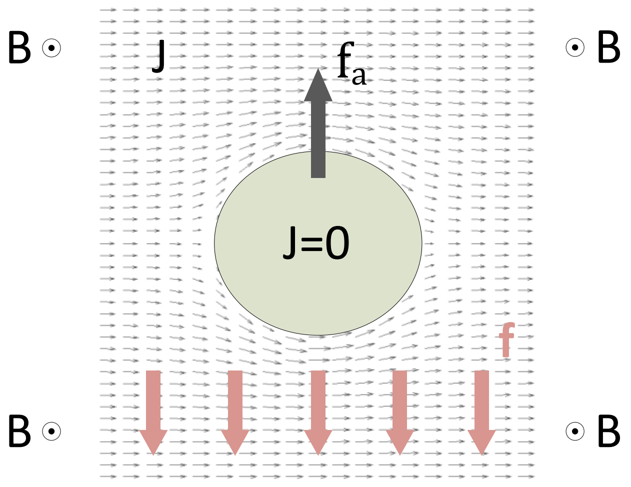

Figure 2-1: Forces acting on a non conductive particle immersed in the molten silicon

In the electromagnetic separation there is more than one possible way to operate. For example it is possible to create the Lorentz force by an induced current or by an injected current. Four methods of electromagnetic separation can be distinguished [9] as listed in Table 2-1.

Injection current separation		Induced current separation	
Pinch-effect separation	Separation in a crossed EM field	Travelling magnetic field separation	Induction coil separation

Table 2-1: Electromagnetic separation methods in liquid metal

2.4.1 Pinch-effect separation

The electrodes are inserted into the melt and the self-induced magnetic field of the injected current provides the source of the Lorentz force. An example is the Figure 2-2. The Lorentz force attempts to compress the conductor material (the pinch effect) producing the pressure distribution in the liquid. As a result, the effective pressure via the Archimedes electromagnetic force acts in the opposite direction and moves inclusions toward the walls of the conductor.

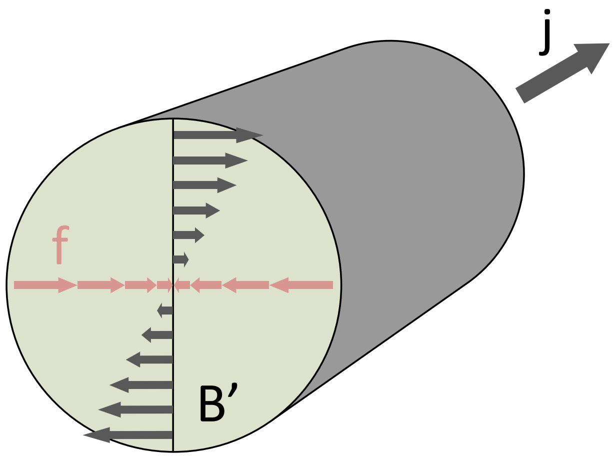

Figure 2-2: Pinch-effect in a liquid conductor

2.4.2 Separation in a crossed EM field

In this method, electrodes are inserted into the melt to create a current density, and an external magnetic field is applied to produce the Lorentz force density as shown in the Figure 2-3. In laboratory tests this method gives very good results due to the available large magnetic field (up to 1T). If this magnetic field is created using permanent magnets, they have to be cooled down below the Curie point, and that can be a technical problem.

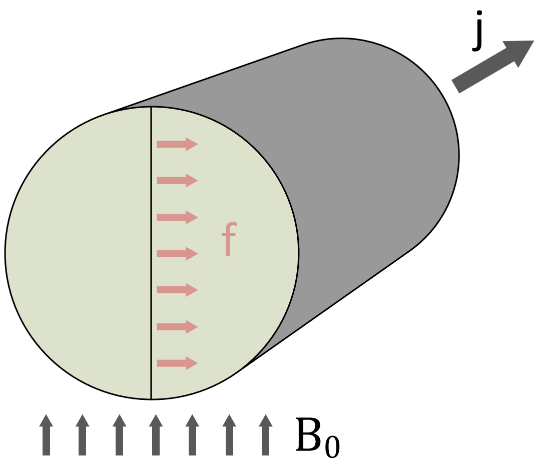

FIGURE 2-3: EXAMPLE OF A SYSTEM USING CROSSED EM FIELD

2.4.3 Travelling magnetic field

In this case, the travelling magnetic field is created in a tube containing the liquid metal with a system similar to a linear motor, producing a non zero steady state Lorentz force component in addition to the time oscillating part.

This method is not as efficient as the previous, but the force can be distributed over significantly larger volumes. To improve the separation efficiency it could be necessary to use more tubes of smaller diameters, but this means a larger contact area with the melt and therefore more contaminations. Despite this, the traveling magnetic field method is the only electromagnetic separation method that is currently in industrial use (e.g., at Pechiney Group in France for the production of aluminium).

2.4.4 Induction coil

The system is formed by an external coil and a crucible with the melt inside. The coil is fed by an AC current that induces a current with an opposite direction in the melt. The Lorentz force is given by the interaction of the current and the total magnetic flux. Due to the skin effect, the force is concentrated on the boundary of the melt (See Figure 2-4).

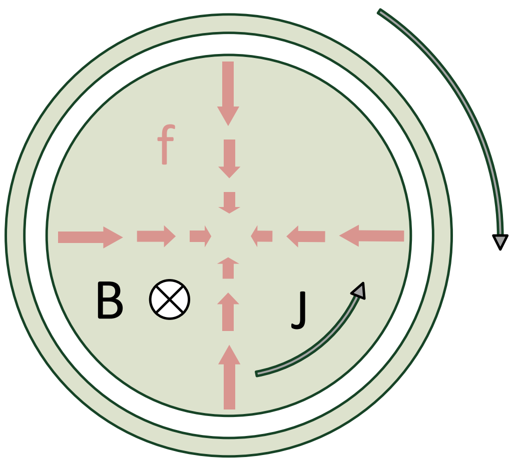

Figure 2-4: Cross section of an induction system

3 Electromagnetic Problem

3.1 Electromagnetic Theory

Here, the theory applied for the construction of the model implemented in MATLAB, is presented.

3.1.1 Constitutive Relations

Constitutive relations describe the medium's properties and effects when two physical quantities are related. In electromagnetics, there are four fundamental constitutive relationships to describe the response of a medium to a variety of electromagnetic input:

$$\boldsymbol{J} = \sigma \boldsymbol{E} \tag{3.1}$$

$$\boldsymbol{D} = \varepsilon \boldsymbol{E} \tag{3.2}$$

$$\boldsymbol{B} = \mu \boldsymbol{H} \tag{3.3}$$

$$\boldsymbol{M} = \chi \boldsymbol{H} \tag{3.4}$$

Where:

J is the current density $\left[\frac{A}{m^2}\right]$

D is the displacement field $\left[\frac{C}{m^2}\right]$

E is the electric field $\left[\frac{V}{m}\right]$

B is the magnetic field $[T] = \left[\frac{Wb}{m^2}\right]$

H is the magnetic field strength $\left[\frac{A}{m}\right]$

M is the magnetic dipole moment $\left[\frac{A}{m}\right]$

And also:

σ is the electric conductivity

ε is the dielectric permittivity

μ is the magnetic permeability

χ is the magnetic susceptibility.

In these equation it is assumed that the medium is linear and isotropic, so the coefficients are not tensors but real numbers.

To be observed that equations 3.3 and 3.4 are not independent, but are related by:

$$\mu = \mu_0(1+\chi) \tag{3.5}$$

3.1.2 Lorentz force

The Lorentz force is the combination of electric and magnetic forces on a point charge due to electromagnetic fields. If a particle of charge q moves with a velocity **v** in the presence of an electric field **E** and a magnetic field **B**, then it will experience a force given by:

$$\boldsymbol{F} = q(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}) \tag{3.6}$$

3.1.3 Ampere's Law

The Ampere's law states that the integral around a closed path of the component of the magnetic field tangent to the direction of the path equals μ_0 times the current intercepted by the area within the path, or in integral notation:

$$\oint_C \boldsymbol{B}\bullet \boldsymbol{dl} = \mu_0 \iint_S \boldsymbol{J}\bullet \boldsymbol{dS} = \mu_0\, \boldsymbol{I} \tag{3.7}$$

Using the Stokes' theorem (See Appendix 1) the equation 3.7 can be written as

$$\nabla \times \boldsymbol{B} = \mu_0\, \boldsymbol{J} \tag{3.8}$$

3.1.4 Faraday's Law

Faraday's law of induction states that the induced magnetomotive force (mmf) e in a coil is proportional to the negative of the rate of change of magnetic flux:

$$e = -\frac{d\phi_B}{dt} \tag{3.9}$$

3.1.5 Electrostatic Potential

The electrostatic potential is defined as follow:

$$V_B - V_A = \int_A^B \boldsymbol{E \bullet dl} \tag{3.10}$$

If **E** is generated by a stationary distribution of charge the field is conservative. In this case, the electrostatic potential does not depend on the path to go from A to B but only from the points A and B.

3.1.6 Gauss' Theorem in Electrostatic

Gauss's theorem states that the surface integral of the electrostatic field **D** over closed surface is equal to the charge enclosed by that surface. That is

$$\oiint_S \boldsymbol{D \bullet dS} = \iiint_V \rho \, dV \tag{3.11}$$

Where ρ is the charge per unit volume.

3.1.7 Magnetostatic Potential

Almost like in the electrostatic case, in magnetostatics it is also possible to define a potential:

$$V_Q - V_P = -\frac{1}{\mu_0} \int_P^Q \boldsymbol{B \bullet dl} \tag{3.12}$$

3.1.8 Gauss' Theorem in Magnetostatic

As done in electrostatic:

$$\oiint_S \boldsymbol{B \bullet dS} = 0 \tag{3.13}$$

This means that there are no sources of magnetic field (there are no monopoles).

3.1.9 Poisson's and Laplace's Equations

The equation 3.19 can be written, considering the constitutive relation 3.2 as

$$\nabla \bullet \boldsymbol{E} = \frac{\rho}{\varepsilon} \tag{3.14}$$

However, **E** can be expressed as:

$$E = -\nabla V \tag{3.15}$$

Therefore,

$$\nabla^2 V = \frac{\rho}{\varepsilon} \tag{3.16}$$

This is the Poisson’s equation. This, in the case where the charge density is zero becomes:

$$\nabla^2 V = 0 \tag{3.17}$$

Which is generally known as Laplace's equation.

3.1.10 Maxwell's Equation

This first Maxwell’s equation describes the electrostatic field and is derived immediately from Gauss's theorem (equation 3.11), in fact applying the divergence theorem (See equation 7.1):

$$\oint\!\!\oint_S \boldsymbol{D} \bullet \boldsymbol{dS} = \iiint_V (\nabla \bullet \boldsymbol{D})\, dV = \iiint_V \rho\, dV \tag{3.18}$$

Therefore, rewriting the first Maxwell equation in differential form:

$$\nabla \bullet \boldsymbol{D} = \rho \tag{3.19}$$

As seen in chapter 3.1.8, and unlike the electrostatic field, magnetic fields have no sources or sinks, so the magnetic lines of force are closed curves. The surface integral of the magnetic field over a closed surface is zero (the field is solenoidal), and therefore, equation (3.13), in differential form becomes:

$$\nabla \bullet \boldsymbol{B} = 0 \tag{3.20}$$

This is namely Maxwell's second equation.

Since ∇•**B**=0 (equation 3.20), it is reasonable to assume that exists a vector **A** such as:

$$\boldsymbol{B} = \nabla \times \boldsymbol{A} \tag{3.21}$$

The **A** vector is called vector potential.

A is not unique because taking ∇X, with X to be a scalar, it satisfies the equation 3.21, since:

$$\nabla \times (\nabla X) = 0 \tag{3.22}$$

The third Maxwell's equation is derived from Ampère's theorem (See 3.1.2) that in the general case it must be read

$$\oint_{\partial S} \boldsymbol{H} \bullet \boldsymbol{dl} = \iint_S \left(\frac{\partial \boldsymbol{D}}{\partial t} + \boldsymbol{J}\right) \bullet \boldsymbol{dS} \tag{3.23}$$

Applying the Stokes' theorem (equation 7.1) at the left-hand side of 3.23:

$$\iint_S (\nabla \times \boldsymbol{H}) \bullet \boldsymbol{dS} = \iint_S \left(\frac{\partial \boldsymbol{D}}{\partial t} + \boldsymbol{J}\right) \bullet \boldsymbol{dS} \tag{3.24}$$

Therefore, we obtain the third Maxwell's equation

$$\nabla \times \boldsymbol{H} = \frac{\partial \boldsymbol{D}}{\partial t} + \boldsymbol{J} \tag{3.25}$$

The fourth Maxwell's equation is derived from the laws of electromagnetic induction. Starting from the Faraday's law (equation 3.9) and taking curl of both sides of the equation we have:

$$\nabla \times \boldsymbol{E} = -\nabla \times \left(\frac{\partial \boldsymbol{\phi}}{\partial t}\right) = -\frac{\partial}{\partial t}(\nabla \times \boldsymbol{\phi}) = -\frac{\partial \boldsymbol{B}}{\partial t} \tag{3.26}$$

Rewriting, we have the fourth Maxwell's equation:

$$\nabla \times \boldsymbol{E} + \frac{\partial \boldsymbol{B}}{\partial t} = 0 \tag{3.27}$$

3.1.11 Skin Effect

Skin effect is the tendency of an alternating electric current to become distributed such that the current density is larger near the surface of the conductor, and decreases with greater depths. The skin effect causes the effective resistance of the conductor to increase at higher frequencies where the skin depth is smaller, thus reducing the available section for the current to flow. The skin effect is due to opposing eddy currents induced by the changing magnetic field resulting from the alternating current. At 50 Hz in copper, the skin depth is about 10 mm. At high frequencies, the skin depth becomes much smaller. Because the interior of a large conductor carries so little of the current, tubular conductors such as pipe can be used to save weight and cost.

The skin depth can be computed with the equation 3.28.

$$\delta = \sqrt{\frac{2\rho}{\mu\omega}} \tag{3.28}$$

In Figure 3-1, is represented the trend of the skin depth for copper and silicon (at melting point), varying the frequency.

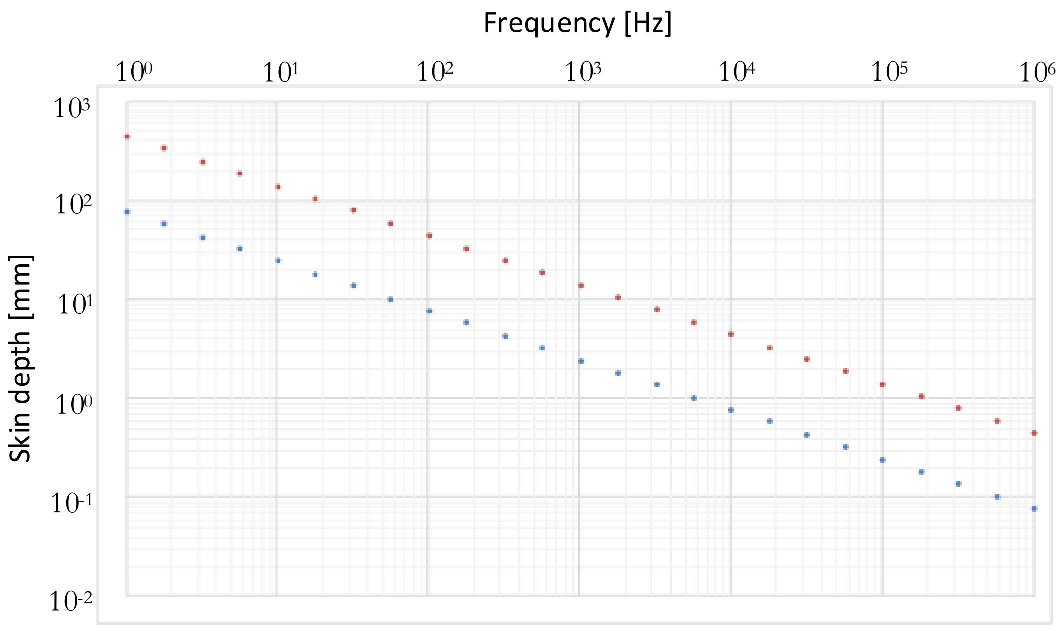

FIGURE 3-1: SKIN DEPTH IN THE SILICON AND COPPER

3.2 MATLAB

The MATLAB environment is very various and can be used for a large variety of applications, but here some useful information will be given regarding the study under consideration.

3.2.1 Introduction

MATLAB (abbreviation of matrix laboratory) is a numerical computing environment and fourth-generation programming language, developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran [10].

With MATLAB is possible to solve differential equation, in particular the following:

- First order ODEs (Ordinary Differential Equation) with the ode series solvers (the most used are the ode15s, the ode45 that use Runge-Kutta and the ode113 that uses the Adams method) [11]
- Higher order ODEs (rewriting each equation as an equivalent system of first order ODE) [11]
- Systems of parabolic and elliptic PDEs (Partial Differential Equation) in 1D with the `pdepe` function [12]
- Systems of elliptic, parabolic, hyperbolic, eigenvalue and non-linear equations with the PDE toolbox [13].

3.2.2 PDE toolbox

The solutions of simple PDEs on complicated geometries can rarely be expressed in terms of elementary functions, but it is possible to have an approximate solution for these problems discretizing the domain and finding an approximate solution for every element. For example, the Partial Differential Equation Toolbox algorithm is a PDE solver that uses FEM for problems defined on bounded domains in the plane.

The first step to solve a problem with the PDE toolbox is to describe a complicated geometry and generate a mesh on it. Then, it is necessary to discretize the PDE on the mesh and build an equation for the discrete approximation of the solution.

The PDE toolbox provides an easy-to-use graphical tool to describe complicated domains and generate triangular meshes. It also discretizes PDEs, finds discrete solutions and plots results [14]

The types of equation that the PDE tool can manage are [14]:

1. *Elliptic equation*:

$$-\nabla \cdot (c\nabla u) + au = f \tag{3.29}$$

2. *Parabolic equation*:

$$d\frac{\partial u}{\partial t} - \nabla \cdot (c\nabla u) + au = f \tag{3.30}$$

3. *Hyperbolic equation*:

$$d\frac{\partial^2 u}{\partial t^2} - \nabla \cdot (c\nabla u) + au = f \tag{3.31}$$

4. *Eigenvalue equation*:

$$-\nabla \cdot (c\nabla u) + au = \lambda du \tag{3.32}$$

5. *Nonlinear equation*:

$$-\nabla \cdot (c(u)\nabla u) + a(u)u = f(u) \tag{3.33}$$

6. *System of N equations* (for example the system of elliptic equation with N=2 is):

$$\begin{aligned} -\nabla \cdot (c_{11}\nabla u_1) - \nabla \cdot (c_{12}\nabla u_2) + a_{11}u_1 + a_{12}u_2 = f_1 \\ -\nabla \cdot (c_{21}\nabla u_1) - \nabla \cdot (c_{22}\nabla u_2) + a_{21}u_1 + a_{22}u_2 = f_2 \end{aligned} \tag{3.34}$$

With the PDE toolbox it is possible to solve only a system with N=2, but there are no limits using command line functions.

The boundary conditions for the equations from one to five are [14]:

A. *Dirichlet*:

$$hu = r \tag{3.35}$$

B. *Generalized Neumann*:

$$\vec{n} \cdot (c\nabla u) + qu = g \tag{3.36}$$

Where $\vec{n}$ is the outward unit vector.

Referring to the system (3.34), the boundary conditions are expressed as follow [14]:

A. *Dirichlet*:

$$\begin{aligned} h_{11}u_1 + h_{12}u_2 = r_1 \\ h_{21}u_1 + h_{22}u_2 = r_2 \end{aligned} \tag{3.37}$$

B. *Generalized Neumann*:

$$\begin{aligned} \vec{n} \cdot (c_{11}\nabla u_1) + \vec{n} \cdot (c_{12}\nabla u_2) + q_{11}u_1 + q_{12}u_2 = g_1 \\ \vec{n} \cdot (c_{21}\nabla u_1) + \vec{n} \cdot (c_{22}\nabla u_2) + q_{21}u_1 + q_{22}u_2 = g_2 \end{aligned} \tag{3.38}$$

C. *Mixed*:

$$\begin{aligned} h_{11}u_1 + h_{12}u_2 = r_1 \\ \vec{n} \cdot (c_{11}\nabla u_1) + \vec{n} \cdot (c_{12}\nabla u_2) + q_{11}u_1 + q_{12}u_2 = g_1 + h_{11}\mu \end{aligned} \tag{3.39}$$

$$\vec{n}\cdot(c_{21}\nabla u_1)+\vec{n}\cdot(c_{22}\nabla u_2)+q_{21}u_1+q_{22}u_2=g_2+h_{12}\mu$$

Where μ is computed such the Dirichlet boundary conditions is satisfied.

3.3 Electromagnetic Model

As will be seen in chapter 3.5.1, the system is axisymmetric, so it is possible to implement a 2D axisymmetric model.

We start from the third Maxwell's equation (3.25). The time derivative of the displacement field can be neglected when (equation 7.10, appendix 1):

$$\left|\omega\frac{r_{max}}{c}\right|=\left|2\pi\cdot 11\cdot 10^3\frac{1\cdot 10^{-3}}{3\cdot 10^5}\right|\approx 1\cdot 10^{-4}\ll 1 \qquad (3.40)$$

Therefore, it is possible to neglect this term without affecting the results, and the 3.25 becomes:

$$\nabla\times\boldsymbol{H}=\boldsymbol{J} \qquad (3.41)$$

Now, taking the fourth Maxwell's equation (equation 3.27) and using the definition of vector potential (equation 3.21), we obtain:

$$\nabla\times\left(\boldsymbol{E}+\frac{\partial\boldsymbol{A}}{\partial t}\right)=0 \qquad (3.42)$$

So it is possible to define a scalar potential such that:

$$\boldsymbol{E}+\frac{\partial\boldsymbol{A}}{\partial t}=-\nabla V \qquad (3.43)$$

Since we are using the axisymmetric model, the vector potential **A** has only the component along the φ axis

$$\boldsymbol{A}=\boldsymbol{A_\varphi}\cdot\boldsymbol{e_\varphi} \qquad (3.44)$$

Now, continuing from 3.41 and using the constitutive relation 3.1 and the equation 3.43:

$$\nabla \times \boldsymbol{H} = \boldsymbol{J} = \sigma \boldsymbol{E} = \sigma \left(-\frac{\partial \boldsymbol{A}}{\partial t} - \nabla V \right) \tag{3.45}$$

In the equation above, the first term in parenthesis, represents the induced current, and the second term, the given current.

The **J** applied is sinusoidal, and the material, for approximation, can be considered linear, so the vector potential is also sinusoidal. In these conditions, the time derivative of **A** take the form:

$$\frac{\partial \boldsymbol{A}}{\partial t} = \frac{\partial}{\partial t} \left(\boldsymbol{A_\varphi} \cdot e^{j\omega t} \right) \cdot \boldsymbol{e_\varphi} = \left(j\omega \boldsymbol{A_\varphi} \cdot e^{j\omega t} \right) \cdot \boldsymbol{e_\varphi} \tag{3.46}$$

Therefore, it is possible to rewrite the last term of 3.45 as follow

$$\nabla \times \boldsymbol{H} = \sigma \left(-\frac{\partial \boldsymbol{A}}{\partial t} - \nabla V \right) = \left(-j\sigma\omega \boldsymbol{A_\varphi} + \boldsymbol{J_\varphi} \right) \cdot e^{j\omega t} \tag{3.47}$$

The equation 3.41 can be rewritten also as

$$\nabla \times \boldsymbol{H} = \nabla \times \left(\frac{1}{\mu} \boldsymbol{B} \right) = \nabla \times \left(\frac{1}{\mu} \nabla \times \boldsymbol{A} \right) \tag{3.48}$$

The curl of **A** in cylindrical coordinates takes the form:

$$\nabla \times \boldsymbol{A} = \frac{1}{r} \begin{vmatrix} \boldsymbol{e_r} & r\boldsymbol{e_\varphi} & \boldsymbol{e_z} \\ \frac{\partial}{\partial r} & \frac{\partial}{\partial \varphi} & \frac{\partial}{\partial z} \\ 0 & r\boldsymbol{A_\varphi} & 0 \end{vmatrix} = \boldsymbol{e_r} \left(-\frac{\partial \boldsymbol{A_\varphi}}{\partial z} \right) + \boldsymbol{e_z} \frac{1}{r} \left(\frac{\partial}{\partial r} \left(r\boldsymbol{A_\varphi} \right) \right) \tag{3.49}$$

Now that $\nabla \times \boldsymbol{A}$ is in explicit form, we can write out all the equation 3.48 as following:

$$
\begin{aligned}
\nabla \times \boldsymbol{H} = \nabla \times \left(\frac{1}{\mu}\nabla \times \boldsymbol{A}\right) &= \frac{1}{r}\begin{vmatrix} \boldsymbol{e_r} & r\boldsymbol{e_\varphi} & \boldsymbol{e_z} \\ \dfrac{\partial}{\partial r} & \dfrac{\partial}{\partial \varphi} & \dfrac{\partial}{\partial z} \\ -\dfrac{1}{\mu}\dfrac{\partial \boldsymbol{A_\varphi}}{\partial z} & 0 & \dfrac{1}{r\mu}\dfrac{\partial}{\partial r}(r\boldsymbol{A_\varphi}) \end{vmatrix} \\
&= \frac{1}{r}\boldsymbol{e_r}\left[\frac{\partial}{\partial \varphi}\left(\frac{1}{r\mu}\frac{\partial}{\partial r}(r\boldsymbol{A_\varphi})\right)\right] \\
&- \boldsymbol{e_\varphi}\left[\frac{\partial}{\partial r}\left(\frac{1}{r\mu}\frac{\partial}{\partial r}(r\boldsymbol{A_\varphi})\right) + \frac{\partial}{\partial z}\left(\frac{1}{\mu}\frac{\partial \boldsymbol{A_\varphi}}{\partial z}\right)\right] \\
&+ \frac{1}{r}\boldsymbol{e_z}\left[\frac{\partial}{\partial \varphi}\left(\frac{1}{\mu}\frac{\partial \boldsymbol{A_\varphi}}{\partial z}\right)\right]
\end{aligned}
\tag{3.50}
$$

Due to the 2D axisymmetric case, the current has only $\mathbf{A_\varphi}$**,** component along the φ axis, so the other term vanishes.

$$
\nabla \times \boldsymbol{H} = \boldsymbol{e_\varphi}\left[-\frac{\partial}{\partial r}\left(\frac{1}{r\mu}\frac{\partial}{\partial r}(r\boldsymbol{A_\varphi})\right) - \frac{\partial}{\partial z}\left(\frac{1}{\mu}\frac{\partial \boldsymbol{A_\varphi}}{\partial z}\right)\right] \tag{3.51}
$$

Introducing then the Psi function

$$
\boldsymbol{\psi} = (\psi_r + j\psi_i)(cos\omega t + jsin\omega t) = \boldsymbol{A_\varphi} r \tag{3.52}
$$

With equation 3.51, the Psi function formulation takes the form:

$$
\nabla \times \boldsymbol{H} = \boldsymbol{e_\varphi}\left[-\frac{\partial}{\partial r}\left(\frac{1}{r\mu}\frac{\partial \boldsymbol{\psi}}{\partial r}\right) - \frac{\partial}{\partial z}\left(\frac{1}{r\mu}\frac{\partial \boldsymbol{\psi}}{\partial z}\right)\right] \tag{3.53}
$$

Now we can combine 3.47 rewritten using the Psi function with 3.53 to obtain the equation for the system.

$$
-\frac{\partial}{\partial r}\left(\frac{1}{r\mu}\frac{\partial \boldsymbol{\psi}}{\partial r}\right) - \frac{\partial}{\partial z}\left(\frac{1}{r\mu}\frac{\partial \boldsymbol{\psi}}{\partial z}\right) = -j\frac{\sigma\omega}{r}\boldsymbol{\psi} + \boldsymbol{J}(s) \tag{3.54}
$$

It is now easy to see the analogy with the elliptic equation of the PDE toolbox (equation (3.29)).

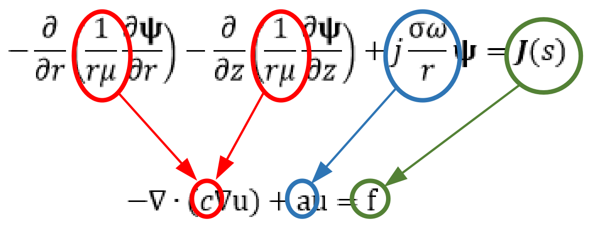
$-\frac{\partial}{\partial r}\left(\frac{1}{r\mu}\frac{\partial \boldsymbol{\psi}}{\partial r}\right)-\frac{\partial}{\partial z}\left(\frac{1}{r\mu}\frac{\partial \boldsymbol{\psi}}{\partial z}\right)+j\frac{\sigma\omega}{r}\boldsymbol{\psi}=\boldsymbol{J}(s)$
$-\nabla\cdot(c\nabla u)+au=f$

FIGURE 3-2: COMPARISON FROM THE ELECTROMAGNETIC MODEL AND THE ELLIPTIC EQUATION IN THE PDE TOOL

Where u is $\boldsymbol{\psi}$.

The boundary conditions to implement are easily derived observing the equation (3.52). On the z axis, we have r=0, so the Psi function becomes zero, while the other edges are far away enough to consider the vector potential equal to zero, so here again the Psi function is equal to zero.

Once computed the solution, we can compute the other vectors of interest:

- Induction field from the equation 3.21

$$\boldsymbol{B}=\nabla\times\boldsymbol{A}=\nabla\times\left(\frac{\boldsymbol{\psi}}{\boldsymbol{r}}\right)=\frac{1}{r}\begin{vmatrix}\boldsymbol{e_r} & r\boldsymbol{e_\varphi} & \boldsymbol{e_z}\\ \frac{\partial}{\partial r} & \frac{\partial}{\partial \varphi} & \frac{\partial}{\partial z}\\ 0 & r\frac{1}{r}\boldsymbol{\psi} & 0\end{vmatrix}=\left(-\frac{1}{r}\frac{\partial \boldsymbol{\psi}}{\partial z}\right)\boldsymbol{e_r}+\left(\frac{1}{r}\frac{\partial \boldsymbol{\psi}}{\partial r}\right)\boldsymbol{e_z} \tag{3.55}$$

- Density of current induced from the equation 3.47

$$\boldsymbol{J_{ind}}=-j\sigma\omega\boldsymbol{A_\varphi}=-j\frac{\sigma\omega}{r}\boldsymbol{\psi} \tag{3.56}$$

- Lorentz force in the molten silicon. Rewriting the vectors as a function of sine and cosine, and taking the real part:

$$\begin{aligned}\Re(\boldsymbol{B})&=\frac{1}{r}\left[\left(\frac{\partial\psi_i}{\partial z}\right)sin(\omega t)-\left(\frac{\partial\psi_r}{\partial z}\right)cos(\omega t)\right]\boldsymbol{e_r}\\&+\frac{1}{r}\left[\left(\frac{\partial\psi_r}{\partial r}\right)cos(\omega t)-\left(\frac{\partial\psi_i}{\partial r}\right)sin(\omega t)\right]\boldsymbol{e_z}\end{aligned} \tag{3.57}$$

$$\Re(\boldsymbol{J})=\frac{\sigma\omega}{r}\big(\psi_i\cdot cos(\omega t)+\psi_r\cdot sin(\omega t)\big)\boldsymbol{e_\varphi} \tag{3.58}$$

Therefore, the force takes the form

$$\begin{aligned}\boldsymbol{F} = \Re(\boldsymbol{J}) \times \Re(\boldsymbol{B}) \\ &= \frac{\sigma\omega}{r^2}\left[\psi_i \cdot cos^2(\omega t)\frac{\partial\psi_r}{\partial r} - \psi_r \cdot sin^2(\omega t)\frac{\partial\psi_i}{\partial r}\right. \\ &\left.+ sin(\omega t)cos(\omega t)\left(\psi_r\frac{\partial\psi_r}{\partial r} - \psi_i\frac{\partial\psi_i}{\partial z}\right)\right]\boldsymbol{e_r} \\ &+ \frac{\sigma\omega}{r^2}\left[\psi_i \cdot cos^2(\omega t)\frac{\partial\psi_r}{\partial z} - \psi_r \cdot sin^2(\omega t)\frac{\partial\psi_i}{\partial z}\right. \\ &\left.+ sin(\omega t)cos(\omega t)\left(-\psi_r\frac{\partial\psi_r}{\partial r} + \psi_i\frac{\partial\psi_i}{\partial z}\right)\right]\boldsymbol{e_z}\end{aligned} \tag{3.59}$$

The average value of the force in a period can be calculated using the trigonometric identities (See Appendix 2), so it equals to:

$$\boldsymbol{F_{avg}} = \frac{\sigma\omega}{2r^2}\left[\psi_i\frac{\partial\psi_r}{\partial r} - \psi_r\frac{\partial\psi_i}{\partial r}\right]\boldsymbol{e_r} + \frac{\sigma\omega}{2r^2}\left[\psi_i\frac{\partial\psi_r}{\partial z} - \psi_r\frac{\partial\psi_i}{\partial z}\right]\boldsymbol{e_z} \tag{3.60}$$

- Specific Joule power can be calculated from the current density

$$\begin{aligned}p = \frac{\Re(\boldsymbol{J})^2}{\sigma} = \frac{\sigma\omega^2}{r^2}\Big(&{\psi_i}^2 \cdot cos^2(\omega t) + {\psi_r}^2 \cdot sin^2(\omega t) \\ &+ 2\psi_r\psi_i sin(\omega t)cos(\omega t)\Big)\end{aligned} \tag{3.61}$$

and the average value is the following

$$p_{avg} = \frac{\sigma\omega^2}{2r^2}\left({\psi_r}^2 + {\psi_i}^2\right) \tag{3.62}$$

3.4 MATLAB Simple Case

Now that we have written out the electromagnetic model, we just have to build it in MATLAB, but before starting with the crucible system, we want to make sure that the model is correct. To verify that, the simple case of a sphere is taken into consideration. The aim is to compute the induced current in the sphere with a numerical solution and compare it with the exact analytical solution.

3.4.1 Geometry

The system is composed of a sphere surrounded by a single coil (Figure 3-3). The analytical solution that we will see in the next chapter (3.4.2) assumes that the conductor is a filament. For this reason, the conductor in the simulation is very small.

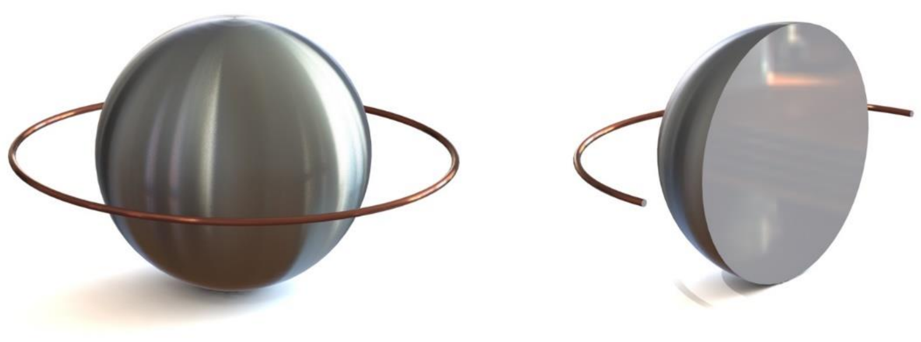

Figure 3-3: Sphere and coil

The sphere is made of silicon and the coil of aluminium. The dimensions are shown in Figure 3-4.

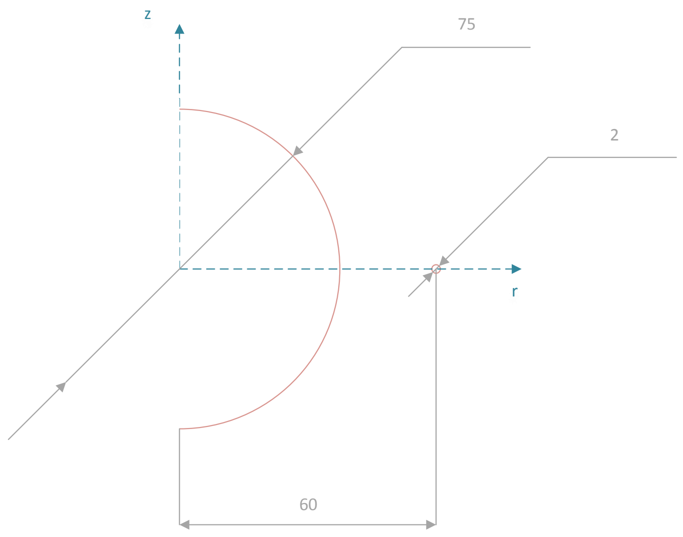

Figure 3-4: Dimensions of the sphere and coil system in a half section of the system

3.4.2 Analytic test case

The analytical solution is given in spherical coordinates and has the following form [15]:

$$A_{\varphi}(R,\vartheta) = e^{jwx}\frac{\mu_o I_s sin(\vartheta_s)}{2\sqrt{jpRR_0}}\sum_{n=1}^{\infty} C_n I_{n+½}\left(\sqrt{jp}R\right)P_n^1(cos\vartheta) \tag{3.63}$$

Where:

$$C_n = \frac{2n+1}{n(n+1)}\left(\frac{R_0}{R_s}\right)^n \frac{P_n^1(cos\vartheta_s)}{I_{n-½}\left(\sqrt{jp}R_0\right)} \tag{3.64}$$

$$p = \sigma\mu_o\omega \tag{3.65}$$

And:

- R_0 is the radius of the sphere
- R_s and ϑ_s are the filament position
- R and ϑ are the position where to compute the current
- I_s and ω are current amplitude and the angular frequency
- n is the harmonic number
- $I_{n+½}$ and $I_{n-½}$ are the modified Bessel functions of complex argument
- P_n^1 the Legendre polynomial.

3.4.3 Numerical

All the simulations, as said before, are created with MATLAB, but not with the GUI (Graphical User Interface) of the PDE toolbox, but writing all at command line. In this way is necessary to decompose the geometry by arc and lines. The decomposition of geometry is shown in Figure 3-5.

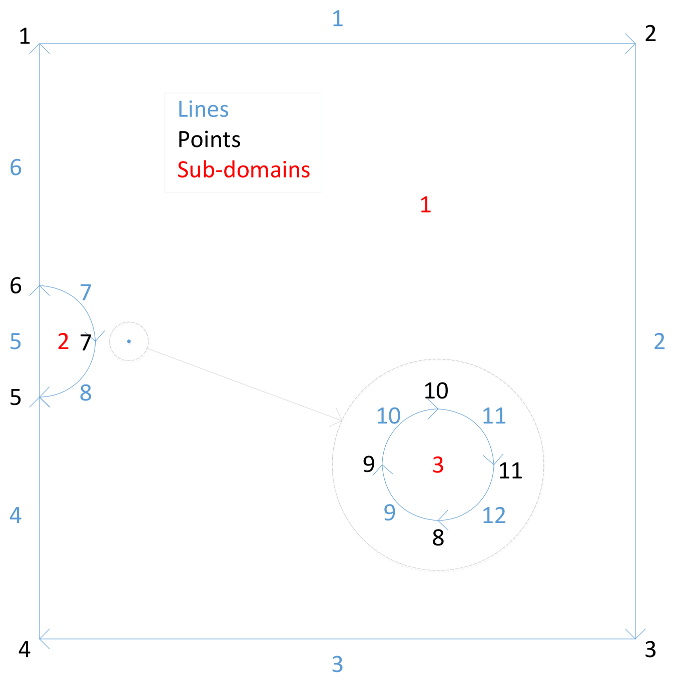

FIGURE 3-5: DECOMPOSED GEOMETRY FOR THE SPHERE CASE

In the figure we can also see the numbering for lines in blue, for points in black and for sub-domains in red.

Once the geometry has been created, is necessary to discretize the domain. For this purpose is used an adaptive mesh. This MATLAB function has the advantage to densify the mesh when necessary, so taking the same number of triangles, it is possible to achieve a better solution. In Figure 3-6 is shown the mesh for this problem, in particular the changes in size of the triangles for a current frequency of 6kHz.

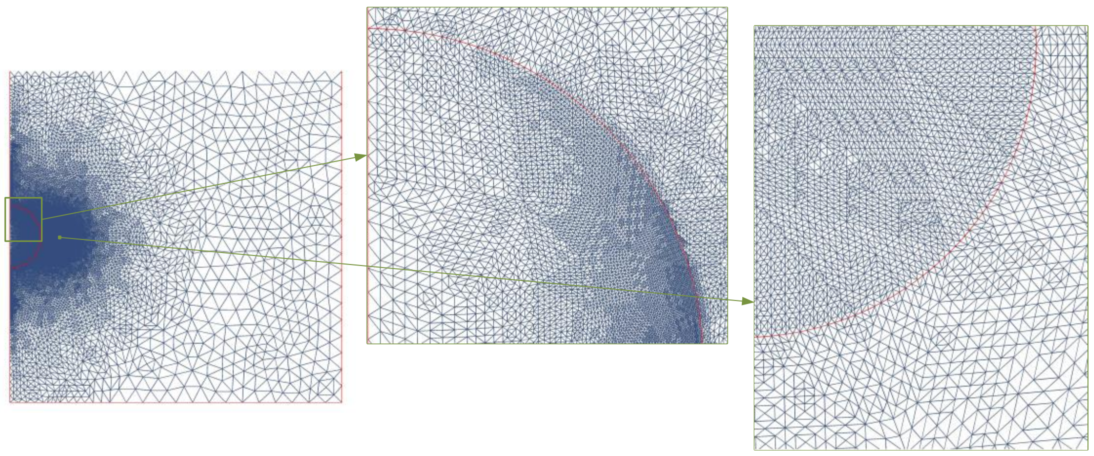

FIGURE 3-6: DETAILS OF THE MESH IN THE SPHERE AND IN THE CONDUCTOR

This mesh is made of 130k triangles of first order.

3.4.4 Solution

The simulation is made with a current of 1000A and a frequency of 6000Hz. The result of the simulation in MATLAB is shown in Figure 3-7, where it is possible to see the current density in the sphere.

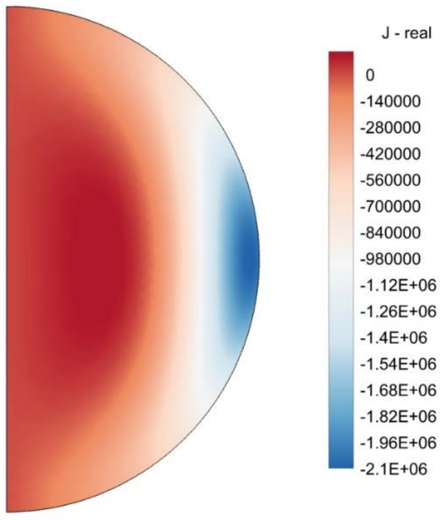

Figure 3-7: Current density distribution in the sphere

To compare the results, the current density is calculated on three surfaces: the equatorial plane, a cone with an angle of 30 degrees and another cone with an angle of 60 degrees (Figure 3-8).

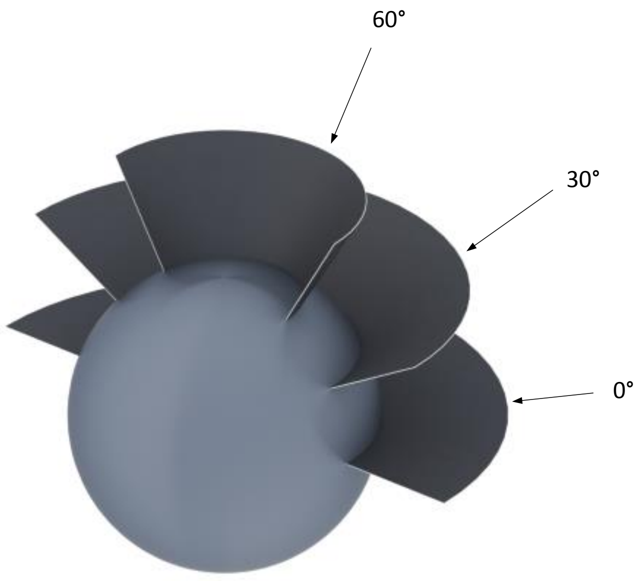

Figure 3-8: Surfaces chosen for the calculation of the current density for the comparison

The Figure 3-9 shows the comparisons from the analytical and the numerical solution.

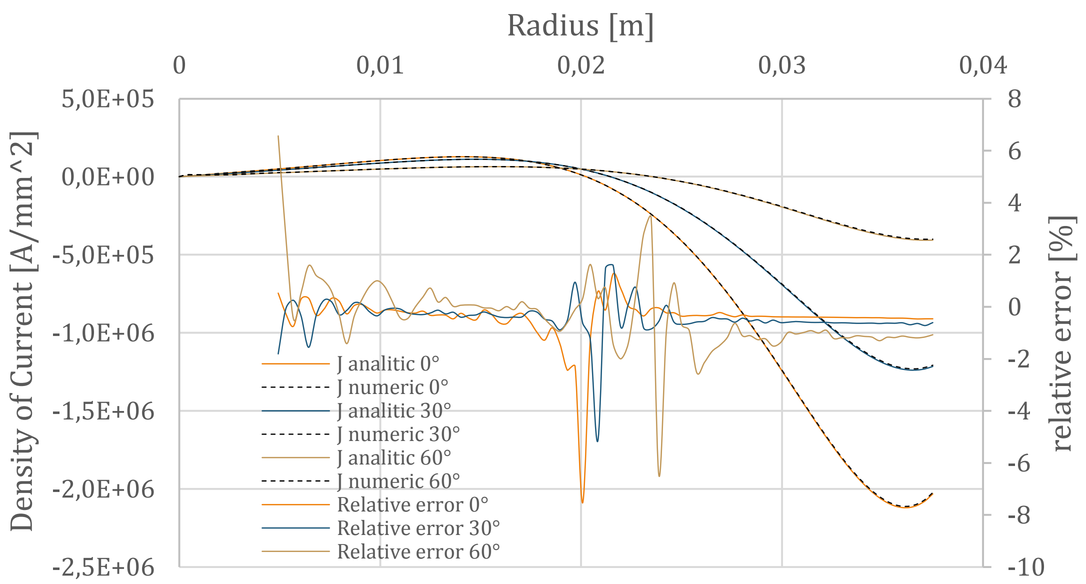

FIGURE 3-9: DENSITY OF CURRENT IN THE ANALYTICAL AND NUMERICAL SOLUTION IN THE SPHERE (LEFT AXIS) AND RELATIVE ERRORS (RIGHT AXIS)

It can be seen that the results are accurate, with the exception of the zone in the proximity of the centre. This is probably due to the singularity in $r = 0$. In general this zone is not so important in our case because the magnitude of current and magnetic field are low compared with the current in the skin depth, so even more for the forces. We can conclude that this model is accurate.

3.5 Electromagnetic crucible set up in MATLAB

Following what has been done for the sphere in chapter 3.4.3, we can do the same for the crucible system.

3.5.1 Geometry

The system studied in this work is very simple (See Figure 3-10).

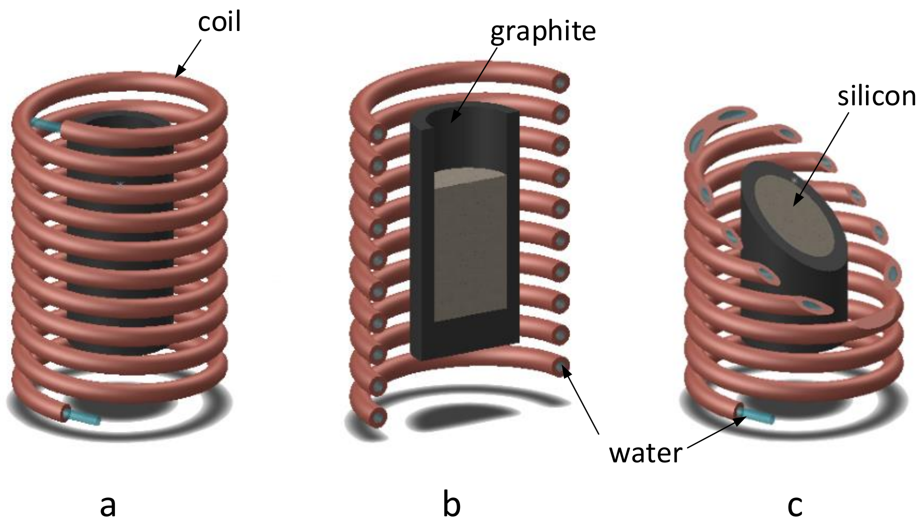

FIGURE 3-10: A) VIEW OF THE SYSTEM. B) HALF SECTION OF THE SYSTEM. C) TRANSVERSAL SECTION OF THE SYSTEM

Externally there is the coil, composed by 10 turns made of copper. The effective value of the current in the coil is fixed at 385 A. With this current, two frequencies have been applied: 1 kHz and 11 kHz. Concentric to the coil there is the crucible of graphite. The purpose of a graphite crucible is to provide a non-reactive vessel that will survive the high temperatures needed for metal melting and processing. This offers a stable container that does not react with the metals at high temperatures. Inside the crucible, there is the silicon. In addition, in Figure 3-10 we can see that the coil is not made of solid copper, but it has a hole to permit the water-cooling. This does not affect the electric conduction because at the frequency we are considering, there is no current in the center of the conductor due to the skin effect.

The silicon core is a small cylinder, with a diameter of 4 cm and a height of 6.5 cm (See Figure 3-11). From this is possible to calculate the weight of the molten silicon as follow.

$$W = V\rho = \pi r^2\, h\rho = \pi(2^{-3}\,)^2\, 6.5^{-3} \cdot 2580 = 0.210[kg] \qquad (3.66)$$

As said before, the system is small, in fact, only 200 grams of silicon is melted in the crucible, but once this configuration is studied, it will be easy to use the same model to a bigger system.

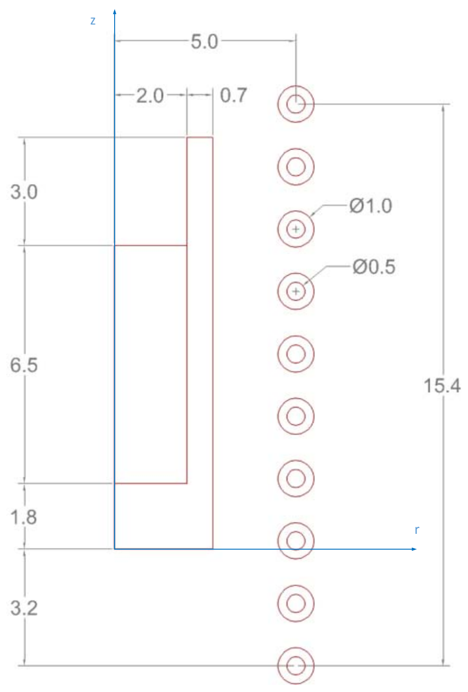

FIGURE 3-11: HALF SECTION OF THE SYSTEM

As it can be seen from Figure 3-11, the system has a symmetry on the z axis that will be exploited for the construction of the model, so an axisymmetric model can be implemented, with the advantage that only a 2D model has to be studied, and consequently a reduction of the computational time.

The characteristics of the materials used in the simulation are grouped in the next table.

	Conductivity $\sigma\ [\Omega \cdot m]$	*Relative magnetic permeability* $\mu_r\ [-]$	*Density* $\rho\ \left[\frac{kg}{m^3}\right]$
Silicon	$1.23 \cdot 10^6$	1	2560
Graphite	$8.65 \cdot 10^4$	1	2160
Copper	$4.1 \cdot 10^7$	1	8920
Air	0	1	0
Water	0	1	1000

TABLE 3-1: CHARACTERISTICS OF THE MATERIALS USED IN THE SIMULATIONS

The program for the simulation used for the 11 kHz case can be seen in appendix 3.

3.5.2 Mesh

We start decomposing the geometry of half system. The result can be seen in Figure 3-12, where it is shown the domain of the simulations.

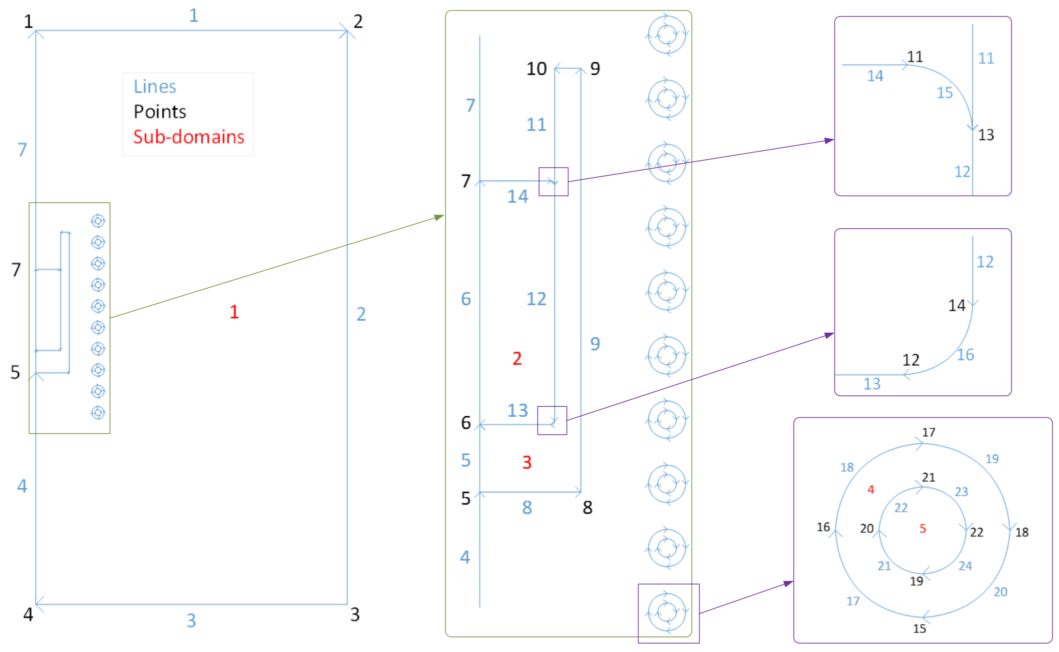

FIGURE 3-12: DECOMPOSED GEOMETRY FOR THE SYSTEM SILICON-CRUCIBLE-TURNS

In the same figure, in the green box there is the particular of the crucible and the turns. All the dimensions are shown in chapter 3.5.1. In Figure 3-12, in the lower purple box, is represented the numeration for the lower conductor. Every conductor has two subdomains, the external for the copper and the internal for the water. The numeration for points, lines and sub-domains of the other conductors, follows the same rule as for the one presented in Figure 3-12. In the other two upper purple boxes, there is the particular of the lower and the upper edge of the silicon cylinder. These edges are rounded to avoid false results in these areas.

Once created the geometry, the system must be subdivided. The mesh is not the same for the 1 kHz case and for the 11 kHz case. This is due to the different frequency that means different skin layer depth. For example, the 11 kHz case has 64% more triangles than the 1 kHz (250 k against 140 k), but for the first case, the triangle on the axis are larger than the ones in the second case, because they are more concentrated in the skin layer (See Figure 3-13 and Figure 3-14).

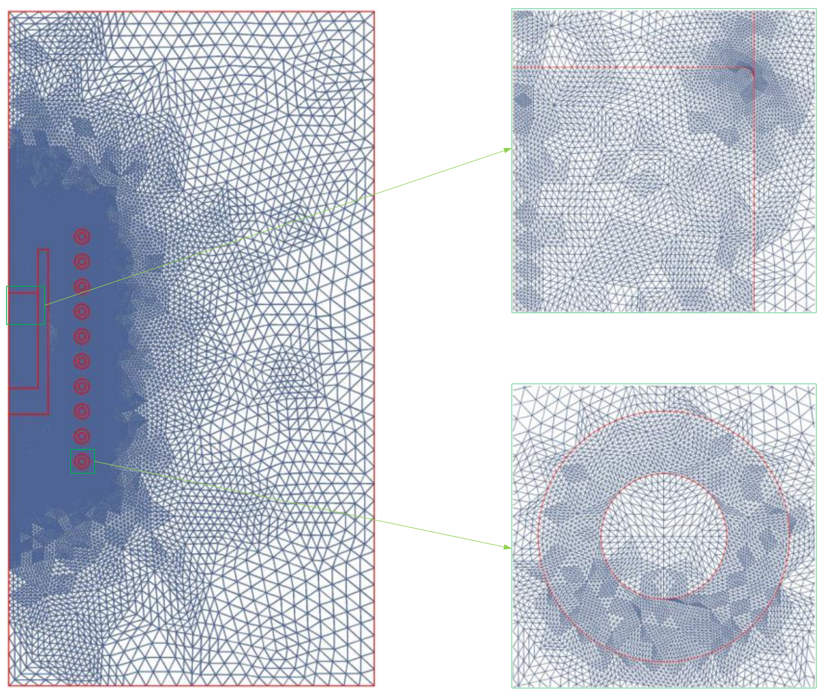
Figure 3-13: Mesh in the domain for the 1 kHz case (140 k triangles). Particular for the upper edge of the silicon and for the lower conductor

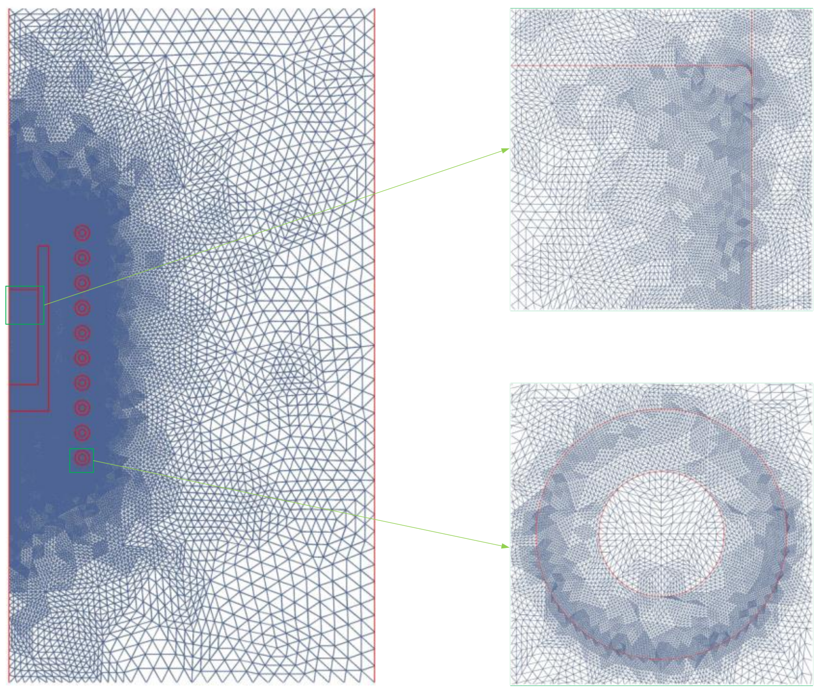
Figure 3-14: Mesh in the domain for the 11 kHz case (250 k triangles). Particular for the upper edge of the silicon and for the lower conductor

3.5.3 Current distribution

The effective current applied in the system is 385 A. It is to be remembered that the current introduced in MATLAB through the equation 3.54, is an imposed current. To this term must be added the induced current that opposes to the cause that creates it. The result is a reduction of the total current. It is possible to summarize all that as follows:

$$\boldsymbol{I}_{\boldsymbol{imposed}} + \boldsymbol{I}_{\boldsymbol{induced}} = \boldsymbol{I}_{\boldsymbol{total}} \tag{3.67}$$

Considering now the current induced, it is possible to rewrite it as a function of the imposed one, and in the case of a single coil system, it can be written as

$$\boldsymbol{I}_{\boldsymbol{induced}} = \frac{\boldsymbol{V}_{\boldsymbol{induced}}}{R} = -\frac{L}{R}\frac{d}{dt}\left(\boldsymbol{I}_{\boldsymbol{imposed}}\right) = \boldsymbol{X} \cdot \boldsymbol{I}_{\boldsymbol{imposed}} \tag{3.68}$$

where **X** is a complex number and in the general case it depends on the value of the current itself. Since our system is linear, the **X** factor is constant. The assumption of the linear system is considered to be a good initial approximation.

Taking into consideration the entire coil composed of ten turns, **X** takes the form of a ten by ten matrix, and the 3.68 becomes

$$\begin{bmatrix} \boldsymbol{I}_{\boldsymbol{induced}_1} \\ \boldsymbol{I}_{\boldsymbol{induced}_2} \\ \vdots \\ \boldsymbol{I}_{\boldsymbol{induced}_{10}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{X}_{1-1} & \boldsymbol{X}_{1-2} & \cdots & \boldsymbol{X}_{1-10} \\ \boldsymbol{X}_{2-1} & \boldsymbol{X}_{2-2} & \cdots & \boldsymbol{X}_{2-10} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{X}_{10-1} & \boldsymbol{X}_{10-2} & \cdots & \boldsymbol{X}_{10-10} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{I}_{\boldsymbol{imposed}_1} \\ \boldsymbol{I}_{\boldsymbol{imposed}_2} \\ \vdots \\ \boldsymbol{I}_{\boldsymbol{imposed}_{10}} \end{bmatrix} \tag{3.69}$$

The coefficients of the matrix can be simply found imposing 1 A current only in one turn at a time, and then computing the induced ones.

For the first turn for example, the first column of the matrix can be computed as follow:

$$\begin{bmatrix} \boldsymbol{I}_{\boldsymbol{induced}_1} \\ \boldsymbol{I}_{\boldsymbol{induced}_2} \\ \vdots \\ \boldsymbol{I}_{\boldsymbol{induced}_{10}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{X}_{1-1} & \boldsymbol{X}_{1-2} & \cdots & \boldsymbol{X}_{1-10} \\ \boldsymbol{X}_{2-1} & \boldsymbol{X}_{2-2} & \cdots & \boldsymbol{X}_{2-10} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{X}_{10-1} & \boldsymbol{X}_{10-2} & \cdots & \boldsymbol{X}_{10-10} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{X}_{1-1} \\ \boldsymbol{X}_{2-1} \\ \vdots \\ \boldsymbol{X}_{10-1} \end{bmatrix} \tag{3.70}$$

To find the current that must be imposed to have the total one equal to 385 A is now simple. Combining equation 3.67 with equation 3.68:

$$\left[\boldsymbol{I}_{\boldsymbol{imposed}}\right] + [\boldsymbol{X}] \cdot \left[\boldsymbol{I}_{\boldsymbol{imposed}}\right] = ([I_n] + [\boldsymbol{X}]) \cdot \left[\boldsymbol{I}_{\boldsymbol{imposed}}\right] = [\boldsymbol{I}_{\boldsymbol{total}}] \tag{3.71}$$

Where I_n is the identity matrix.

The current to impose will be

$$[\boldsymbol{I_{imposed}}] = ([I_n] + [\boldsymbol{X}])^{-1} \cdot [\boldsymbol{I_{total}}] \tag{3.72}$$

3.5.4 Solution

Once computed ψ, it is possible to compute the induction with the equation 3.55. The results of the two cases can be seen in Figure 3-15.

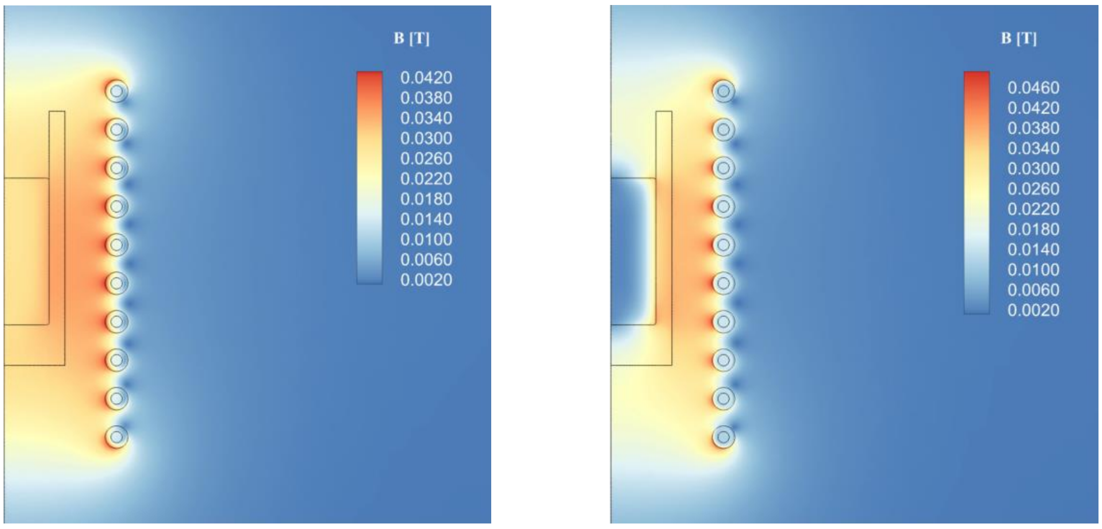

FIGURE 3-15: INDUCTION FIELD IN THE CRUCIBLE SYSTEM FOR THE 1 KHZ CASE (ON THE LEFT) AND FOR THE 11 KHZ CASE (ON THE RIGHT)

The maximum value of the induction field is almost the same for both cases, but for the higher frequency one, it is possible to see that the skin effect does not permit to the induction field to penetrate inside the silicon like in the other case.

The density of current can be computed with the equation 3.56. The result is shown in the Figure 3-16, where it is possible to see, first of all, how the current is concentrated in the corner for the 11 kHz, and second, the difference from the maximum values of the density of current. We can already have an idea on how the forces are distributed in the two cases.

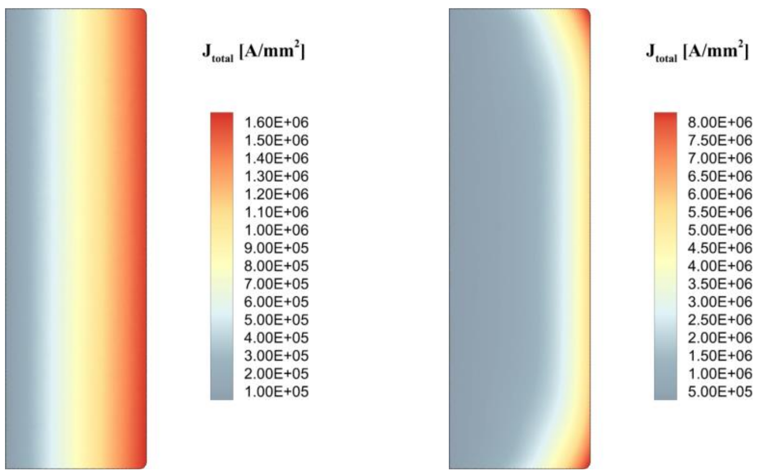

Figure 3-16: Density of current in the silicon for the 1 kHz case (on the left) and for the 11 kHz case (on the right)

Another important variable to analyse is the specific joule power transmitted to the silicon. This tells us how and where the silicon is being heated. Using the equation 3.61, are obtained the results presented in Figure 3-17.

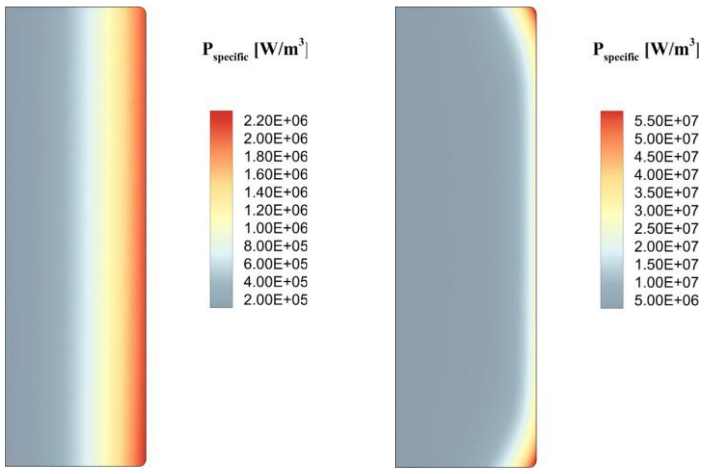

Figure 3-17: Specific joule power in the silicon for the 1 kHz case (on the left) and for the 11 kHz case (on the right)

As expected, the specific joule power is concentrated in the corners for the 11 kHz case, but despite that, much more power is transmitted in this case. Integrating the results obtained in Figure 3-17 in the volume of silicon, it is possible to see that for the 1 kHz case, there are 85 W dissipated as joule power, and 614 W in the other case.

The forces along the r and z axis (there are no forces in the φ axis due to the axisymmetric model implemented) can be computed with the equation 3.60. In the Figure 3-18, it is possible to see the results, with the vectors, to understand also the direction of the force. The first thing to notice is that the maximum magnitude is almost 10 times bigger in the 11 kHz case. The second thing to notice is the different distribution of the forces. For the 1 kHz case, the maximum is located in the center, while in the other case, in the corners.

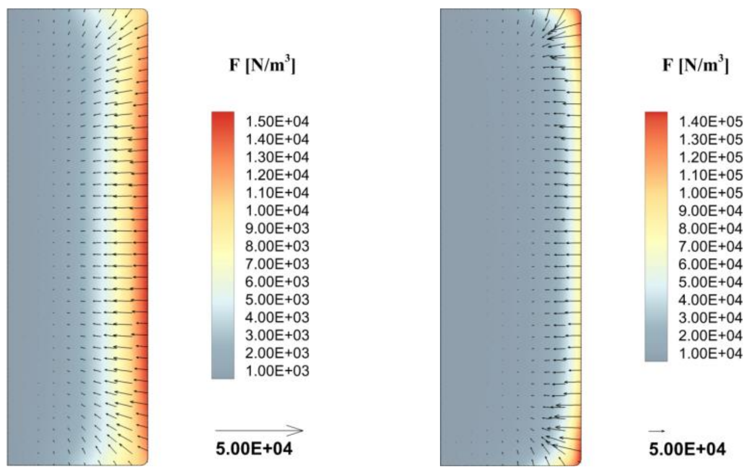

Figure 3-18: magnitude of the average forces in the silicon for the 1 kHz case (on the left) and for the 11 kHz case (on the right)

In Figure 3-19, it is possible to observe the particular of the r and z components of the forces.

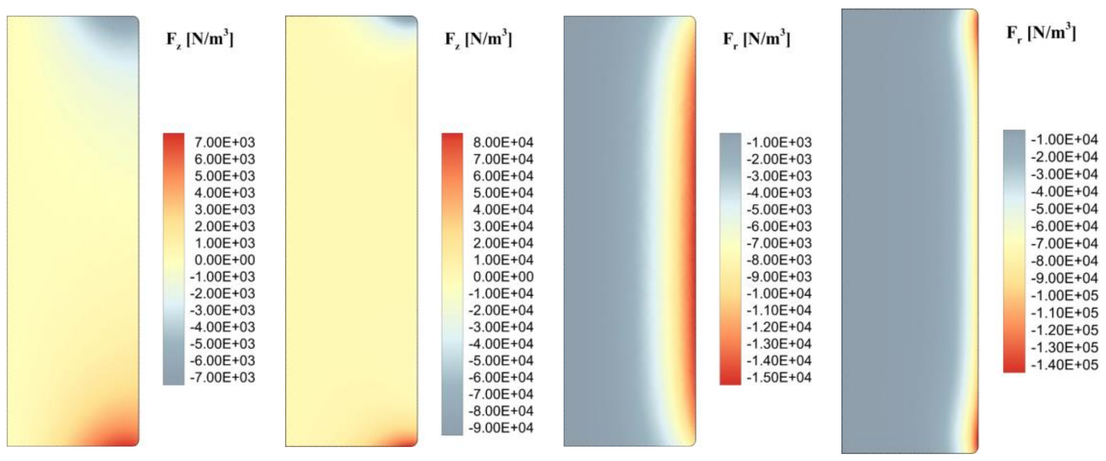

Figure 3-19: magnitude of the average forces in the silicon along the r and z axis for the 1 kHz case (first and third figures) and for the 11 kHz case (second and fourth figures

4 THERMAL PROBLEM

The aim of the thermal simulation is to better understand the system from the thermal point of view as well. In particular, the focus is on the conductors. The specific joule power is taken from the previous simulations and introduced in these simulations as heating source.

4.1 Thermal Theory

It is easy to derive the heat transfer equation directly in cylindrical coordinates.

The energy balance in a volume can generally be written as

$$E_{stored} = E_{in} - E_{out} + E_{generated} \tag{4.1}$$

With obvious meaning of the terms.

Taking a control volume in cylindrical coordinates and writing the energy flowing in and out from the faces, we have the situation described in Figure 4-1.

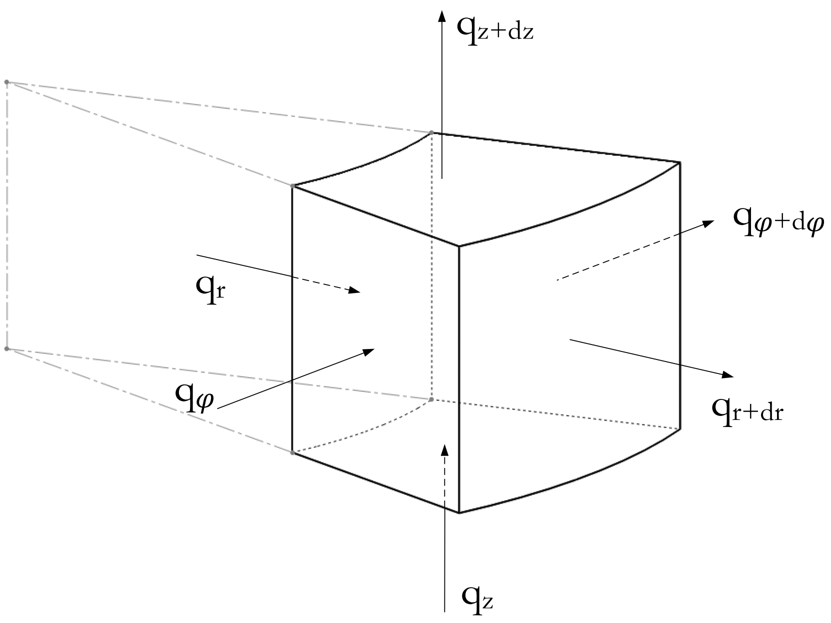

FIGURE 4-1: ENERGY FLOWING THROUGH THE CONTROL VOLUME

The energy generated in the volume can be expressed as:

$$E_{generated} = q \cdot V = q \cdot dr\, r\, d\varphi\, dz \tag{4.2}$$

Where q is the rate of specific energy generation.

On the other hand, the energy stored can be expressed as:

$$E_{stored} = \rho(dr\, r\, d\varphi\, dz\,)C_p \frac{\partial T}{\partial t} \tag{4.3}$$

Where ρ is the volume density and C_p is the heat capacity

Now, rewriting equation 4.1

$$\begin{aligned}\rho(dr\, r\, d\varphi\, dz\,)C_p \frac{\partial T}{\partial t} \\ = q_r + q_\varphi + q_z - q_{r+dr} - q_{\varphi+d\varphi} - q_{z+dz} + q dr\, r\, d\varphi\, dz\end{aligned} \tag{4.4}$$

The amount of energy flowing out to the control volume, can be expressed using the Taylor series expansion as follow:

$$\begin{aligned} q_{r+dr} &= q_r + \frac{\partial q_r}{\partial r} dr \\ q_{\varphi+d\varphi} &= q_\varphi + \frac{\partial q_\varphi}{\partial \varphi} d\varphi \\ q_{z+dz} &= q_z + \frac{\partial q_z}{\partial z} dz \end{aligned} \tag{4.5}$$

and the equation 4.4 becomes

$$\rho(dr\, r\, d\varphi\, dz\,)C_p \frac{\partial T}{\partial t} = -\frac{\partial q_r}{\partial r} dr - \frac{\partial q_\varphi}{\partial \varphi} d\varphi - \frac{\partial q_z}{\partial z} dz + q dr\, r\, d\varphi\, dz \tag{4.6}$$

We can write now the energy flowing q_r, q_φ and q_z through the Fourier's law:

$$q_r = -k \frac{\partial T}{\partial r} r\, d\varphi\, dz \tag{4.7}$$

$$q_\varphi = -k\frac{\partial T}{r\partial\varphi}dr\,dz$$

$$q_z = -k\frac{\partial T}{\partial z}dr\,r\,d\varphi$$

where k is the heat transfer coefficient. In 4.7 there are three equations. In all of them, the first part of the right term $\left(-\mathrm{k}\frac{\partial T}{\partial}\right)$ is the flux, while the second part is the cross section where this flux goes through.

Putting together 4.6 and 4.7:

$$\begin{aligned}\rho(dr\,r\,d\varphi\,dz\,)C_p\frac{\partial T}{\partial t}\\ &= +\frac{\partial}{\partial r}\left(k\,r\frac{\partial T}{\partial r}\right)dr\,d\varphi\,dz + \frac{\partial}{\partial\varphi}\left(k\,\frac{\partial T}{r\partial\varphi}\right)dr\,d\varphi\,dz \\ &+ \frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right)dr\,r\,d\varphi\,dz + q dr\,r\,d\varphi\,dz\end{aligned} \tag{4.8}$$

Now we can drop out the volume, to obtain the heat equation in cylindrical coordinates.

$$\rho C_p\frac{\partial T}{\partial t} = \frac{1}{r}\frac{\partial}{\partial r}\left(k\,r\frac{\partial T}{\partial r}\right) + \frac{1}{r^2}\frac{\partial}{\partial\varphi}\left(k\,\frac{\partial T}{r\partial\varphi}\right) + \frac{\partial}{\partial z}\left(k\,\frac{\partial T}{\partial z}\right) + q \tag{4.9}$$

The boundary conditions implemented in a thermal problem can be one of the following:

- *Constant surface temperature:* this is he the simplest boundary condition to deal with and can be expressed as:

$$T = T_s \tag{4.10}$$

- *Constant surface heat flux:* if we are introducing a known amount of heat in the system, we can use this boundary condition:

$$-k\nabla T = q \tag{4.11}$$

- *Adiabatic surface:* to implement a surface that does not permit to the energy to pass by, we impose the normal flux equal to zero as follow

$$-k\nabla T = 0 \tag{4.12}$$

- *Convection surface:* if we have a surface surrounded by a fluid, the right boundary condition to use is the following

$$-k\nabla T = h(T_\infty - T) \tag{4.13}$$

Where h is the convection coefficient from the surface and the fluid, and T_∞ is the temperature of the fluid far from the surface.

- *Radiation surface:* when the temperature is high, we must take into consideration the flux due to the radiation. This boundary condition can be expressed as:

$$-k\nabla T = \sigma\varepsilon\left({T_\infty}^4 - T^4\right) \tag{4.14}$$

Where σ is the Stefan-Boltzmann constant, ε is the emissivity coefficient of the material, and T_∞ is the temperature of the surrounding surfaces.

4.2 Thermal Model

In the thermal model of the conductors, the interest is not in the time dependence, but in the maximum temperature achieved after a long enough amount of time, so in the steady-state. For this reason, from the equation 4.9, the left term that express the time dependence must be neglected, and the equation becomes:

$$\frac{1}{r}\frac{\partial}{\partial r}\left(k\,r\frac{\partial T}{\partial r}\right) + \frac{1}{r^2}\frac{\partial}{\partial \varphi}\left(k\,\frac{\partial T}{r\partial \varphi}\right) + \frac{\partial}{\partial z}\left(k\,\frac{\partial T}{\partial z}\right) + q = 0 \tag{4.15}$$

Now, since we are considering a 2D axisymmetric case, the derivation along the φ axis of the temperature, is zero

$$\frac{1}{r}\frac{\partial}{\partial r}\left(k\,r\frac{\partial T}{\partial r}\right) + \frac{\partial}{\partial z}\left(k\,\frac{\partial T}{\partial z}\right) + q = 0 \tag{4.16}$$

Multiplying by r, the previous formula can be rewritten as:

$$-\frac{\partial}{\partial r}\left(k\,r\frac{\partial T}{\partial r}\right) - \frac{\partial}{\partial z}\left(k\,r\,\frac{\partial T}{\partial z}\right) = qr \tag{4.17}$$

It is easy now to see the similarity with the equation accepted in the PDE tool (equation (3.29)).

$$-\frac{\partial}{\partial r}\left(k\,r\frac{\partial T}{\partial r}\right)-\frac{\partial}{\partial z}\left(k\,r\frac{\partial T}{\partial z}\right)=qr$$

$$-\nabla\cdot(c\nabla u)+\underbrace{a}_{=0}u=f$$

FIGURE 4-2: COMPARISON FROM THE THERMIC MODEL AND THE ELLIPTIC EQUATION IN THE PDE TOOL

The q term in equation 4.17 is the specific heat source, which in this case, is given from the electromagnetic simulations.

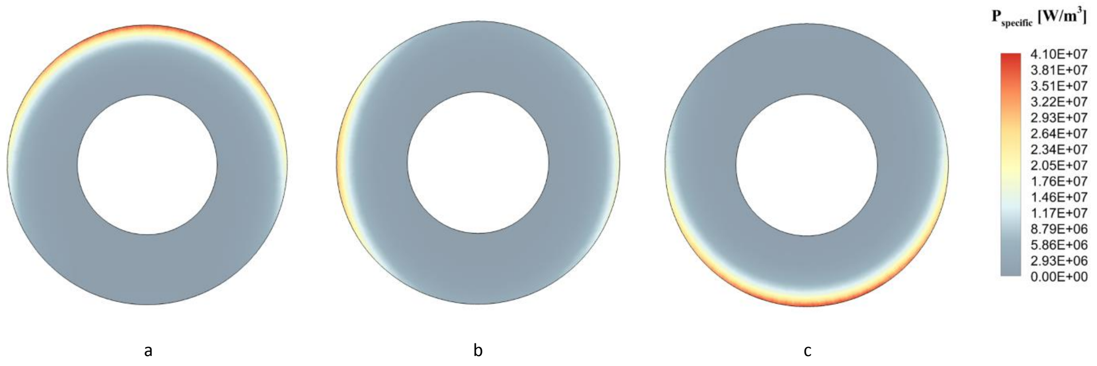

FIGURE 4-3: SPECIFIC JOULE POWER FOR THE FIRST CONDUCTOR (A), SIXTH CONDUCTOR (B) AND TENTH CONDUCTOR (C) FOR THE 11 KHZ CASE

In Figure 4-3 it is possible to see the specific joule power for the first conductor (the upper one), the sixth conductor (in the middle) and the tenth conductor (the lower one).

Because we are using the specific joule power from the electromagnetic simulations, we will use the same mesh, which for the thermal problem is fine enough. The way to proceed, is to solve a turn at a time, in order to compute the total energy given to the water and to update that for the next turn.

Internally the conductor, there is the hole to permit the water cooling. We can assume a velocity of the fluid of $v = 0.2\left[\frac{m}{s}\right]$, that corresponds to a flow rate of

$$fr = v\cdot t\cdot S = v\cdot 3600\cdot\pi\cdot r^2 = 0.0141\left[\frac{m^3}{h}\right] \qquad (4.18)$$

Considering then the inflow temperature of the water at 20°C, we can investigate to see if the flow is laminar or turbulent. The Reynolds number and the Prandtl number are given from:

$$Re = \frac{\rho\cdot v\cdot D}{\mu}\approx 1000 \qquad (4.19)$$

$$Pr = \frac{\mu \cdot cp}{k} \approx 7 \tag{4.20}$$

where:

- ρ is the density of the fluid $\left[\frac{kg}{m^3}\right]$
- μ is the dynamic viscosity of the fluid $\left[\frac{kg}{m \cdot s}\right]$
- cp is the specific heat capacity constant pressure $\left[\frac{J}{kg \cdot K}\right]$
- k is the thermal conductivity $\left[\frac{W}{m \cdot K}\right]$
- D is the diameter of the hole $[m]$

We know that the flow in a pipe is turbulent when the Reynolds number is greater than 2300 and the Prandtl number is greater than 0.7, but the circuit is short, so there is not enough space to become laminar. Therefore we can say that the flow is turbulent, which means that the thermal exchange coefficient is high enough to consider the internal surface at constant temperature (the temperature of the water). The boundary condition to express that, is the one in the equation 4.10. The temperature applied for this boundary conditions is calculated between one coil section and the next one.

Considering now the external surface, we have to implement the convection and radiation boundary condition. The convection coefficient h is computed for every turn and is given by:

$$h = \frac{Nu \cdot k}{De} \tag{4.21}$$

where:

- Nu is the Nusselt number given by

$$Nu = \left(\frac{0.6 + \left(0.387 \cdot Ra^{1/6}\right)}{\left(1 + \left(\frac{0.559}{Pr}\right)^{9/16}\right)^{8/27}} \right)^2 \tag{4.22}$$

- Ra is the Rayleigh number given by

$$Ra = Pr \cdot Gr \tag{4.23}$$

- Gr is the Grashof number given by

$$Gr = \frac{De^3 \cdot \rho^2 \cdot 9.81 \cdot \Delta T \cdot \beta}{\mu^2} \tag{4.24}$$

- De is the external diameter of the conductor
- ΔT is the difference from the surface temperature of the conductor and the temperature of the air supposed at 20 °C
- β is the expansion coefficient $\left[\frac{1}{K}\right]$

The other coefficients have the same meaning as in the case of the water.

Once calculated the h coefficient, we can express the boundary condition on the external boundary simply putting together the equation 4.13 and 4.14. The result is:

$$-k\nabla T = h(T_\infty - T) + \sigma\varepsilon\left({T_\infty}^4 - T^4\right) \tag{4.25}$$

Due to the fourth power in the temperature of this boundary condition, a nonlinear solver must be used for this problem.

Once the solution is computed, it is easy to calculate how much power goes in the water and how much in the ambient. This can be done computing the flux in the solution, but since we know the temperature in all the points on the external surface, it is easier and more accurate to use the formula 4.25 to compute the power that goes outside; then, knowing the total power generated in every turn, subtract the previous result to obtain the power transmitted to the water.

The characteristics of the materials used in the thermal simulations are grouped in the next table.

	Thermal conductivity $k\left[\frac{W}{m*K}\right]$	*Specific Heat Capacity at Constant Pressure* $cp\left[\frac{J}{Kg*K}\right]$	*Dynamic viscosity* $\mu\left[\frac{kg}{m*s}\right]$	*Density* $\rho\left[\frac{kg}{m^3}\right]$	*Expansion coefficient* $\beta\left[\frac{1}{K}\right]$
Copper	391	–	–	–	–
Air	$2.624*10^{-2}$*	1004.9*	$1.846*10^{-5}$*	1.177*	$1.75*10^{-3}$*
Water	–	$4.1818*10^{3}$**	–	998.21**	–

**the coefficients in the table for the air are given at 300K. In the simulations, , the coefficients are interpolated at the exact temperature*

** *the coefficients in the table for the water are given at 20°C. In the simulations, the coefficients are interpolated at the exact temperature*

TABLE 4-1: CHARACTERISTICS OF THE MATERIALS USED IN THE SIMULATIONS

4.3 Solution

The solutions are obtained varying the water velocity between 0.05 and 0.5 m/s. This is done in order to see the changes in temperature through the coil varying the water cooling.

With the minimum velocity, the temperatures are shown in Figure 4-4 (for convenience, the figure is rotated 90° clockwise and divided in two parts).

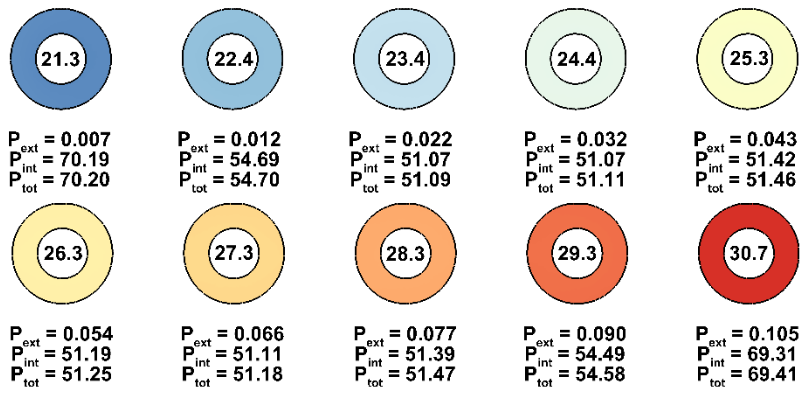

FIGURE 4-4: TEMPERATURES AND POWERS IN THE COIL WITH A WATER VELOCITY OF 0.05 M/S

From this figure, it is possible to see that the power transmitted to the ambient is very low compared with the one given to the water. This is because the temperatures reached are low to transmit a large amount of power with convection and radiation. In addition, as expected, the joule power generated in the lower and upper conductor (1 and 10) are greater than the other because of the skin effect (the first conductor has almost 40% more power than the third one). Another thing to notice is that the temperature gained by the water travelling through the coil is only 10.7°C even with a velocity of the fluid so low. This is due to the large heat capacity of the water.

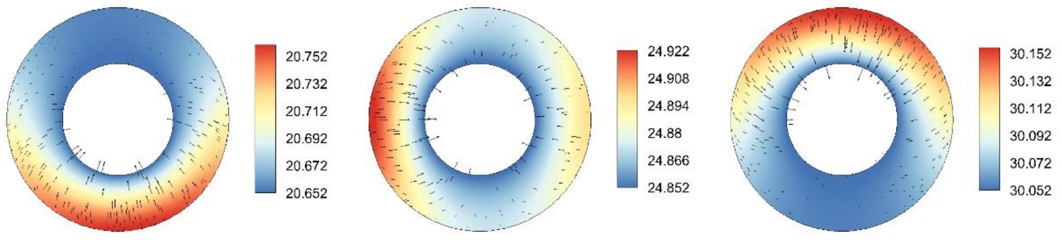

FIGURE 4-5: PARTICULAR OF THE TEMPERATURE FOR THE LOWER CONDUCTOR (1), MIDDLE CONDUCTOR (5) AND UPPER CONDUCTOR (10)

The high thermal coefficient of the copper makes that the maximum difference in temperature in the cross section of the coil is very small (a tenth of a degree) as can be seen in Figure 4-5. In the same figure are also plotted the vectors corresponding to the gradient of the temperature (the flux). From these can be seen that almost all the thermal flux goes inside the conductor, in the water.

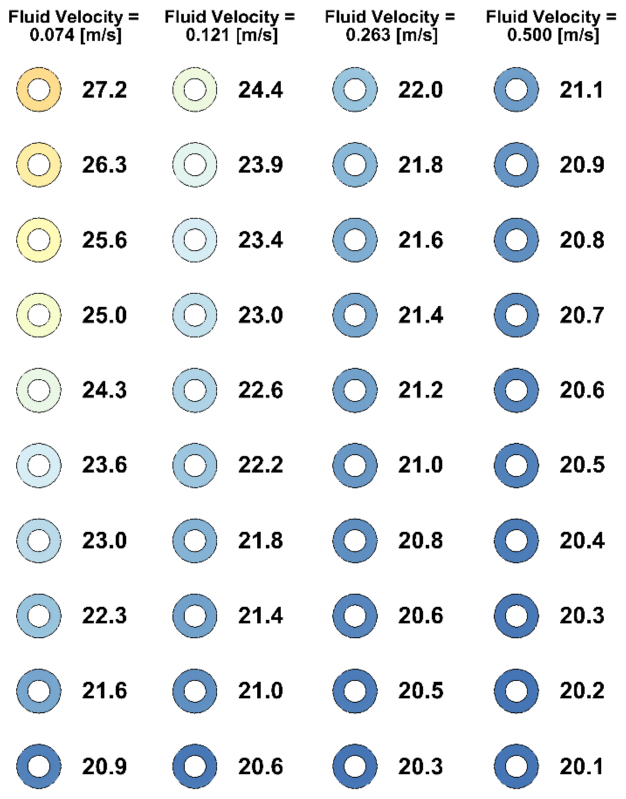

Figure 4-6: Details of the temperatures in the coil varying the velocity of the water

In Figure 4-6 it is possible to observe other results obtained varying the velocity of the water. For a fluid velocity of 0.5 $[m/s]$, that corresponds to a flow rate of 2.35 $[l/min]$, the ΔT of the water is only $1.1°C$.

5 Conclusions

The aim of this study was to take confidence in the use of the PDEtool and to study the system for the separation of impurities in the molten silicon by the application of EM induction. In particular, the problem has been solved using finite element methods on a 2D axisymmetric model, and implemented in the MATLAB environment. The code has been written from scratch, and also a useful function for the export of the result from MATLAB to Tecplot has been created (see Appendix 3, function "pdetool2tecplot").

For the electromagnetic problem, the focus has been taken to the 1 kHz and 11 kHz power supply cases. The results show that the joule power generated in the silicon are 85 W for the former case and 614 W for the latter. Due to the skin effect, the distribution of this power is very different in the two cases. In the first one, it is almost constant for a fixed value of the r axis, while for the 11 kHz case, it is concentrated in the corners. Taking a look at the forces, it is possible to see that despite the fact that in a period of the current they act inside as well as outside the silicon, there is an average component that compress the molten metal and push the impurities at the boundary. Moreover, the maximum magnitude of the forces are almost ten times bigger in the 11 kHz case, while the distribution is similar to the distribution of the specific joule power.

Once solved the electromagnetic problem, the focus has been taken on the thermal problem. The input were the specific joule power computed previously in the 11 kHz case. To see the changes in temperature through the coil varying the water cooling, the solutions are obtained varying the fluid velocity between 0.05 and 0.5 m/s. The results show that almost all of the power generated in the turns, goes in the water, and a very small amount is dissipated in the ambient due to convection and radiation. It is also possible to see that, a small fluid velocity in enough to contain the difference in temperature of the water between inlet and outlet (with a velocity of 0.05m/s, the temperature gained by the water travelling through the coil is only 10.7°C).

Previous results using spectral methods were used, so it was possible to compare and validate the MATLAB FE solutions.

In conclusion, this study confirm that using the electromagnetic field is a valid approach for the EM separation of impurities from the molten silicon.

6 REFERENCES

[1] International Energy Agency, *World Energy Outlook 2013*, 2013.

[2] European Photovoltaic Industry Association, *Global Market Outlook For Photovoltaic 2013-2017*, 2013, pp. 5-14. http://www.epia.org/fileadmin/user_upload/Publications/GMO_2013_-_Final_PDF.pdf.

[3] A. Luque and S. Hegedus, Handbook of photovoltaic science and engineering, USA: Wiley, 2011, pp. 169-215.

[4] V. Bojarevics, K. Pericleous, M. Wickins and R. Harding, "The Development and Experimental Validation of a Numerical Model of an Induction Skull Melting Furnace," *Metallurgical and Material Transactions B*, vol. 35B, pp. 1-19, 2003.

[5] S. Asai, "Recent development and prospect of electromagnetic processing of materials," *Science and technology of advanced materials*, pp. 191-200, 2000.

[6] V. Bojarevics and A. Roy, "Effect on Magnetic Forces on Bubble Transport and MHD Stability of Aluminium Electrolysis Cells," *Magnetohydrodynamics*, vol. 48, no. 1, pp. 125-136, 2012.

[7] E. Scheil, *Metallkd*, vol. 34, 1942.

[8] A. Çiftja, "Solar silicon refining; Inclusions, settling, filtration, wetting," Trondheim, 2009.

[9] S. Makarov, R. Ludwig and D. Apelian, "Electromagnetic Separation Techniques in Metal Casting. I. Conventional Methods," *IEEE Transactions on Magnetics*, vol. 36, no. 4, 2000.

[10] "Wikipedia," [Online]. http://en.wikipedia.org/wiki/MATLAB. [Accessed 14 May 2014].

[11] "Ordinary Differential Equations," MathWorks, [Online]. http://www.mathworks.co.uk/help/matlab/math/ordinary-differential-equations.html. [Accessed 14 May 2014].

[12] MathWorks, "pdepe," [Online]. http://www.mathworks.co.uk/help/matlab/ref/pdepe.html. [Accessed 14 May 2014].

[13] MathWorks, "pdetool," [Online]. http://www.mathworks.co.uk/help/pde/ug/pdetool.html. [Accessed 14 May 2014].

[14] MathWorks, "PDE Toolbox Help".

[15] V. Bojarevics, K. Pericleous and M. Cross, "Modeling the Dynamics of Magnetic Semilevitation Melting," *Metallurgical and material transactions B*, vol. 31B, pp. 179-189, 2000.

7 Appendices

APPENDIX 1

Stokes' theorem

Let be

S: an oriented, piecewise smooth surface

∂S: a curve that bounds S

F: a vector field whose components have continuous derivatives in an open region of $\mathbb{R}^3$ containing S.

Then:

$$\iint_S (\nabla \times \boldsymbol{F})\bullet d\boldsymbol{S} = \oint_{\partial S} \boldsymbol{F}\bullet d\boldsymbol{l} \tag{7.1}$$

Divergence theorem

Let V be a subset of $\mathbb{R}^n$ which is compact and has a piecewise smooth boundary S. If **F** is a continuously differentiable vector field defined on a neighbourhood of V, then we have:

$$\iiint_V (\nabla \times \boldsymbol{F}) dV = \oiint_S \boldsymbol{F} \bullet d\boldsymbol{S} \tag{7.2}$$

Therefore, with this theorem is possible to transform a volume integral (left side) in a surface integral (right side) where the left side represent the total of the source in the volume while the right side represent the total flow across the surface.

Quasi-stationary approximation

We can consider the third Maxwell's equation (3.25) we know that it is possible to simplify in:

$$\nabla \times \boldsymbol{H} = \boldsymbol{J} \tag{7.3}$$

However, we can obviously ask ourselves when this approximation is legitimate.

The possibility to neglect the time derivative depends not only from the magnitude of the vector, but also from the velocity of variation of that vector. Therefore, neglecting the time derivative of the displacement field is much more acceptable when the variation is slow.

To neglect the time derivative means to neglect the retard of propagation, namely the temporal variations of the variables that propagates instantaneously in all the points of the system. It is obvious that, for neglecting the delay time, the system must be considered geometrically limited.

Given a pair of points P and P_0 belonging to the system under consideration, we call:

$$r_{max} = max(|P - P_0|) \tag{7.4}$$

$$t_{max} = \frac{r_{max}}{c} \tag{7.5}$$

where c is the speed of light in vacuum.

If the relations of negligibility is verified for distances equal to r_{max}, even more so it will be for all other value of the distance r.

Neglecting the time delay means, as said, suppose that the causes that originate the field do not vary in the time interval under consideration, namely:

$$\boldsymbol{J}(P, t - t_{max}) \approx \boldsymbol{J}(P, t) \tag{7.6}$$

Considering the series expansion 7.6 can be rewritten in the form:

$$\left|\frac{\partial \boldsymbol{J}(P,t)}{\partial t}\frac{r_{max}}{c}\right| \ll |\boldsymbol{J}(P,t)| \tag{7.7}$$

If we consider the sinusoidal case with pulse ω, it is possible to write:

$$\boldsymbol{J}(P,t) = \boldsymbol{J}(P)cos(\omega t + \varphi) \tag{7.8}$$

$$\frac{\partial \boldsymbol{J}(P,t)}{\partial t} = -\omega \boldsymbol{J}(P)sin(\omega t + \varphi) \tag{7.9}$$

So 7.7 is verified when

$$\left|\omega \frac{r_{max}}{c}\right| \ll 1 \tag{7.10}$$

This is the condition to neglect the time derivative of the displacement field in the third Maxwell's equation.

APPENDIX 2

Identities in Cartesian coordinates

- Gradient of a scalar

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right) \tag{7.11}$$

- Divergence of a vector

$$\nabla \boldsymbol{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z} \tag{7.12}$$

- Curl of a vector

$$\nabla \times \boldsymbol{F} = \begin{vmatrix} \boldsymbol{e_x} & \boldsymbol{e_y} & \boldsymbol{e_z} \\ \dfrac{\partial}{\partial x} & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial z} \\ F_x & F_y & F_z \end{vmatrix} \tag{7.13}$$

Identities in cylindrical coordinates

- Gradient of a scalar

$$\nabla f = \left(\frac{\partial f}{\partial r}, \frac{1}{r}\frac{\partial f}{\partial \varphi}, \frac{\partial f}{\partial z}\right) \tag{7.14}$$

- Divergence of a vector

$$\nabla \boldsymbol{F} = \frac{1}{r}\frac{\partial}{\partial r}(rF_r) + \frac{1}{r}\frac{\partial F_\varphi}{\partial \varphi} + \frac{\partial F_z}{\partial z} \tag{7.15}$$

- Curl of a vector

$$\nabla \times \boldsymbol{F} = \frac{1}{r}\begin{vmatrix} \boldsymbol{e}_r & r\boldsymbol{e}_\varphi & \boldsymbol{e}_z \\ \frac{\partial}{\partial r} & \frac{\partial}{\partial \varphi} & \frac{\partial}{\partial z} \\ F_r & rF_\varphi & F_z \end{vmatrix} \tag{7.16}$$

Trigonometric Identities

$$sin^2(u) = \frac{1 - cos(2u)}{2} \tag{7.17}$$

$$cos^2(u) = \frac{1 + cos(2u)}{2} \tag{7.18}$$

$$sin(u)cos(u) = \frac{1}{2}sin(2u) \tag{7.19}$$

APPENDIX 3

Main function “Solve”

```
function Solve
clc
clear variables
%Choice = 0     compute:    geometry  ->  mesh  ->   solution  ->   calculation  ->
tecplot write
%Choice = 1     compute:                             solution  ->   calculation  ->
tecplot write
%Choice = 2     compute:                                            calculation  ->
tecplot write
Choice = 2;     nomefile = '11000Hz';    %name for the output
% Creation of the name of the matrices, so if I chance theinput file, I do
% not clear the other solution computed
Result_name = ['(',nomefile,')_Result.mat'];

if ( Choice==0 ) || ( Choice==1 )
    if ( Choice==0 )
%% GEOMETRY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ti = clock;
        fprintf('\n   ---Geometry construction---');
        %Construction of geometry matrices
        [ b, g, Materials, Turns ] = bgMT_Construction2( 0 );
        %plot the geometry
        figure(1);
        pdegplot(g)
        axis equal
        %load the X matrix before delete it so if the geometry is not
        %changed, I can use that one instead of recompute it again
        if exist(Result_name,'file')==2
            load(Result_name,'X')
        end
        save(Result_name,'Materials','Turns','b','g');
        tf = clock;
        fprintf('    COMPLETED in %4.1f s\n',etime(tf,ti));
%% MESH %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        ti = clock;
        fprintf('\n   ---Mesh construction---\n');
        MaxIter = 15;
        Iimposed = Turns(2,:);%The current imposed is the peak value
        [ a, c, f, ~ ] = acfdConstruction( Iimposed );
        [~,p,e,t]=adaptmesh(g,b,c,a,f,'MesherVersion','R2013a','Ngen',MaxIter);
        [p,e,t]=refinemesh(g,p,e,t);    [p,e,t]=refinemesh(g,p,e,t);
        %Plot the mesh
        figure(2);
        pdemesh(p,e,t)
        drawnow
        axis equal
        fprintf('Number of triangles in the mesh = %6.f\n', size(t,2));
        %save the area of triangles, so when I need it, I don't have to
        %compute it again
        [A,~,~,~] = pdetrg(p,t);
        save(Result_name,'p','e','t','A','-append');
        tf = clock;
        fprintf('    COMPLETED in %4.1f s\n',etime(tf,ti));
%% X MATRIX %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% X*Iimposed = Iinduced
        ti = clock;
        nt = size(Turns,2);
        %The computation of the X matrix takes a while, so it is computed
        %only if needed
        compute = 1;
        if exist('X','var')==1
            if size(X,2)==nt
                C = eye(nt) + X;
                Iideal = ones(1,nt);
                Iimposed = +( C\Iideal.' ).';
                [ a, c, f, ~ ] = acfdConstruction( Iimposed );
                u = assempde(b,p,e,t,c,a,f);
                [I, ~] = ComputeI( u );%vector of currents obtained in the turns
                Ireal = Iimposed + I;
                error = (Ireal-Iideal)./Ireal;
                if sum( abs( error ) ) < 1e-4
                    compute = 0;
                end
            else
                compute = 1;
            end
        else
            compute = 1;
        end
        %Computation of X matrix
        if compute == 1
            fprintf('\n   ---Matrix X construction---\n');
            X = zeros(nt);
            for k=1:nt
                Iimposed = zeros(1,nt);
                Iimposed(1,k) = 1;
                [ a, c, f, ~ ] = acfdConstruction( Iimposed );
                u = assempde(b,p,e,t,c,a,f);
                [I, ~] = ComputeI( u );
                X(:,k) = I;
                fprintf('%2.0f ',nt-k);
            end
        end
        save(Result_name,'X','-append');
        tf = clock;
        fprintf('    COMPLETED in %4.1f s\n',etime(tf,ti));
    else
        if exist(Result_name,'file')~=2
            warndlg(['The matrices does not exist for ',nomefile,'.'], ['Error in
',mfilename,'.m.']);
            return
        end
        load(Result_name,'Materials','Turns','b','g','p','e','t','A','X')
    end
%% SOLUTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ti = clock;
    fprintf('\n   ---Computation of solution---\n');
    % construction of the matrix with the currents I want to impose
    turns_index = Turns(1,:);
    nt = size(turns_index,2);
    Iideal = Turns(2,:);
    C = eye(nt) + X;
    Iimposed = +( C\Iideal.' ).';
    [ a, c, f, ~ ] = acfdConstruction( Iimposed );
    u = assempde(b,p,e,t,c,a,f);
    [I, V] = ComputeI( u );        %vector of currents obtained in the turns
    Ireal = Iimposed + I;
    error = (Ireal-Iideal)./Ireal;                      %computation of the error
    fprintf('Error on the current = %6.5f \n', sum(abs(error)));
    save(Result_name,'u','I','Iimposed','V','error','-append');
    tf = clock;
    fprintf('    COMPLETED in %4.1f s\n',etime(tf,ti));
elseif ( Choice==2 )
    if exist(Result_name,'file')~=2
        warndlg(['The matrices does not exist for ',nomefile,'.'], ['Error in
',mfilename,'.m.']);
        return
    end

load(Result_name,'Materials','Turns','b','g','p','e','t','A','u','I','Iimposed','V','
error')
else
    warndlg('Variable Choice must be an integer from 0 to 2', ['Error in
',mfilename,'.m.']);
    return
end
%% CALCOLI %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nt = size(t,2);%number of triangles
np = size(p,2);%number of points
freq = Turns(5,1);
Ireal = Iimposed + I;
omega = 2*pi*freq;
sd = unique(t(4,:));
ns = size(sd,2);%number of subdomains
mt = unique(Materials(1,:));
nm = size(mt,2);%number of materials
PSIfi = u;%evaluated on points
PSIfi_t = pdeintrp(p,t,PSIfi);%interpolation on triangles
Afi_p = PSIfi./(p(1,:))';
%In the previous formula there is a division by zero on the axis, so the
%nans must be replaced
[Afi_p] = replace_nan (p, e, t, Afi_p);
Afi_t = pdeintrp(p,t,Afi_p);
%gradient of PSI function
[Dr_PSIfi,Dz_PSIfi] = pdegrad(p,t,PSIfi);%evaluated on triangles
%coordinates of the center of triangles
xpts=(p(1,t(1,:))+p(1,t(2,:))+p(1,t(3,:)))/3;
```

```
ypts=(p(2,t(1,:))+p(2,t(2,:))+p(2,t(3,:)))/3;
%Induction
Br = abs( -Dz_PSIfi./xpts );%evaluated on triangles
Bz = abs( Dr_PSIfi./xpts );%evaluated on triangles
%Forces
Fr = (imag(PSIfi_t).*real(Dr_PSIfi) -
real(PSIfi_t).*imag(Dr_PSIfi)).*omega.*Materials(3,t(4,:))./(2*xpts.^2);
Fz = (imag(PSIfi_t).*real(Dz_PSIfi) -
real(PSIfi_t).*imag(Dz_PSIfi)).*omega.*Materials(3,t(4,:))./(2*xpts.^2);
sigma_t = Materials(3,t(4,:));
%J induced (instant value)
Jind = -1i*omega.*sigma_t.*Afi_t;
%J imposed
Jimp = zeros(1,nt);
for k=1:size(Turns,2)
    Jimp1 = Iimposed(1,k)/Turns(3,k);
    i_t = pdesdt(t,Turns(1,k));
    Jimp(1,i_t) = Jimp1;
end
%J real
Jreal = Jimp + Jind;
%J real (mean value)
Jreal_m = abs(Jreal)./sqrt(2);
Ireal_mean = abs(mean(Ireal))/sqrt(2);
%Magnitude of induction field
B_magn = sqrt(Br.^2 + Bz.^2);
%Magnitude of force
F = sqrt(Fr.^2 + Fz.^2);
%Specific power
Pspec = Jreal_m.^2./Materials(3,t(4,:));
Pspec(isnan(Pspec))=0;
i_t_s = pdesdt(t,2);
%The power is integrated in the silicon
[ ~, Power ] = u_Integral1( p, t, Pspec, i_t_s );
%% WRITING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The results are written to a file for the plotting of the results in
%tecplot
pdetool2tecplot (nomefile, p, e, t, Br, Bz, B_magn, Jreal_m, Fr, Fz, F, Pspec,...
                    'VarName', {'r' 'z' 'Br' 'Bz' 'B' 'J' 'Fr' 'Fz' 'F' 'Pspec'},...
                    'ZonesName',{'Air' 'Silicon' 'Graphite' 'Copper' 'Copper'
'Copper' 'Copper' 'Copper' 'Copper' 'Copper' 'Copper' 'Copper' 'Copper' 'Water'
'Water' 'Water' 'Water' 'Water' 'Water' 'Water' 'Water' 'Water' 'Water'},...
                    'DSAD',{'Frequency'; 'Current'; 'Power'},{freq; Ireal_mean; Power
},...
                    'Title', nomefile);
%For the vector plotting of the forces, an additional file is written with
%a grid mesh
nomefile_grid = [nomefile,'_grid'];
xgrid = linspace(0,1,10);
ygrid = linspace(0,1,40);
pdetool2tecplot (nomefile_grid, p, e, t, Fr, Fz,...
                    'grid','xy',xgrid,ygrid,...
                    'VarName', {'r' 'z' 'Fr1' 'Fz1'},...
                    'Subdomain', 2,...
                    'ZonesName', {'Silicon_grid'},...
```

```
                      'Title', nomefile_grid);
end
%% SUBFUNCTIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [var] = replace_nan (p, e, t, var)
    var = var.';
    np = size(p,2);
    nt = size(t,2);
    nvar = size(var,2);
    i_nan = find(isnan(var)==1);
    n_nan = size(i_nan,2);

    while n_nan~=0
        if size(nvar)==nt
            for k=1:n_nan
                tri = i_nan(1,k);
                ntl=pdeent(t,tri);
                sum = 0;
                h = 0;
                for j=1:size(ntl,2)
                    if ~isnan(var(1,ntl(1,j)))
                        sum = sum + var(1,ntl(1,j));
                        h = h + 1;
                    end
                end
                if sum~=0
                    sum = sum / h;
                    var(1,tri) = sum;
                end
            end
        else
            for k=1:n_nan
                point = i_nan(1,k);
                i_e1 = ( find( e(1,:)==point ) );
                i_e2 = ( find( e(2,:)==point ) );
                i_e = unique([i_e1, i_e2]);
                i_p = unique( [e(1,i_e) e(2,i_e)] );
                sum = 0;
                h = 0;
                for j=1:size(i_p,2)
                    if ~isnan(var(1,i_p(1,j)))
                        sum = sum + var(1,i_p(1,j));
                        h = h + 1;
                    end
                end
                if sum~=0
                    sum = sum / h;
                    var(1,point) = sum;
                end
            end

        end
        i_nan = find(isnan(var)==1);
        n_nan = size(i_nan,2);
    end
    var = var.';
end
```

Function “acfdConstruction”

```
function [ a, c, f, d ] = acfdConstruction( Iimposed )
%   Given the matrix Materials.mat and Turns.mat created in
%   bgMT_Construction2.m, the function creates the matrices "a", "c", "f" e
%   "d" for the computation of the solution.
%   For a reference on the form of Materials.mat and Turns.mat, see the
%   function bgMT_Construction2.m

%Take the matrices from the caller function
Materials = evalin('caller','Materials');
Turns = evalin('caller','Turns');
freq = Turns(5,1);
omega = 2*pi*freq;
% "a", "c", "f" e "d" are char vectors. They are initialized empty
a = '';
c = '';
f = '';
d = '';
%Start writing in them
for i=1:size(Materials,2)
    %Creating "a"
    if Materials(3,i)~=0
        a = [a,strcat('i.*',num2str(Materials(3,i)*omega,'%g'),'./x')];
    else
        a = [a,'0.0'];
    end
    %Creating "c"
    c = [c,strcat('1./(x.*',num2str(Materials(2,i),'%g'),')')];
    %Creating "f"
    T_i = find(Turns(1,:)==i);
    if isempty(T_i)
        f = [f,'0.0'];
    else
        f = [f,num2str(Iimposed(1,T_i)/Turns(3,T_i))];
    end
    %Creating "d"
    d = [d,'1.0'];
    %In the vectors, between two coefficients, there must be an exclamation
    %mark
    if i~=size(Materials,2)
        a = [a,'!'];
        c = [c,'!'];
        f = [f,'!'];
        d = [d,'!'];
    end
end
%the vectors must have the same size, so add some spaces in the shorter
```

```
%ones
k = max( [size(a,2); size(c,2); size(d,2); size(f,2)] );
for i=size(a,2):k-1
    a = [a,' '];
    i = i+1;
end
for i=size(c,2):k-1
    c = [c,' '];
    i = i+1;
end
for i=size(d,2):k-1
    d = [d,' '];
    i = i+1;
end
for i=size(f,2):k-1
    f = [f,' '];
    i = i+1;
end
end
```

Function “bgMT_Construction2”

```
function [ b, g, Materials, Turns ] = bgMT_Construction2( fig_yn )
%This function creates the matrices used in the solver ("b", "g") and for
%the computations
%fig_yn = 1 to plot the geometry
%% DATA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%geometry data
S_crd = [    0       2;
            -3       3.53]/100;
GB_crd = [   0       2.7;
            -4.8    -3]/100;
GL_crd = [   2       2.7;
            -3         6.5]/100;
T_crd = [    5       5;
            -8         7.4]/100;
nt = 10;%Number of turns
dc = 1/100;%Diameter of conductor
dc1 = 0.5/100;%diameter of hole in conductor [m]
racc = 0.1/100;%radius of rounded angle in silicon
% general data
freq = 11000;
mu0 = 4*pi*1e-7;

%Material 1: data air
murA = 1; muA = mu0*murA;%magnetic permeability
sigmaA = 0;%Conductivity [Ohm*m]
densityA = nan;%mass density [kg/m^3]
ThCoA = nan;%thermal conductivity k [W/m*K]
SpHeCaA = nan;%specific heat capacity Cp [J/kg*K]
```

```
alphaA = nan;%coefficient of temperature for conductivity [k^-1]
tsA = nan;%temperature at which the sigma is given [K]
epsilonA = nan;%Emissivity

%Material 2: data silicon
murS = 1; muS = mu0*murS;
sigmaS = 1.23e6;
densityS = 2560;
ThCoS = 149;
SpHeCaS = 710;
alphaS = -70e-3;
tsS = 1000;
epsilonS = 0.3;

%Material 3: data graphite
murG = 1; muG = mu0*murG;
sigmaG = 8.65e4;
densityG = 2160;
ThCoG = 80;
SpHeCaG = 710;
alphaG = -5e-3;
tsG = 320;
epsilonG = 0.75;

%Material 4: data copper
murC = 1; muC = mu0*murC;
sigmaC = 4.1e7;
densityC = 8920;
ThCoC = 391;
SpHeCaC = 390;
alphaC = 3.93e-3;
tsC = 320;
epsilonC = 0.03;

IC = 385*sqrt(2);%Peak value of the current
SC = pi*dc^2/4-pi*dc1^2/4;%Section of the conductor

%data water
murW = 1; muW = mu0*murW;
sigmaW = 0;
densityW = 1000;
ThCoW = 580;
SpHeCaW = 4185;
alphaW = 0;
tsW = 0;
epsilonW = nan;
%% GEOMETRY COEFFICIENTS%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Center coordinates of turns
Tx_crd = linspace(T_crd(1,1),T_crd(1,2),nt);
Ty_crd = linspace(T_crd(2,1),T_crd(2,2),nt);
%Limit of the entities in the domain for the creation of the domain itself
xmax = max( [max(S_crd(1,:)), max(GB_crd(1,:)), max(GL_crd(1,:)), max(S_crd(1,:)),
max(T_crd(1,:))] );
ymin = min( [min(S_crd(2,:)), min(GB_crd(2,:)), min(GL_crd(2,:)), min(S_crd(2,:)),
min(T_crd(2,:))] );
```

```
ymax = max( [max(S_crd(2,:)), max(GB_crd(2,:)), max(GL_crd(2,:)), max(S_crd(2,:)),
max(T_crd(2,:))] );
h = ymax - ymin;
A_crd = [    0,       xmax*5;     %creation of domain
             ymin-h, ymax+h];
num_seg = 16+4*nt;
%% CREATION OF g %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%"g" is the matrix that describes the geometry
%Matrix with coordinates of points [x,y]
Points_rept = [ min(A_crd(1,:))         max(A_crd(2,:));        %Point 1
                max(A_crd(1,:))         max(A_crd(2,:));        %Point 2
                max(A_crd(1,:))         min(A_crd(2,:));        %Point 3
                min(A_crd(1,:))         min(A_crd(2,:));        %Point 4
                min(A_crd(1,:))         min(GB_crd(2,:));       %Point 5
                min(A_crd(1,:))         max(GB_crd(2,:));       %Point 6
                min(A_crd(1,:))         max(S_crd(2,:));        %Point 7
                max(GL_crd(1,:))        min(GB_crd(2,:));       %Point 8
                max(GL_crd(1,:))        max(GL_crd(2,:));       %Point 9
                min(GL_crd(1,:))        max(GL_crd(2,:));       %Point 10
                max(S_crd(1,:))         max(S_crd(2,:))-racc;   %Point 11
                max(S_crd(1,:))         min(S_crd(2,:))+racc;   %Point 12
                max(S_crd(1,:))-racc    max(S_crd(2,:));        %Point 13
                max(S_crd(1,:))-racc    min(S_crd(2,:))];       %Point 14

%Matrix with the segments of rectangles
%[point start, point end, subdomain on left, subdomain on right]
seg_rept = [    1   2   0   1;      %Segment 1
                2   3   0   1;      %Segment 2
                3   4   0   1;      %Segment 3
                4   5   0   1;      %Segment 4
                5   6   0   3;      %Segment 5
                6   7   0   2;      %Segment 6
                7   1   0   1;      %Segment 7
                5   8   3   1;      %Segment 8
                8   9   3   1;      %Segment 9
                9   10  3   1;      %Segment 10
                10  11  3   1;      %Segment 11
                11  12  3   2;      %Segment 12
                14  6   3   2;      %Segment 13
                7   13  1   2];     %Segment 14

%g contrain the information of all the segments
%if the segment is a line, the column for that segment is:
%row1   2
%row2   x coordinate of the first point
%row3   x coordinate of the second point
%row4   y coordinate of the first point
%row5   y coordinate of the second point
%row6   subdomain on the left
%row7   subdomain on the right
g = zeros(12,num_seg);
for k=1:size(seg_rept,1)
    g(1,k) = 2;
    g(2,k) = Points_rept(seg_rept(k,1),1);
    g(3,k) = Points_rept(seg_rept(k,2),1);
    g(4,k) = Points_rept(seg_rept(k,1),2);
    g(5,k) = Points_rept(seg_rept(k,2),2);
    g(6,k) = seg_rept(k,3);
    g(7,k) = seg_rept(k,4);
end

%if the segment is an arc, the column for that segment is:
%row1   1
%row2   x coordinate of the first point
%row3   x coordinate of the second point
%row4   y coordinate of the first point
%row5   y coordinate of the second point
%row6   subdomain on the left
%row7   subdomain on the right
%row8   x coordinate of the center
%row9   y coordinate of the center
%row10  radius of the arc

i_sub = max( [ seg_rept(:,3)', seg_rept(:,4)' ] );
n_rect = i_sub;
i_seg = size(seg_rept,1);

%Write rounded segment 15
i_seg = i_seg +1;
p1 = 11;
p2 = 13;
g(1,i_seg) = 1;
g(2,i_seg) = Points_rept(p1,1);
g(3,i_seg) = Points_rept(p2,1);
g(4,i_seg) = Points_rept(p1,2);
g(5,i_seg) = Points_rept(p2,2);
g(6,i_seg) = 2;
g(7,i_seg) = 1;
g(8,i_seg) = Points_rept(p2,1);
g(9,i_seg) = Points_rept(p1,2);
g(10,i_seg) = racc;

% %Write rounded segment 16
i_seg = i_seg +1;
p1 = 14;
p2 = 12;
g(1,i_seg) = 1;
g(2,i_seg) = Points_rept(p1,1);
g(3,i_seg) = Points_rept(p2,1);
g(4,i_seg) = Points_rept(p1,2);
g(5,i_seg) = Points_rept(p2,2);
g(6,i_seg) = 2;
g(7,i_seg) = 3;
g(8,i_seg) = Points_rept(p1,1);
g(9,i_seg) = Points_rept(p2,2);
g(10,i_seg) = racc;

%write the turns (external circle)
for k=1:nt
    i_sub = i_sub + 1;
    xc = Tx_crd(1,k);
    yc = Ty_crd(1,k);
    rc = dc/2;
    teta = 0;
    for j=1:4
        i_seg = i_seg +1;
        g(1,i_seg) = 1;
        g(2,i_seg) = xc+rc*cos(teta);
        g(3,i_seg) = xc+rc*cos(teta+pi/2);
        g(4,i_seg) = yc+rc*sin(teta);
        g(5,i_seg) = yc+rc*sin(teta+pi/2);
        teta = teta + pi/2;
        g(6,i_seg) = i_sub;
        g(7,i_seg) = 1;
        g(8,i_seg) = xc;
        g(9,i_seg) = yc;
        g(10,i_seg) = rc;
    end
end

%write water (internal circle)
for k=1:nt
    i_sub = i_sub + 1;
    xc = Tx_crd(1,k);
    yc = Ty_crd(1,k);
    rc = dc1/2;
    teta = 0;
    for j=1:4
        i_seg = i_seg +1;
        g(1,i_seg) = 1;
        g(2,i_seg) = xc+rc*cos(teta);
        g(3,i_seg) = xc+rc*cos(teta+pi/2);
        g(4,i_seg) = yc+rc*sin(teta);
        g(5,i_seg) = yc+rc*sin(teta+pi/2);
        teta = teta + pi/2;
        g(6,i_seg) = i_sub;
        g(7,i_seg) = i_sub-nt;
        g(8,i_seg) = xc;
        g(9,i_seg) = yc;
        g(10,i_seg) = rc;
    end
end

nsd = i_sub;%Number of subdomains
num_seg = size(g,2);%Number of segments

%plot the geometry to see if it is right
if fig_yn
    figure(100)
    pdegplot(g,'subdomainLabels','on')
    axis equal
end
%% CREATION OF b %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%"b" is a matrix that describes the boundary conditions
%First I create a "Bound" matrix
```

```
%Column1   segment
%Column2   1=dirichlet (h*u=r)   0=neumann (n*c*grad(u)+q*u=g)
%Column3   h (o q)
%Column4   r (o g)

Bound = {   '1',            '1',            '1',            '0';
            '2',            '1',            '1',            '0';
            '3',            '1',            '1',            '0';
            '4',            '1',            '1',            '0';
            '5',            '1',            '1',            '0';
            '6',            '1',            '1',            '0';
            '7',            '1',            '1',            '0'     };

%the "b" matrix is made of ASCII characters
%For example, if the boundary condition is Neumann, the column is:
%row1   1
%row2   0
%row3   Number of digit in the q coefficient
%row4   Number of digit in the g coefficient
%row5 to row(5+row3) contains the ascii number corresponding to the numbers of q
%row(5+row3+1) to row(5+row3+1+row4) is like the above row but for g
b = zeros(10,num_seg);
intern_seg = [  0   1   1   1   1   1   48  48  49  48]';
for k=1:num_seg
    b(:,k) = intern_seg;
end
%Routine for the creation of "b"
for k=1:size(Bound,1)
    if isnumeric(str2double(Bound{k,1}))
        seg = str2double(Bound{k,1});
        b(1,seg) = 1;
        if strcmp(Bound{k,2},'1')    % Dirichlet
            h_coeff = Bound{k,3};
            r_coeff = Bound{k,4};
            b(5,seg) = length(h_coeff);
            b(6,seg) = length(r_coeff);
            pos = 9;
            for j=1:length(h_coeff)
                b(pos,seg) = invchar(h_coeff(1,j));
                pos = pos+1;
            end
            for j=1:length(r_coeff)
                b(pos,seg) = invchar(r_coeff(1,j));
                pos = pos+1;
            end
        elseif strcmp(Bound{k,2},'0')    % Neumann
            b(2,seg) = 0;
            q_coeff = Bound{k,3};
            g_coeff = Bound{k,4};
            b(3,seg) = length(q_coeff);
            b(4,seg) = length(g_coeff);
            pos = 5;
            for j=1:length(q_coeff)
                b(pos,seg) = invchar(q_coeff(1,j));
                pos = pos+1;
            end
            for j=1:length(g_coeff)
                b(pos,seg) = invchar(g_coeff(1,j));
                pos = pos+1;
            end
        else
            warndlg('Second column of matrix Bound must be 0 or 1', ['Error in
',mfilename,'.m.']);
            return
        end
    else
        warndlg('First column of matrix Bound must be a number', ['Error in
',mfilename,'.m.']);
        return
    end

end
%% CREATION OF "Materials" MATRIX %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%subdomains
%subdomain 1 = Air
%subdomain 2 = Silicon
%subdomain 3 = Graphite
%subdomain 4-... = TurnS
%nsd = number of subdomains (air, 2*graphite, silicon, nt turns)
material_subd = zeros(1,nsd);
%subdomain          1       2       3       4       5       6       7       8
...
%subdomain example:
%subdomain = [      1,      2,      3,      4,      4,      4,      4,      4,
...
%the meaning of the numbers is:
%           1 per Air
%           2 per Silicon
%           3 per Graphite
%           4 per Copper
%           5 per Water

material_subd(1,1:3) = [1, 2, 3];
for i=4:(nt+3)
    material_subd(1,i) = 4;
end
for i=(nt+4):(nsd)
    material_subd(1,i) = 5;
end


%The matrix Materials.mat is formed like that:
%Material                                        1        2        3        4
...
%Mu                                              mu1      mu2      mu3      mu4
...
%Sigma                                           sigma1   sigma2   sigma3   sigma4
...
%Density                                         dens1    dens2    dens3    dens4
...
```

```
%Thermal conductivity                           k1       k2       k3       k4
...
%Specific Heat Capacity                         c1       c2       c3       c4
...
%Coefficient of temperature for conductivity    alpha1   alpha2   alpha3   alpha4
...
%Temperature at which the sigma is given        Ta1      Ta2      Ta3      Ta4
...
%Emissivity                                     epsilon1 epsilon2 epsilon3 epsilon4
...
Materials = [material_subd;zeros(5,nsd)];
for i=1:nsd
    switch Materials(1,i)
        case 1  %Air
            Materials(2:9,i) =
[muA;sigmaA;densityA;ThCoA;SpHeCaA;alphaA;tsA;epsilonA];
        case 2  %Silicon
            Materials(2:9,i) =
[muS;sigmaS;densityS;ThCoS;SpHeCaS;alphaS;tsS;epsilonS];
        case 3  %Graphite
            Materials(2:9,i) =
[muG;sigmaG;densityG;ThCoG;SpHeCaG;alphaG;tsG;epsilonG];
        case 4  %Copper
            Materials(2:9,i) =
[muC;sigmaC;densityC;ThCoC;SpHeCaC;alphaC;tsC;epsilonC];
        case 5  %Water
            Materials(2:9,i) =
[muW;sigmaW;densityW;ThCoW;SpHeCaW;alphaW;tsW;epsilonW];
    end
end
%% Turns.mat %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The matrix Turns.mat is formed like that:
%Subdomain          S1      S2      S3      S4      S5      S6      ...
%Current            I1      I2      I3      I4      I5      I6      ...
%Section            S1      S2      S3      S4      S5      S6      ...
%Radius of turn     R1      R2      R3      R4      R5      R6      ...
%frequency          f       f       f       f       f       f       ...
%diameter of cond   dc1     dc2     dc3     dc4     dc5     dc6     ...
%Diameter of hole   dh1     dh2     dh3     dh4     dh5     dh6     ...
Turns = zeros(5,nt);

for k=1:(nt)
    Turns(1:7,k) = [k+n_rect, IC, SC, Tx_crd(1,k), freq, dc, dc1];
end
end
function charcode = invchar(c)
    charcode = find(char(0:255) == c) - 1;
end
```

Function “ComputeI”

```
function [ I, V ] = ComputeI( u )
%Given the solution, this function compute the current in the turns

%Here are taken the matrices from the calle function
p = evalin('caller','p');
t = evalin('caller','t');
Materials = evalin('caller','Materials');
Turns = evalin('caller','Turns');
[A,~,~,~] = pdetrg(p,t);
%% COMPUTATION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
turns_index = Turns(1,:);
omega = 2*pi*Turns(5,1);
PSIfi = u;
Afi_p = PSIfi./(p(1,:))';%On the axis there are divisions by zero, so the
Afi_p(isnan(Afi_p)) = 0;%nans are replaced with zero
Afi_t = pdeintrp(p,t,Afi_p);
%Initialization of the current and voltage vectors to zero
I = zeros(1,size(turns_index,2));
V = I;
%Iteration through the turns
for k=1:size(turns_index,2);%k is the subdomain number
    i_t = pdesdt(t,turns_index(1,k));%index of triangles in subdomain k
    for h=1:size(i_t,2)
        J = -1i*omega*Materials(3,t(4,i_t(1,h)))*Afi_t(1,i_t(1,h));
        area = A(1,i_t(1,h));
        I(1,k) = I(1,k) + area*J;
    end
    V(1,k) = 2*pi*Turns(4,k)*I(1,k)/(Materials(3,turns_index(1,k))*Turns(3,k));
end
end
```

Function “u_Integral1”

```
function [ Area_integral, Volume_integral ] = u_Integral1( varargin )
%This function integrates the solution on the area and on the volume. If no
%indices of triangles are given, the function integrates in all the domain
%[ Area_integral, Volume_integral ] = u_Integral( p, t, u )
%[ Area_integral, Volume_integral ] = u_Integral( p, t, u, i_t )

if nargin==3
    p = varargin{1};
    t = varargin{2};
    u = varargin{3};
    i_t = (1:1:size(t,2));
elseif nargin==4
    i_t = varargin{4};
    p = varargin{1};
    t = varargin{2}(:,i_t);
    u = varargin{3}(:,i_t);
else
    error('Wrong number of variables in input.\n');
end
xpts=(p(1,t(1,:))+p(1,t(2,:))+p(1,t(3,:)))/3;
[A,~,~,~] = pdetrg(p,t);
temp = A.*u;
Area_integral = sum(temp);
temp = temp.*2.*pi.*xpts;
Volume_integral = sum(temp);

end
```

Function “pdetool2tecplot”

```
function [] = pdetool2tecplot (name_out, p, e, t, varargin)
%% PDETOOL2TECPLOT (name_out, p, e, t, V1, V2, ..., Options)
%Given an output of the pdetool ( p, e, t, and variable), this function
%creates a file (txt or binary) readable by tecplot.
%The min function call is pdetool2tecplot (name_out, p, e, t), where:
%name_out is a string rappresenting the name of the output file
%p, e and t have the same meaning as in the pdetool
%Optionally it is possible to add Variables and Options
%pdetool(name_out, p, e, t, V, Options) where:
%      V are variables. The variables must have the same the same number of
% elements as p or t. It is possible to add as many variables as you
% want. If a variable is a complex number, the variable will be splitted
% in real and imaginary part.
%      Options can be one of follow elements
%     'Precision': Must be used if you want to specify the number of
%     digits saved after the point. It must be a positive integer number.
%     For default 6 digits are saved after the point. The max in a double
%     precision number are 15, so a number greater than that will be set
%     to 15. For example you can format the numbers like this 8.95672e6
%     (5 digits after the point) as follow:
%         pdetool2tecplot2('prova', p, e, t, 'Precision', 5)
%     'Subdomain': It is possible to not copy all the subdomains in the
%     file. The argument of this option must be a number or a vector of
%     numbers. If not specified, all the domains will be included.
%     For example if you want to save only the subdomain 2 and 7 you can
%     do that as follow:
%         pdetool2tecplot2 ('prova', p, e, t, 'Subdomain', [2, 7])
%     'Division': It is possible to divide the subdomains, so TecPlot can
%     manage them separately. The argument must be 'Yes'(Default) or 'No'.
%         pdetool2tecplot2 ('prova', p, e, t,...
%                    'Division', 'No')
%     'VarName': It is possible to specify the names of the variables.
%     The input must be a cell array of strings and the number of strings
%     must be equal to the number of variables (or the number of
%     variables +2 if you want to include the names of x and y axes).
%     In the following example the names of the axes are included.
%         pdetool2tecplot2 ('prova', p, e, t, Var1, Var2,...
%                      'VarName', {'r' 'z' 'Pressure' 'Temperature'})
%     'ZonesName': It is possible to specify the names of the zones. The
%     input must be a cell array of strings and the number of strings
%     must be equal to the number of zones. If you choose to not divide
%     the subdomains there will be only a zone, otherwise there will be a
%     zone for each subdomain.
%     You can specify the names as follow:
%         pdetool2tecplot2 ('prova', p, e, t, Var1, Var2,...
%                     'Subdomain', [1,2,3],...
%                     'ZonesName',{'Air' 'Silicon' 'Graphite'});
%     'ZAD' means "Zone Auxiliary Data". It is possible to specify
%     auxiliary data in order to use it in tecplot. You need to specify
%     the name of the variable and the value of the variable for each
%     zone.
%     The call must be
%         'ZAD', Names, Values,
%     where:
%     "Names" is a cell array and have dimensions nv by 1 where nv is the
%     number of variables.
%     "Values" is a cell array and have dimensions nv by nz where nz is
%     the number of zones.
%     For example I can save the total flux from external boundary and
%     the total flux from internal boundary for the 2 zone as follow
%       pdetool2tecplot2 ('prova', p, e, t, Temp,...
%         'ZAD',{'ext_flux';'int_flux'},{flu_ext1, flu_ext2; flu_int1, flu_int2 },...
%              'ZonesName',{'Cond1' 'Cond2'});
%     'DSAD' means "DataSet Auxiliary Data". It is similar to ZAD,
%     with the difference that this data is common for all the dataset,
%     so you don't need data for each Zone. The syntax is symilar to the
%     previous, but this time nz = 1.
%     'Title': To specify the title of the plot. The argument must be a
%     string
%     You can set the title as follow
%         pdetool2tecplot2 ('prova', p, e, t,...
%                     'Title', 'round plate')
%     'Time': It is possible to specify at whitch the the simulation is
%     taken, so is simple to manage them with tecplot (for example to
%     create an animation). The time must be expressed in seconds.
%     You can specify the time as follow
%         pdetool2tecplot2 ('prova', p, e, t, Var1,...
%                     'Time',0.0001)
%     'TypeOut': It is possible to save the output file as a binary or a
%     txt file. The advantage af a binary file is to save space on the HD
%     (usually a half). The argument of this option can be
%         'ASCII' to save the txt file
%         'binary' (Default) to save the binary file
%         'both' to save both the txt and binary file
%     Writing the binary file is possible only if Tecplot is installed in
%     the computer.
%     Example
%         pdetool2tecplot2 ('prova', p, e, t, Var1,...
%                     'TypeOut','both')
%     'Grid': is possible to interpolate the value of the FE mesh into a
%     mapped mesh. After "Grid" you have to specify if the grid start
%     from the x or y values ('xy' or 'yx'). Then you have to specify the
%     two vectors of coordinates. The vectors are taken from 0 to 1. The
%     function will expand the points on all the domain. For example you
%     can specify as follow
%         pdetool2tecplot ('Example_grid', p, e, t, u,...
%                     'grid','xy',x,y,...
%                     'subdomain',2,...
%                     'title', 'Sphere');
%% ERRORS AND CONTROL OF INPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%generate error if the name of the file is not a string
if ~ischar(name_out)
    error('The name of the file must be a string.')
end
%remove blank spaces in the name
name_out = sscanf(name_out,'%s');
%remove the extension (if present)
if size(name_out,2)>=5
    if (strcmpi(name_out((end-3):end), '.plt') )
        name_out(end-3:end) = [];
    end
end
%create the name for ASCII and binary file
name_out_ascii = [name_out,'_ASCII.plt'];
name_out_bin = [name_out,'_bin.plt'];

nt = size(t,2);%number of triangles
np = size(p,2);%number of points
sd = unique(t(4,:));%subdomains
xpts=(p(1,t(1,:))+p(1,t(2,:))+p(1,t(3,:)))/3;
ypts=(p(2,t(1,:))+p(2,t(2,:))+p(2,t(3,:)))/3;
nin_varargin = length(varargin);%number of input variables
%% CONTROL OF THE INPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%generate error if the number of input are not enough
narginchk(4, inf)
if nin_varargin > 0
    k = 1;
    flagvals = {'Precision' 'Subdomain' 'Division' 'VarName' 'Title' 'Grid' '-Space'
'TypeOut' 'Time' 'ZonesName' 'ZAD' 'DSAD'};
    nflagvals = size(flagvals,2);
    flag_value = cell(1,nflagvals);
    i = 1;

    %DEFAUL VALUES AND FLAGS
    %Precision
    Precision_value = 6;%default value for precision
    %SubDomain
    SubDomains = sd;%default
    %Division
    Division_flag = {'yes' 'no'};
    Division = 1;%default value
    %Title
    Title = 'DATA';%Default
    %Grid
    grid_flag = {'xy' 'yx'};%Possible flag values for grid
    %-Space
    min_space = false;%default value for flag "-Space"
    %TypeOut
    TypeOut_flag = {'ASCII' 'binary' 'both'}; %Possible flags values for TypeOut
    TypeOut = 2;%default value
    %Time
    Time = nan;
    %ZonesName
    ZonesName = {};

    Variables = {};
    Var_Name = {'x' 'y'};
    nod_cell = {'n' 'n'};
    while i<=nin_varargin
        if ischar( varargin{i} )
            %deblank flag
            flag = sscanf(varargin{i},'%s');
            foundflag = strcmpi(flag,flagvals);
            % If the flag not existent -> error
            if ~any(foundflag)
                error(['Unknown Flag "',flag,'".']);
            end
            %if a flag is found more than 1 time -> error
            if sum(strcmpi(flag_value,varargin{i}))~=0
                error(['Flag "',flag,'" repeated.']);
            end
            %the next value must be assigned to the flag
            i = i+1;
            if i > nin_varargin
                error(['Too few arguments for "',varargin{i-1},'".']);
            end
            %Check if the argument is ok
            switch find(foundflag==1)
                case 1%Precision
                    try
                        if ~isreal(varargin{i})||~(varargin{i}>0)
                            error('Invalid precision.');
                        end
                    catch
                        error('Invalid precision.');
                    end
                    flag_value{foundflag} = varargin{i};
                    if varargin{i} > 15
                        Precision_value = 15;
                    else
                        Precision_value = varargin{i};
                    end
                case 2%Subdomain
                    try
                        if size(intersect(varargin{i},sd),2)~=size(varargin{i},2)
                            error('Invalid subdomain.');
                        end
                    catch
                        error('Invalid subdomain.');
                    end
                    SubDomains = sort(varargin{i});
                case 3%Division
                    Choice_Division = find(strcmpi(varargin{i},Division_flag)==1);
                    if isempty(Choice_Division)
                        error('Invalid division value.');
                    else
                        flag_value{foundflag} = varargin{i};
                        Division = Choice_Division;
                    end
                case 4%Var_Name
                    if ~iscell(varargin{i})
                        error('Var_Name value is not a cell.');
                    end
                    for k=1:size(varargin{i},2)
                        if ~ischar(varargin{i}{k})
                            error('One of the Var_Name values is not a string.');
                        end
                    end
                    flag_value{foundflag} = varargin{i};
                case 5%Title
                    if ~ischar(varargin{i})
                        error('Title argument is not a string.');
                    end
                    Title = varargin{i};
                case 6%Grid
                    flag_value{foundflag} = varargin{i-1};
                    %Grid must have 4 arguments in total
                    if i+2 > nin_varargin
                        error('Too few arguments for "Grid".');
                    end
                    if ischar(varargin{i}) %
                        grid_flag_value = varargin{i};
                        grid_mod = find(strcmpi(grid_flag_value,grid_flag)==1);
                        %control if the flag exist
                        if isempty(grid_mod)
                            error(['The value (',grid_flag_value,') specified for
"Grid" is not a valid value']);
                        end
                        i = i+1;
                        Grid{1} = varargin{i};
                        i = i+1;
                        Grid{2} = varargin{i};
                        %control if are numbers
                        if ~( isnumeric(Grid{1}) && isnumeric(Grid{2}) )
                            error('"Grid" arguments are not numbers.');
                        end
                        %control if are two points
                        if all(size(Grid{1})==1) && all(size(Grid{2})==1)
                            error('The arguments for "Grid" are two points.');
                        end
                        %control if are row vectors
                        if ~( size(Grid{1},1) && size(Grid{2},1) )
                            error(['For ',grid_flag_value,' you must specify two row
vectors']);
```

```
                        end
                        %are sorted?
                        if ~issorted(Grid{1})||~issorted(Grid{2})
                            Grid{1} = sort(Grid{1});
                            Grid{2} = sort(Grid{2});
                            warning('The vectors were not sorted.');
                        end
                        if any( Grid{1}>1 ) || any( Grid{1}<0 ) || any( Grid{2}>1 )
|| any( Grid{2}<0 )
                            error(['For ',grid_flag_value,' the values cannot be >1
or <0.']);
                        end
                    else
                        error('For "Grid" you need to specify also "xy" or "yx".');
                    end
                case 7%Space
                    min_space = true;
                    i = i - 1;
                case 8%binary
                    choice_TypeOut = find(strcmpi(varargin{i},TypeOut_flag)==1);
                    if isempty(choice_TypeOut)
                        error('Invalid "choice_TypeOut" value.');
                    else
                        TypeOut = choice_TypeOut;
                    end
                case 9%Time
                    if isequal(size(varargin{i}),[1,1]) && isnumeric(varargin{i})
                        Time = varargin{i};
                    else
                        error('Invalid "Time" value.');
                    end
                case 10%ZonesName
                    if ~iscell(varargin{i})
                        error('"ZonesName" value is not a cell.');
                    end
                    for k=1:size(varargin{i},2)
                        if ~ischar(varargin{i}{k})
                            error('One of the "ZonesName" values is not a string.');
                        end
                    end
                    ZonesName = varargin{i};
                case 11%ZAD ex: 'ZAD', 'Name', [Value/Values]
                    ZAD.name = varargin{i};
                    i = i + 1;
                    ZAD.values = varargin{i};
                case 12%DSAD ex: 'DSAD', 'Name', [Value/Values]
                    DSAD.name = varargin{i};
                    i = i + 1;
                    DSAD.values = varargin{i};
            end
        else
            %generate error if the values are matrix
            if all(size( varargin{i} ) > 1 )
                error(['The variable "',inputname(i+4),'" is a matrix.'])
            end
            %Creation of a vector with the name of variables.
            %If the name can't be readed, give the number of input
            if strcmp(inputname(i+4),'')
                sd_name{k} = num2str(i+4); %#ok<AGROW>
            else
                sd_name{k} = inputname(i+4); %#ok<AGROW>
            end
            %generate error if the values varargin have not the same dimensions of p
or t
            if ~any(size( varargin{i} ) == np ) && ~any(size( varargin{i} ) == nt )
                error(['The length of the input variable "',sd_name{k},'" is
different from the length of p or t.'])
            end
            if isreal(varargin{i})
                %creation of a vector that tell me if the variable in on nodes or
cell centered
                if any( size( varargin{i} ) == nt )
                    nod_cell{k+2} = 'c';
                else
                    nod_cell{k+2} = 'n';
                end

                %All the variables are row vectors
                if size( varargin{i}, 1 ) == 1
                    Variables{k} = varargin{i}; %#ok<AGROW>
                else
                    Variables{k} = varargin{i}.'; %#ok<AGROW>
                end

                %Build the Var_Name variable. If the names are specified, the
                %value will be overwritten
                if strcmp(inputname(i+4),'')
                    Var_Name{k+2} = num2str(i+4);
                else
                    Var_Name{k+2} = inputname(i+4);
                end
                k = k + 1;
            else %if not a real number, split in real and imaginary
                %creation of a vector that tell me if the variable in on nodes or
cell centered
                if any( size( varargin{i} ) == nt )
                    nod_cell{k+2} = 'c';
                    nod_cell{k+3} = 'c';
                else
                    nod_cell{k+2} = 'n';
                    nod_cell{k+3} = 'n';
                end
                %All the variables are row vectors
                if size( varargin{i}, 1 ) == 1
                    Variables{k} = real(varargin{i}); %#ok<AGROW>
                    Variables{k+1} = imag(varargin{i}); %#ok<AGROW>
                else
                    Variables{k} = real(varargin{i}.'); %#ok<AGROW>
                    Variables{k+1} = imag(varargin{i}.'); %#ok<AGROW>
                end
```

```
                  %Build the Var_Name variable. If the names are specified, the
                  %value will be overwritten
                  if strcmp(inputname(i+4),'')
                      Var_Name{k+2} = [num2str(i+4),' - real'];
                      Var_Name{k+3} = [num2str(i+4),' - imag'];
                  else
                      Var_Name{k+2} = [inputname(i+4),' - real'];
                      Var_Name{k+3} = [inputname(i+4),' - imag'];
                  end
                  k = k + 2;
              end
          end
          i = i+1;
      end

  % Control of fighting flags
      %no 'Division' and 'Grid' at the same time
      if ~isempty(flag_value{3})&&~isempty(flag_value{6})
          error('"Division" and "Grid" cannot be set at the same time.');
      end
      %no '-Space' and 'Grid' at the same time
      if ~isempty(flag_value{7})&&~isempty(flag_value{6})
          error('"-Space" and "Grid" cannot be set at the same time.');
      end
      nin_Variable = length(Variables);
      % set value 'Precision'
      Precision = ['%+.',num2str(Precision_value),'e'];
      % set Var_Name
      if ~isempty(flag_value{4})
          if size(flag_value{4},2)==nin_Variable+2
              k=1; i=1;
          elseif size(flag_value{4},2)==nin_Variable
              k=3; i=1;
          else
              error('Wrong number of Var_Name Values. Complex values?');
          end
          while k <= nin_Variable + 2
              Var_Name{k} = flag_value{4}{i};
              k = k+1; i=i+1;
          end
      end
      % Set Grid
      if ~isempty(flag_value{6})
          status = 1;
          %div_flag 'yes' 'no'
          if Division==1 %if division == yes
              warning('With "Grid" is not possible to divide the domain. Division set
  to "no".');
              Division = 2;%division set to no
          end
          if grid_mod == 1
              [POINTS_X, POINTS_Y] = points_matrix_xy ( );
              if status==0
                  [POINTS_X, POINTS_Y] = points_matrix_yx ( );
                  if status==1
```

```
                    warning('The "Grid" method has been changed from "xy" to "yx".');
                else
                    error('The domain has holes or irregularities.');
                end
            end
        else
            [POINTS_X, POINTS_Y] = points_matrix_yx ( );
        end
    end
    %Number of Zones
    if Division == 2%div_flag 'yes' 'no'
        n_Zones = 1;
    else
        n_Zones = size(SubDomains,2);
    end
    % set ZonesName
    if size(ZonesName,2)>0
        if size(ZonesName,2)~=n_Zones
            error('Wrong number of "ZonesName" Values.');
        end
    else
        ZonesName = cell(1,n_Zones);
        if n_Zones==1
            ZonesName{1} = '';
            for k=1:size(SubDomains,2)
                ZonesName{1} = strcat(ZonesName{1},num2str(SubDomains(k)),',');
            end
            %remove the last ,
            ZonesName{1}(end) = [];
        else
            for k=1:n_Zones
                ZonesName{k} = num2str(SubDomains(k));
            end
        end
    end
    %Control of Zone AuxData
    if exist('ZAD','var')
        if size(ZAD.name,1)~=size(ZAD.values,1)
            error('Number of Names for "ZAD" different from number of Values');
        end
        if size(ZAD.values,2)~=n_Zones
            error('Number of Values for "ZAD" different from number of Zones');
        end
    end
    %Control of Dataset AuxData
    if exist('DSAD','var')
        if size(DSAD.name,1)~=size(DSAD.values,1)
            error('Number of Names for "DSAD" different from number of Values');
        end
        if size(DSAD.values,2)~=1
            error('Number of Values for "DSAD" different from number of Zones');
        end
    end
end
%% OPEN FILE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%free some space
clear varargin
%Open the file
[fileID, message]= fopen(name_out_ascii,'w');
%generate error if the file is not opened
if fileID < 0
    error(['Fail to open the file to write. ',message]);
end
%% WRITE FILE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%write the format. New line after 10 numbers
number = [Precision,'\t'];
format = repmat(number,1,10);
format = [format,'\n'];
fprintf (fileID,'TITLE= "%s"\n', Title);
%Initialize some char variables
str_time = '';
str_name_variables = '';
str_nodal = '';
str_cell = '';
write_node_order = false;
write_strings()
fprintf (fileID, '%s\n',str_name_variables);

if isempty(flag_value{6})
%% WRITE FILE - FE TRIANGLE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    str_zone = 'ZONETYPE = FETRIANGLE, DATAPACKING = BLOCK';
    %div_flag 'yes' 'no'
    if Division==2
        if ~isequal(SubDomains,sd)
            write_zone_red (1, SubDomains );
        else
            write_zone ( 1 );
        end
    else
        for k=1:size(SubDomains,2)
            if k==1
                fprintf('Writing subdomain: ');
            else
                for h=1:l_z+1
                    fprintf('\b');
                end
            end
            fprintf('%s\n',ZonesName{k});
            write_zone_red (k, SubDomains(k) );
            l_z = length(ZonesName{k});
        end
    end
else
%% WRITE FILE - ORDERED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    str_zone = 'ZONETYPE = ORDERED, DATAPACKING = BLOCK';
    write_zone_ordered ( 1 )
end
%before closing the file, the dataset aux data must be written
write_DSAD;
fclose(fileID);
```

```
%% OUTPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%If the binary is needed and tecplot is installed, create the file
if TypeOut==2 || TypeOut==3
    command = ['preplot ',name_out_ascii,' ',name_out_bin];
    status_bin = dos(command);
end
switch TypeOut
    case 1%ASCII
        fprintf('ASCII file "%s" created.\n',name_out_ascii);
    case 2%binary
        if status_bin~=-1
            delete(name_out_ascii);
            fprintf('Binary file "%s" created.\n',name_out_bin);
        else
            warning('Binary file not created. (Maybe you do not have TecPlot
installed). Created the ASCII file"%s" instead',name_out_ascii);
%              warning('progr:Nneg','Input N=%d must be positive\n',N);
        end
    case 3%both
        if status_bin~=-1
            fprintf('ASCII file "%s" created.\n',name_out_ascii);
            fprintf('Binary file "%s" created.\n',name_out_bin);
        else
            fprintf('ASCII file "%s" created.\n',name_out_ascii);
            warning('Binary file not created. (Maybe you do not have TecPlot
installed)');
        end
end
%% NESTED FUNCTIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function write_ZAD (Zone_Number)
        if exist('ZAD','var')
            for kk=1:size(ZAD.name,1)
                name = ZAD.name{kk,1};
                Value = ZAD.values{kk,Zone_Number};
                if isnumeric(name)
                    name = num2str(name);
                end
                if isnumeric(Value)
                    Value = num2str(Value);
                end
                str_ZAD = ['AUXDATA ',name,' = "',Value,'"'];
                fprintf (fileID,'%s\n',str_ZAD);
            end
        end
    end
    function write_DSAD
        if exist('DSAD','var')
            for kk=1:size(DSAD.name,1)
                name = DSAD.name{kk,1};
                Value = DSAD.values{kk,1};
                if isnumeric(name)
                    name = num2str(name);
                end
                if isnumeric(Value)
                    Value = num2str(Value);
                end
                str_DSAD = ['DATASETAUXDATA ',name,' = "',Value,'"'];
                fprintf (fileID,'%s\n',str_DSAD);
            end
        end
    end
%% WRITE STRINGS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function write_strings()
        if ~isnan(Time)
            str_time = sprintf('SOLUTIONTIME = %s',num2str(Time));
        end
        %Condition when to write node order
        if min_space == false
            if ~isequal(SubDomains,sd)
                write_node_order = true;
            elseif Division==1%div_flag 'yes' 'no'
                write_node_order = true;
            end
        end
        %Write str_name_variables
        str_name_variables = 'VARIABLES =';
        for kk=1:size(Var_Name,2)
            str_name_variables = strcat(str_name_variables,sprintf ('
"%s"',Var_Name{kk}));
        end
        if write_node_order==true && isempty(flag_value{6})
            str_name_variables = strcat(str_name_variables,sprintf (' "%s"','Node-
Order'));
        end
        % write str_nodal
        nodal = find(strcmpi(nod_cell,'n')==1);
        str_nodal = 'VARLOCATION = ([';
        for kk=1:size(nodal,2)
            str_nodal = strcat(str_nodal,num2str(nodal(kk)));
            if kk==2 && write_node_order==true && isempty(flag_value{6})
                str_nodal = strcat(str_nodal,',3');
            end
            if kk~=size(nodal,2)
                str_nodal = strcat(str_nodal,', ');
            else
                if write_node_order==true && isempty(flag_value{6})
                    str_nodal = strcat(str_nodal,',',num2str(nin_Variable+3),']=
NODAL)');
                else
                    str_nodal = strcat(str_nodal,']= NODAL)');
                end
            end
        end
        % write str_cell
        cellcentered = find(strcmpi(nod_cell,'c')==1);
        if any(cellcentered>0)
            str_cell = 'VARLOCATION = ([';
            for kk=1:size(cellcentered,2)
                str_cell = strcat(str_cell,num2str(cellcentered(kk)));
                if kk~=size(cellcentered,2)
```

```
                    str_cell = strcat(str_cell,',');
                end
            end
            str_cell = strcat(str_cell,']= CELLCENTERED)');
        end
    end
%% WRITE ZONE REDUCED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function write_zone_red (Zone, SubDomains )
        zone_name = ZonesName{Zone};
        [int, cont] = pdesdp(p,e,t,SubDomains);
        i_p = [int cont];
        i_t = pdesdt(t,SubDomains);
        p_red = p(:,i_p);
        np_red = size(p_red,2);
        nt_red = size( t(:,i_t),2 );
        fprintf (fileID, '\n\n#ZONE %s\n',zone_name);
        fprintf (fileID,'ZONE T ="%s", N = %d, E = %d', zone_name, np_red, nt_red);
        fprintf (fileID,'%s\n',str_time);
        write_ZAD(Zone);
        if write_node_order==true
            fprintf (fileID, sprintf(', NV = %s\n',num2str(nin_Variable+3)));
        else
            fprintf (fileID,'\n');
        end
        fprintf (fileID,'%s\n',str_nodal);
        if ~strcmp(str_cell,'')
            fprintf (fileID,'%s\n',str_cell);
        end
        fprintf (fileID,'%s\n',str_zone);
        %write x points
        fprintf (fileID, '\n\n#ZONE %s - VARIABLE %s\n',zone_name,Var_Name{1});
        fprintf (fileID, format, p(1,i_p));
        fprintf (fileID, '\n');
        %write y points
        fprintf (fileID, '\n\n#ZONE %s - VARIABLE %s\n',zone_name,Var_Name{2});
        fprintf (fileID, format, p(2,i_p));
        fprintf (fileID, '\n');
        %write other variables
        for j=1:length(Variables)
            fprintf (fileID, '\n\n#ZONE %s - VARIABLE %s\n',zone_name,Var_Name{j+2});
            if size(Variables{j},2)==np
                fprintf (fileID, format, Variables{j}(1,i_p));
            elseif size(Variables{j},2)==nt
                fprintf (fileID, format, Variables{j}(1,i_t));
            else
                error('Error in write.');
            end
            fprintf (fileID, '\n');
        end
        %write "Node-Order"
        if write_node_order==true
            fprintf (fileID, '\n\n#ZONE %s - VARIABLE "Node-Order"\n',zone_name);
            format_NO = [repmat('%d\t',1,10),'\n'];
            fprintf (fileID, format_NO, i_p);
            fprintf (fileID, '\n');
        else
            t_red = new_t (t(1:3,i_t), i_p);
        end
        %Write the connectivity list
        fprintf (fileID, '\n\n#ZONE %s - CONNECTIVITY LIST\n',zone_name);
        if write_node_order==true
            fprintf (fileID, '%-d\t%-d\t%-d\n', t(1:3,i_t) );
        else
            fprintf (fileID, '%-d\t%-d\t%-d\n', t_red );
        end
        fprintf (fileID, '\n\n\n\n');
    end
%% WRITE ZONE ENTIRE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function write_zone (Zone )
        zone_name = ZonesName{Zone};
        fprintf (fileID, '\n\n#ZONE %s\n',zone_name);
        fprintf (fileID,'ZONE T ="%s", N = %d, E = %d \n', zone_name, np, nt);
        fprintf (fileID,'%s\n',str_time);
        write_ZAD(Zone);
        fprintf (fileID,'%s\n',str_nodal);
        if ~strcmp(str_cell,'')
            fprintf (fileID,'%s\n',str_cell);
        end
        fprintf (fileID,'%s\n',str_zone);
        %write x points
        fprintf (fileID, '\n\n#ZONE %s - VARIABLE %s\n',zone_name,Var_Name{1});
        fprintf (fileID, format, p(1,:));
        fprintf (fileID, '\n');
        %write y points
        fprintf (fileID, '\n\n#ZONE %s - VARIABLE %s\n',zone_name,Var_Name{2});
        fprintf (fileID, format, p(2,:));
        fprintf (fileID, '\n');
        %write other variables
        for j=1:length(Variables)
            fprintf (fileID, '\n\n#ZONE %s - VARIABLE %s\n',zone_name,Var_Name{j+2});
            fprintf (fileID, format, Variables{j});
            fprintf (fileID, '\n');
        end
        %Write the connectivity list
        fprintf (fileID, '\n\n#ZONE %s - CONNECTIVITY LIST\n',zone_name);
        fprintf (fileID, '%-5d\t%-5d\t%-5d\n', t(1:3,:) );
        fprintf (fileID, '\n\n\n\n');

    end
%% WRITE ZONE ORDERED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function write_zone_ordered (Zone )
        zone_name = ZonesName{Zone};
        I = size( POINTS_X,2 );
        J = size( POINTS_X,1 );
        K = 1;
        fprintf (fileID, '\n\n#ZONE %s\n',zone_name);
        fprintf (fileID,'ZONE T ="%s"\n', zone_name);
        fprintf (fileID,'%s\n',str_time);
        write_ZAD(Zone);
        fprintf (fileID,'I = %i, J = %i, K = %i \n', I, J, K);
        fprintf (fileID,'%s\n',str_zone);
        format_ord = [repmat(number, 1, J), '\n'];
        %write x points
        fprintf (fileID, '\n\n#ZONE %s - VARIABLE %s\n',zone_name,Var_Name{1});
        fprintf (fileID, format_ord, POINTS_X.');
        %write y points
        fprintf (fileID, '\n\n#ZONE %s - VARIABLE %s\n',zone_name,Var_Name{2});
        fprintf (fileID, format_ord, POINTS_Y.');
        for j=1:length(Variables)
            fprintf (fileID, '\n\n#ZONE %s - VARIABLE %s\n',zone_name,Var_Name{j+2});
            if size(Variables{j},2)==np
                uI = scatteredInterpolant(p(1,:).',p(2,:).',Variables{j}.','linear');
            else
                uI = scatteredInterpolant(xpts(:),ypts(:),Variables{j}(:),'linear');
            end
            V_ord = uI(POINTS_X, POINTS_Y);
            fprintf (fileID, format_ord, V_ord.');
        end
    end
%% CONSTRUCT MATRIX OF POINTS FOR THE GRID %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [POINTS_X, POINTS_Y] = points_matrix_yx ( )
    status = 1;
    [int, cont] = pdesdp(p,e,t,SubDomains);
    i_p = [int cont];
    i_t = pdesdt(t,SubDomains);
    p_red = p(:,i_p);
    x_min = min(p_red(1,:)); x_max = max(p_red(1,:)); Dx = x_max - x_min;
    y_min = min(p_red(2,:)); y_max = max(p_red(2,:)); Dy = y_max - y_min;
    x_uni = Grid{1};
    y_uni = Grid{2};
    nx = size(x_uni,2);
    ny = size(y_uni,2);
    %creation of y vector
    y = y_min + y_uni*Dy;
    %take all the edges. In this way I can exclude easily all internal
    %edges
    logic = ismember(e(6:7,:),SubDomains);
    i_e = find( xor(logic(1,:),logic(2,:)) );
    i_p_e = unique([e(1,i_e),e(2,i_e)]);
    ne = size(i_e,2);
    for kk=1:ny
        %find the two edges that contain that y value
        e1_found = false;
        e2_found = false;
        for j=1:ne
            p1 = e(1,i_e(1,j));
            p2 = e(2,i_e(1,j));
            if ( (p(2,p1) > y(1,kk))&&(p(2,p2) < y(1,kk)) )||( (p(2,p1) <
y(1,kk))&&(p(2,p2) > y(1,kk)) )
                if e1_found == false
                    e1 = j;
                    e1_found = true;
                elseif e2_found == false
                    e2 = j;
                    e2_found = true;
```

```
                  else
                      status = 0;
                      return
                  end
              end
          end
          if e1_found == false
              %find distance from 2 points in y
              if kk==1
                  dist = 2*abs(y(1,1)-y(1,2));
              elseif kk==ny
                  dist = 2*abs(y(1,ny-1)-y(1,ny));
              else
                  dist = abs(y(1,kk-1)-y(1,kk+1));
              end
              %introduction error
              er = dist/50;
              %find all edges in that error
              ptemp = p(:,i_p_e);
              ind = find(abs(ptemp(2,:)-y(1,kk))<er);
              if isempty(ind)
                  status = 0;
                  return
              end
              %find min e max in x of these points
              [~,Imin] = min(ptemp(1,ind));
              [~,Imax] = max(ptemp(1,ind));
              pmin = i_p_e(1,ind(1,Imin));
              pmax = i_p_e(1,ind(1,Imax));
              x1 = p(1,pmin);
              x2 = p(1,pmax);
          else
              %trovo x1
              p1 = e(1,i_e(1,e1));
              p2 = e(2,i_e(1,e1));
              x1 = p(1,p1) + (p(1,p2)-p(1,p1))*((p(2,p1)-y(1,kk)))/(p(2,p1)-p(2,p2));
              %trovo x2
              p1 = e(1,i_e(1,e2));
              p2 = e(2,i_e(1,e2));
              x2 = p(1,p1) + (p(1,p2)-p(1,p1))*((p(2,p1)-y(1,kk)))/(p(2,p1)-p(2,p2));
          end
          Dx = abs(x2 - x1);
          x = min([x1, x2]) + x_uni*Dx;
          POINTS_X(kk,:) = x;
          POINTS_Y(kk,:) = ones(1,size(x,2))*y(1,kk);
      end
  end
  function [POINTS_X, POINTS_Y] = points_matrix_xy ( )
      status = 1;
      [int, cont] = pdesdp(p,e,t,SubDomains);
      i_p = [int cont];
      i_t = pdesdt(t,SubDomains);
      p_red = p(:,i_p);
      x_min = min(p_red(1,:)); x_max = max(p_red(1,:)); Dx = x_max - x_min;
      y_min = min(p_red(2,:)); y_max = max(p_red(2,:));
```

```
    x_uni = Grid{1};
    y_uni = Grid{2};
    nx = size(x_uni,2);
    ny = size(y_uni,2);
    %creation of x vector
    x = x_min + x_uni*Dx;
    %take all the edges. In this way I can exclude easily all internal
    %edges
    logic = ismember(e(6:7,:),SubDomains);
    i_e = find( xor(logic(1,:),logic(2,:)) );
    i_p_e = unique([e(1,i_e),e(2,i_e)]);
    ne = size(i_e,2);
    for kk=1:nx
        %find the two edges that contain that y value
        e1_found = false;
        e2_found = false;
        for j=1:ne
            p1 = e(1,i_e(1,j));
            p2 = e(2,i_e(1,j));
            if ( (p(1,p1) > x(1,kk))&&(p(1,p2) < x(1,kk)) )||( (p(1,p1) <
x(1,kk))&&(p(1,p2) > x(1,kk)) )
                if e1_found == false
                    e1 = j;
                    e1_found = true;
                elseif e2_found == false
                    e2 = j;
                    e2_found = true;
                else
                    status = 0;
                    return
                end
            end
        end
        if e1_found == false
            %find distance from 2 points in y
            if kk==1
                dist = 2*abs(x(1,1)-x(1,2));
            elseif kk==nx
                dist = 2*abs(x(1,nx-1)-x(1,nx));
            else
                dist = abs(x(1,kk-1)-x(1,kk+1));
            end
            %introduction error
            er = dist/20;
            %find all edges in that error
            ptemp = p(:,i_p_e);
            ind = find(abs(ptemp(1,:)-x(1,kk))<er);
            if isempty(ind)
                status = 0;
                return
            end
            %find min e max in y of these points
            [~,Imin] = min(ptemp(2,ind));
            [~,Imax] = max(ptemp(2,ind));
            pmin = i_p_e(1,ind(1,Imin));
            pmax = i_p_e(1,ind(1,Imax));
            y1 = p(2,pmin);
            y2 = p(2,pmax);
        end
        %interpolation
        if e1_found == true
            %trovo y1
            p1 = e(1,i_e(1,e1));
            p2 = e(2,i_e(1,e1));
            y1 = p(2,p1) + (p(2,p2)-p(2,p1))*((p(1,p1)-x(1,kk)))/(p(1,p1)-p(1,p2));
            %trovo y2
            p1 = e(1,i_e(1,e2));
            p2 = e(2,i_e(1,e2));
            y2 = p(2,p1) + (p(2,p2)-p(2,p1))*((p(1,p1)-x(1,kk)))/(p(1,p1)-p(1,p2));
        end
        Dy = abs(y2 - y1);
        y = min([y1, y2]) + y_uni*Dy;
        POINTS_Y(:,kk) = y.';
        POINTS_X(:,kk) = (ones(1,size(y,2))*x(1,kk)).';
    end
    POINTS_Y = flipud(POINTS_Y);%Only to have the points in the right order
    POINTS_X = flipud(POINTS_X);
end
end
%% OTHER FUNCTIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function temp_t1 = new_t (temp_t, i_p)
    nt_k = size(temp_t,2);
    np_k = size(i_p,2);
    np1_max = max( max( temp_t(1:3,:) ) );
    temp_t1 = zeros(3, nt_k);
    i_p_ind = zeros(1,np1_max);
    k = 0;
    for h=1:3
        for j=1:nt_k
            if i_p_ind(1,temp_t(h,j))==0 && k<=np_k
                i_p_ind(1,temp_t(h,j)) = find(i_p==temp_t(h,j),1,'first');
                k = k+1;
            end
            temp_t1(h,j) = i_p_ind(1,temp_t(h,j));
        end
    end
end
```