**Università degli Studi di Padova**

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

Corso di laurea magistrale in Ingegneria dell'Energia Elettrica

# Numerical methods for nonlinear circuit and field problems in periodic time domain

Relatore: Riccardo Torchio
Correlatore: Piergiorgio Alotto

Autore: Alberto Teatini

Anno Accademico 2021/2022

# Sommario

In questo lavoro di tesi vengono presentate, sviluppate ed applicate alcune tecniche numeriche per lo studio di problemi time-domain non lineari. L'obbiettivo principale di questo lavoro è l'implementazione di un codice che risolva il problema elettromagnetico di una macchina elettrica rotante, in questo caso un motore sincrono. Per arrivare a studiare un problema così complesso, sono stati studiati diversi casi. Inizialmente dispositivi statici non lineari, successivamente a geometrie più complicate è stata aggiunta la rotazione.

Viene descritto il metodo dell' **Harmonic Balance**, una tecnica matematica per risolvere equazioni differenziali non lineari nel dominio del tempo basata sull'analisi di Fourier. Tale metodo viene discusso e applicato in questa tesi per due differenti casi: un problema circuitale e un problema campistico. Il metodo degli Elementi Finiti con Harmonic Balance (**HBFEM**) viene implementato per casi di studio statici, e ne è stata studiata l'estensione ad oggetti rotanti.

I metodi sono stati implementati in MATLAB e i risultati sono confrontati con i risultati ottenuti in COMSOL per attestare la validità e l'accuratezza dei codici. In appendice sono riportati, quasi interamente, i codici MATLAB frutto di questo lavoro.

# Abstract

In this thesis work some numerical techniques for the study of nonlinear time-domain problems are presented, developed and applied. The main objective of this work is the implementation of a code that solves the electromagnetic problem of a rotating electric machine, in this case a synchronous motor. To get to study such a complex issue, several cases are examined. Initially, static nonlinear devices are considered, subsequently rotation is added.

The **Harmonic Balance** method, a mathematical technique for solving nonlinear differential equations in the time domain based on Fourier analysis, is described. In this thesis such method is discussed and applied to two different cases: a circuit problem and a field problem. The Finite Element Method with Harmonic Balance (**HBFEM**) is implemented for static case studies, and its extension to rotating objects is studied.

The methods are implemented in MATLAB and the results are compared with the results obtained in COMSOL to certify the validity and accuracy of the codes. In the appendix the MATLAB codes resulting from this work are, almost entirely, reported.

# Table of Contents

# Chapter 1

# Harmonic Balance Analysis of Nonlinear Circuits

The Harmonic Balance (HB) method is a numerical method to determine the steady-state solution of nonlinear problems. It was firstly introduced to study nonlinear differential equations. In this field, Cesari [1] and Urabe [2] derived the mathematical framework of the method. Later, the works of Yamada and Bessho [3] and others [4] extended the method with Finite Element analysis, giving rise to the so called Harmonic Balance Finite Element method (HBFEM).

The Harmonic Balance method is a frequency-domain method, which combines time and frequency representations in order to achieve an advantage compared with the classical time-consuming transient analysis. In fact, if we are only interested in the solution, the HB method can determine it directly. On the other hand, a transient solver has to explicitly solve several time-stepping computations that could become really expensive, in particular with large time constants, in order to reach the solution.

The purpose of this chapter is to simulate the steady state solution of a simple electrical circuit, with a strong nonlinear element: a diode. A similar algorithm can be used to simulate other kinds of circuits, in which nonlinearities come, for example, from microwave or RF components.

There are two main formulations of the same method, namely the **Nodal Harmonic Balance method** and the **Piecewise Harmonic Balance method**. The first one considers an entire circuit applying Kirchhoff's laws in every node, giving rise to a large and, generally, full nonlinear system. In this case, each variable is represented by a Fourier series expansion, whose coefficients are the unknowns of the problem. It is a relatively easy approach but, due to the high number of variables, it is not widely used.

The Piecewise Harmonic Balance method, instead, has emerged as the most convenient approach, and is analysed in detail hereafter. Nowadays, due to its diffusion, the Piecewise Harmonic Balance method is simply identified as Harmonic Balance, and so is done in this work.

## 1.1 Circuital HB Method

The Harmonic Balance method, applied to circuits, requires to divide into two subsystems the system that needs to be analysed: a linear and a nonlinear part. The two subsystems are connected by $M$ ports, as shown in figure 1-1. The number of nonlinear ports has to be reduced in order to reduce the complexity of the problem.

The aim of the Harmonic Balance algorithm is to find the set of port voltages that satisfy Kirchhoff's current law at each port: the "linear" current has to be equal to the "nonlinear" one in each port.

The port's voltage can be expressed with the Fourier Series Expansion:

$$v^m = Re\left(\sum_{n=0}^{N_H} V_n^m \cdot e^{jn\omega_0 t}\right) \tag{1.1}$$

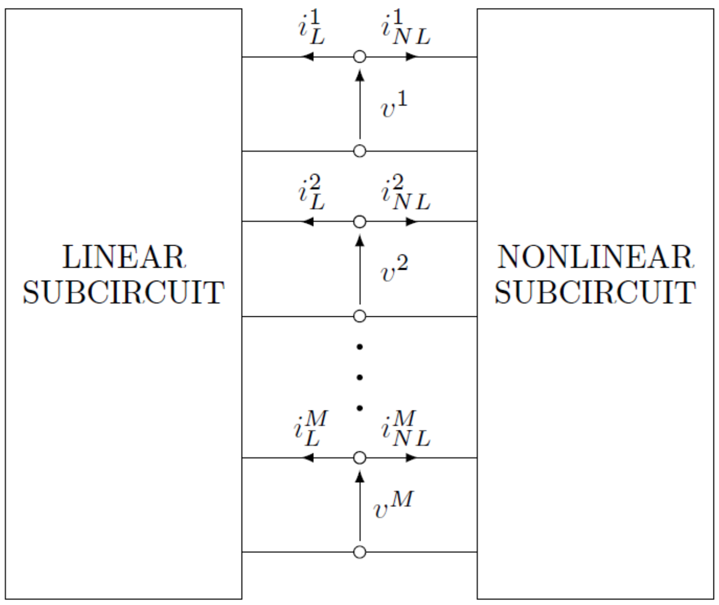


**Figure 1-1:** General circuit divided into a linear and a nonlinear subsystem and connected by *M* ports.

where $V_n^m$ are the unknown phasors of the problem. At the same port, Kirchhoff's current law has to be verified, so:

$$i_L^m(t) + i_{NL}^m(t) = 0 \qquad\qquad m = 1, ..., M \tag{1.2}$$

where $i_L^m(t)$ represents the linear current at the $m$ port, and $i_{NL}^m(t)$ represents the nonlinear current at the same port.

All these terms can be rearranged into a vector, in which the dimensions depend on the number of ports and on the harmonics that are chosen: $M(N_h + 1)$.

$$\mathbf{V} = \begin{bmatrix} V_0^1 & V_1^1 & \ldots & V_0^2 & \ldots V_{Nh}^2 & \ldots & V_{Nh}^M \end{bmatrix} \tag{1.3}$$

Therefore, the previous equations can be rearranged into a vectorial form.

Since the current and voltage signals are real, all the vectors previously defined in frequency domain present hermitian symmetry. Thus, the negative frequency components are the complex conjugate of the positive ones. This observation allows to take into account only positive frequencies plus the DC term, so the dimensions of the problem are reduced to $M(\frac{N_h}{2} + 1)$.

The solution to the problem is found thanks to the combination of the solutions of the two separated sub-circuits. The linear sub-circuit is analysed in the frequency domain, using the superposition of effects principle. The nonlinear circuit, instead, is analysed in the time domain.

### 1.1.1 Linear Subircuit

By means of the Norton equivalent theorem, as can be seen in figure 1-2, the linear circuit can be expressed by the relation:

$$\mathbf{I}_L = \mathbf{Y} \cdot \mathbf{V} + \mathbf{I}_{eq} \tag{1.4}$$

$\mathbf{Y}$ is the admittance matrix, while $\mathbf{I_{eq}}$ is the vector of the equivalent current, that has the same construction as the voltage vector in equation 1.3.

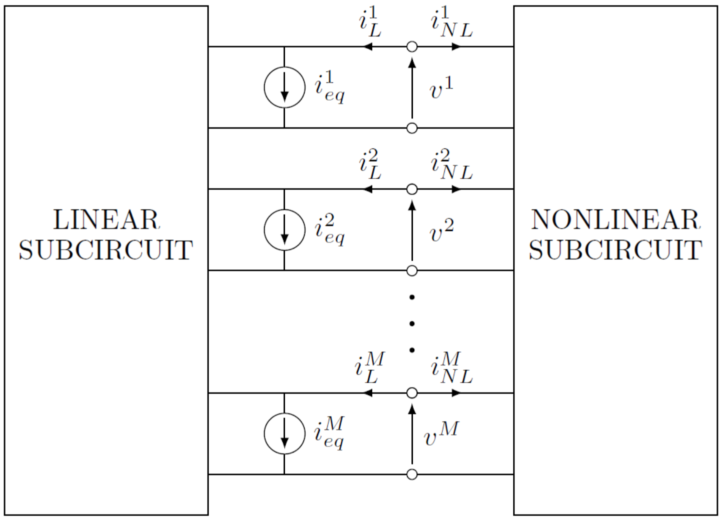


**Figure 1-2:** Norton equivalent of the linear sub-circuit.

The admittance matrix **Y** is a sparse block matrix, where each block is a diagonal matrix representing the value $Y^{i,j}$ between the ports $i$ and $j$ at each harmonic:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{Y}^{1,1} & \mathbf{Y}^{1,2} & \dots & \mathbf{Y}^{1,M} \\ \mathbf{Y}^{2,1} & \mathbf{Y}^{2,2} & \dots & \mathbf{Y}^{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{Y}^{M,1} & \mathbf{Y}^{M,2} & \dots & \mathbf{Y}^{M,M} \end{bmatrix} \tag{1.5}$$

Each submatrix can be build as following:

$$\mathbf{Y}^{i,j} = \begin{bmatrix} Y_0^{1,1} & 0 & \dots & 0 \\ 0 & Y_1^{2,2}(\omega_0) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & Y_{N_h}^{M,M}(N_h \cdot \omega_0) \end{bmatrix} \tag{1.6}$$

The self and mutual admittances at each port have to be calculated for each frequency solving different circuits. For this purpose, methods like Nodal Analysis or Tableau Analysis can be used.

### 1.1.2 Nonlinear Subcircuit

The analysis of the nonlinear circuit is carried out in the time domain, so it has to follow a different approach. As a first step, the vector of the time domain voltage has to be computed from frequency domain, by means of the inverse Fourier transform:

$$\mathbf{v}(t) = \mathscr{F}^{-1}(\mathbf{V}) \tag{1.7}$$

The second step comes directly from the knowledge of the constitutive law of the circuit part in the nonlinear circuit, therefore $\mathbf{i}_{NL} = f(\mathbf{v}(t))$ and the system becomes:

$$\mathbf{I}_{NL} = \mathscr{F}(f(\mathbf{v}(t))) \tag{1.8}$$

These steps are carried out using the Fast Fourier Transform: the **FFT** algorithm. Therefore, the number of samples must satisfy the Shannon's sampling theorem hypothesis, in order to avoid aliasing or any other numerical error.

A recap of the procedure is reported below:

$$\mathbf{V} \xrightarrow{\mathscr{F}^{-1}} \mathbf{v}(t) \xrightarrow{f(v(t))} \mathbf{i}_{NL}(t) \xrightarrow{\mathscr{F}} \mathbf{I}_{NL} \tag{1.9}$$

### 1.1.3 Solving the System

The nonlinear system to solve consists in:

$$\mathbf{f}(\mathbf{V}) = \mathbf{I}_L(\mathbf{V}) + \mathbf{I}_{NL}(\mathbf{V}) = 0 \tag{1.10}$$

where the unknowns of the problem are the phasors of the truncated Fourier series expansion of the **voltage** at the ports of the whole circuit. The residue $\mathbf{f}(\mathbf{V})$ is the function that the algorithm has to minimize and it is called current-error vector.

As previously said, the critical aspect of the Harmonic Balance Method is the great number of degrees of freedom: a generic circuit with $M$ ports and $N_h + 1$ harmonics ($N_h$ harmonics + DC term) gives rise to a nonlinear system with $M(N_h + 1)$ variables. If we consider both the real and imaginary parts, the total number of unknowns grows to $2M(N_h + 1)$. For the solution of this problem, a common Newthon-Raphson's method is chosen, as described by the following relation:

$$\mathbf{V}^{(k+1)} = \mathbf{V}^{(k)} - (\mathbf{J}^{(k)})^{-1} \cdot \mathbf{F}(\mathbf{V}^{(k)}) \tag{1.11}$$

where $\mathbf{J}$ is the Jacobian Matrix that has this structure:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial F_1^0}{\partial V_1^0} & \frac{\partial F_1^0}{\partial V_1^1} & \cdots & \frac{\partial F_1^0}{\partial V_N^{N_h}} \\ \frac{\partial F_1^1}{\partial V_1^0} & \frac{\partial F_1^1}{\partial V_1^1} & \cdots & \frac{\partial F_1^1}{\partial V_N^{N_h}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_1^{N_h}}{\partial V_1^{N_h}} & \frac{\partial F_1^0}{\partial V_1^1} & \cdots & \frac{\partial F_1^{N_h}}{\partial V_N^{N_h}} \end{bmatrix} \tag{1.12}$$

where the subscripts denote the ports, whereas the superscipts indicate the harmonic order.

The Jacobian can also be calculated as:

$$\mathbf{J} = \mathbf{Y} + \frac{d\mathbf{I}_{NL}}{d\mathbf{V}} \tag{1.13}$$

The Jacobian matrix can be computed analytically, if the nonlinear function is known in analytical form, or numerically, by incremental ratio. The analytical derivation, however, has better numerical properties, and it is advisable when available.

When the vector of the error currents decreases below a toleration limit $\varepsilon$, the algorithm stops, and the solution is found:

$$||\mathbf{f}^{(k)}|| < \varepsilon \tag{1.14}$$

### 1.1.4 The Source Stepping Method

The choice of the first guess is a key point to speed the convergence of the algorithm. For this reason, the linear solution, obtained for a low-level input, is a good starting point for the iteration of nonlinear circuits. The level of the input is then gradually incremented by steps ($\lambda$) through the iterations. This is the so called Source Stepping Method.

$$\mathbf{V}^{(k+1)} = \mathbf{V}^{(k)} - \lambda(\mathbf{V}^{(k)})(\mathbf{J}^{(k)})^{-1} \cdot \mathbf{F}(\mathbf{V}^{k}) \tag{1.15}$$

An overview of several other techniques for the analysis of nonlinear circuits can be found in [5].

## 1.2 Example: Half Wave Rectifier

In order to test the algorithm presented above, a simple nonlinear circuit was studied. The circuit in figure 1-3 has a strong nonlinearity caused by the diode. The algorithm was implemented in MATLAB, and in Appendix A some of the main scripts are reported.

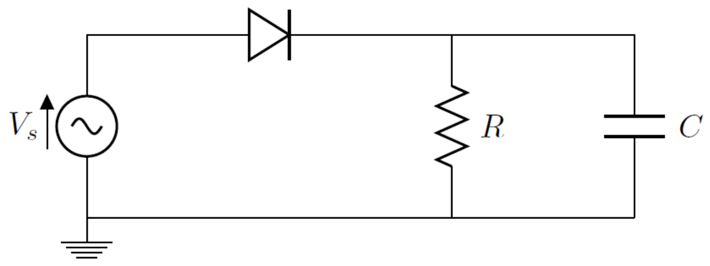


**Figure 1-3:** Half-wave rectifier.

This simple circuit can be easily rearranged in a pattern similar to the one in figure 1-1, where the linear and nonlinear components are divided, and there is only one port in between (figure 1-4). Because of the strong nonlinearity introduced by the diode, a Source Stepping Technique has to be implemented as discussed in the section 1.1.4.

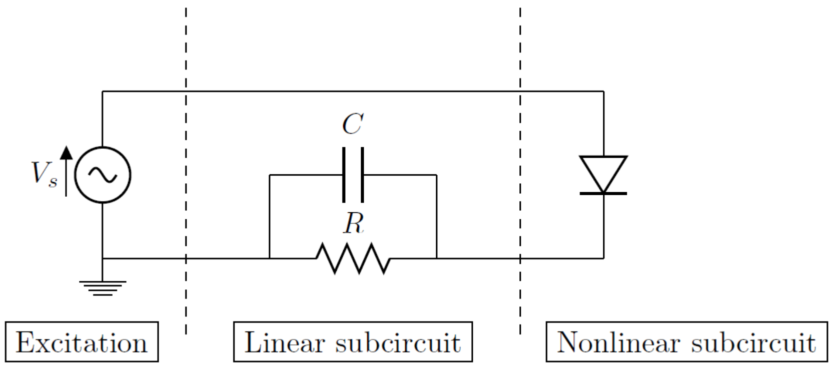


**Figure 1-4:** Half-wave rectifier rearranged.

The circuit is characterised by a sinusoidal source:

$$v_s = V_{max} sin(\omega t) \tag{1.16}$$

with $V_{max} = 1\ V$ and $\omega = 20\ kHz$. The diode is characterised by this equation:

$$i_d = I_d(e^{\frac{v}{V_t}} - 1) \tag{1.17}$$

with reverse current $I_d = 0.1\ nA$ and threshold voltage $V_t = 0.025\ V$. The value of the resistor is set to $R = 100\ \Omega$ while the capacitance is $C = 0.1\ mF$.

The circuit was studied with different numbers of harmonics $N_h$ in order to evaluate the accuracy of the method. The higher the number of the harmonics is, the higher the accuracy of the results is, but also the higher computational costs are. A number of harmonics equal to 16 is sufficient to simulate the circuit and to guarantee an optimal quality of the results.

To compute the Fourier Transform and the Inverse Fourier Transform, the MATLAB *fft* and *ifft* algorithms were adopted. The number of harmonics and the behaviour of the *fft* algorithm have to match. The *fft* code produces a vector of $N_h$ components that has these characteristics:

$$\mathbf{V} = [V_{dc} \, , \, V_1 \, , \, V_2 \, , \ldots , \, V_{max-1} \, , \, V_{max} \, , \, conj(V_{max-1}) \, , \ldots , \, conj(V_2) \, , \, conj(V_1)] \tag{1.18}$$

where the *DC* and the *max* term appear only one time, while all the other components appear also as their complex conjugate.

In figure 1-5 the voltage on the Diode is plotted. It is the direct output of the code for a simulation with 16 harmonics. This vector of voltage is obtained by the *ifft* of the voltage vector in the frequency domain.

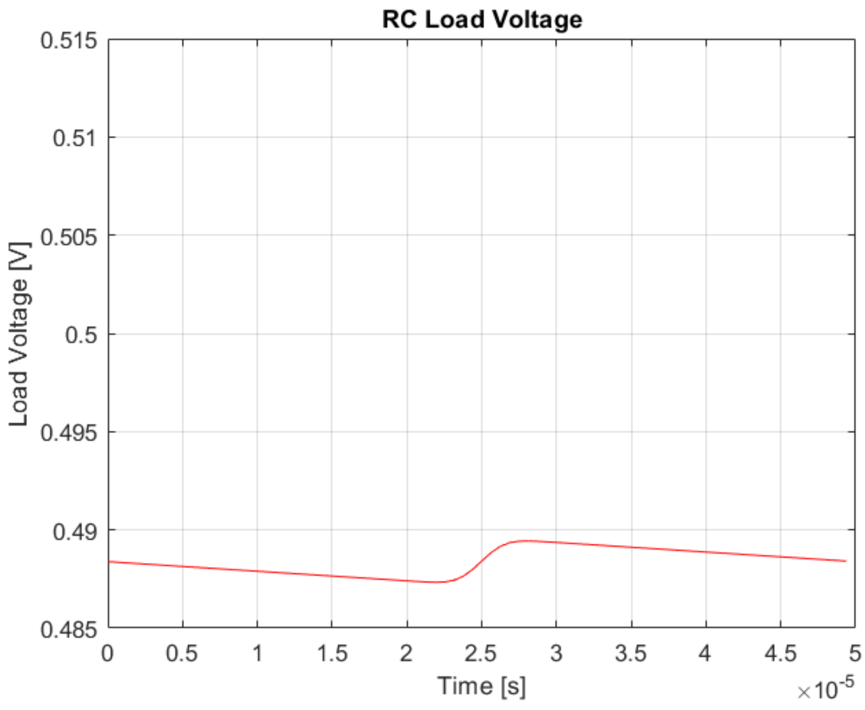


**Figure 1-5:** Load Voltage.

Both the figures 1-6 and 1-7 are rather interesting for different reasons. Both are plots of the current on the diode, but in figure 1-6 it is clear why the number of harmonics of the simulation is so important. A low $N_h$ ensures high speed of the computation, but the errors introduced may be too high (up to 50% in the case of only 4 harmonics).

In figure 1-7 there is a comparison between plotting directly the output of the *ifft* algorithm, versus reconstructing the signal from the Fourier coefficients in the frequency domain and then plot the solution. In appendix A.3 this simple code is reported. However, it has to be implemented with careful attention to the harmonics coefficients.

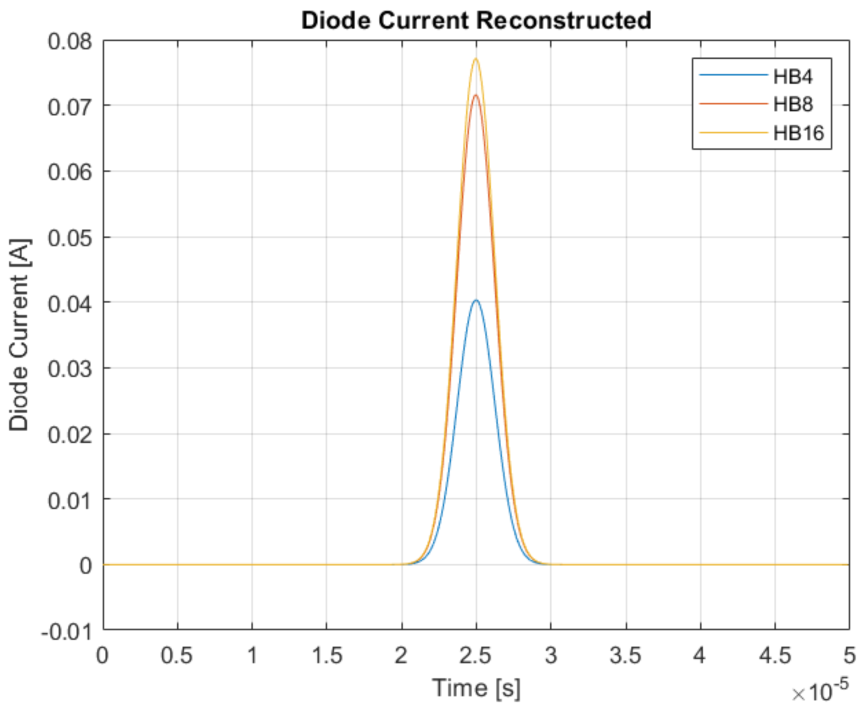


**Figure 1-6:** Different number of harmonics comparison.

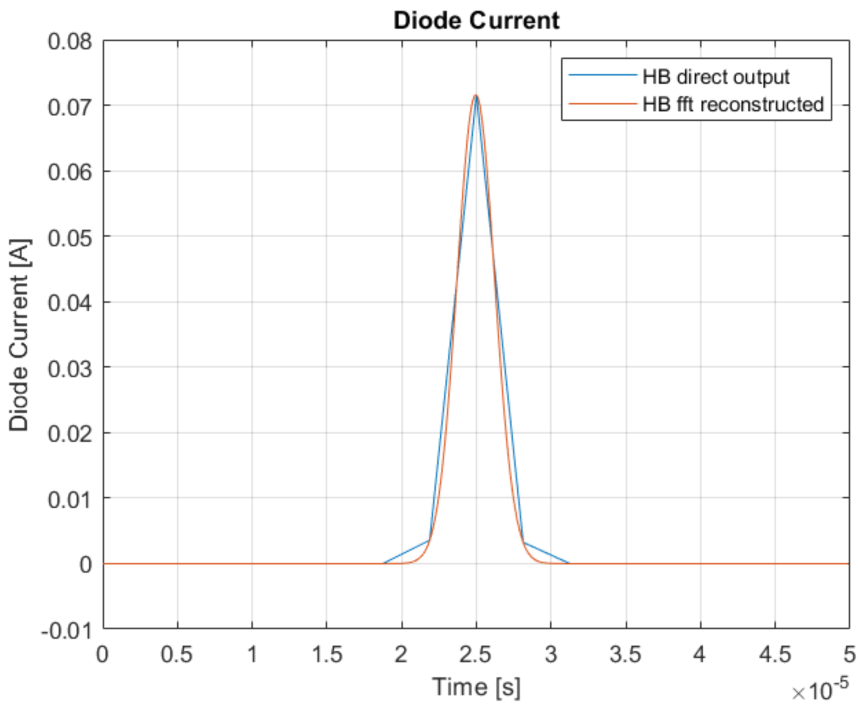


**Figure 1-7:** Diode Current.

# Chapter 2

# Harmonic Balance Finite Element Method

The main purpose of this thesis is the study of electromagnetic fields in time domain. Usually, this kind of problems is studied with a time domain solver in FEM, because the presence of nonlinearities does not allow the use of a time-harmonic solver. An Harmonic Balance method can be an interesting solution to this kind of problems **if** we are interested in the steady-state solution and not in the transient regime.

In this chapter the Harmonic Balance Finite Element Method is introduced, starting from a basic recap of the FEM algorithm.

## 2.1 Physical Problem

In this section the physical laws that manage the electromagnetic field are recalled .

### 2.1.1 Maxwell's Equations

Electromagnetic fields are described by Maxwell's equations:

- Faraday-Neumann-Lenz law:
$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{2.1}$$
- Ampère-Maxwell law:
$$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t} \tag{2.2}$$
- Gauss law for the magnetic induction field:
$$\nabla \cdot \mathbf{B} = 0 \tag{2.3}$$
- Gauss law for the electric induction field:
$$\nabla \cdot \mathbf{D} = \rho \tag{2.4}$$

In these equations, "$\nabla\times$" indicates the curl operator, "$\nabla\cdot$" indicates the divergence operator, **E** is the electric field, **H** is the magnetic field, **J** is the current density,**B** is the magnetic induction field, **D** is the electric induction field and $\rho$ is the charge density.

It is also important to recall the charge conservation equation, that is obtained from the previous equations:
$$\nabla \cdot \mathbf{J} = -\frac{\partial \rho}{\partial t} \tag{2.5}$$

In the so called *magneto-quasi-static case*, Maxwell's equations can be simplified as follow:

$$\nabla \times \mathbf{H} = \mathbf{J} \tag{2.6}$$

$$\nabla \cdot \mathbf{J} = 0 \tag{2.7}$$

### 2.1.2 Material Laws

The material's laws for linear materials are:

$$\begin{cases} \mathbf{D} = \varepsilon \mathbf{E} \\ \mathbf{J} = \sigma \mathbf{E} \\ \mathbf{B} = \mu \mathbf{H} \end{cases} \tag{2.8}$$

where $\varepsilon$ is the permittivity, $\sigma$ is the conductivity and $\mu$ is the magnetic permeability. However, iron, which is a very common material, is nonlinear and the relation looks like the curve shown in Fig 2-1 (neglecting hysteresis). For H values larger than a certain threshold (the so-called knee value) the slope decreases and then leans to $\mu_0$. This kind of materials are the sources of nonlinearities in the HBFEM study.

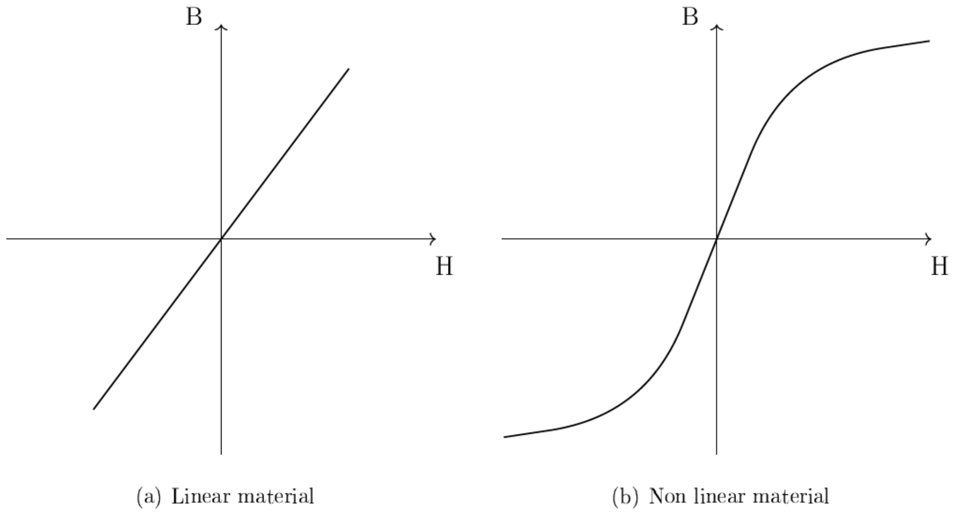


**Figure 2-1:** BH relation in a linear and a nonlinear material.

### 2.1.3 Eddy Current

Starting from the previous equations, the eddy current PDE is derived. From the definition of magnetic vector potential **A**:

$$\mathbf{B} = \nabla \times \mathbf{A} \tag{2.9}$$

we can reformulate the first Maxwell's equation (2.1) :

$$\nabla \times \mathbf{E} = -\frac{\partial}{\partial t}(\nabla \times \mathbf{A}) \tag{2.10}$$

and so:

$$\nabla \times \left( \mathbf{E} + \frac{\partial \mathbf{A}}{\partial t} \right) = 0 \tag{2.11}$$

If the domain is simply connected for lines, the electric scalar potential **V** can be introduced:

$$\mathbf{E} + \frac{\partial \mathbf{A}}{\partial t} = -\nabla \mathbf{V} \tag{2.12}$$

Now, adding the constitutive equations, the system results in:

$$\begin{cases} \mathbf{J} = \sigma\mathbf{E} = \sigma\left(-\nabla\mathbf{V} - \frac{\partial\mathbf{A}}{\partial t}\right) \\ \mathbf{H} = \nu\mathbf{B} \end{cases} \tag{2.13}$$

Therefore, substituting the current density in Eq.2.6 and using the magnetic vector potential definition, we arrive at the final equation that describes eddy current problems:

$$\nabla \times (\nu\nabla \times \mathbf{A}) + \sigma\frac{\partial\mathbf{A}}{\partial t} = \mathbf{J} \tag{2.14}$$

In 2D cases, as for the problems presented in this work, the equation can be simplified to a scalar equation, considering $\mathbf{A} = A_z(x,y)$:

$$-\nabla \cdot \nu\nabla A_z + \sigma\frac{\partial A_z}{\partial t} = J_{s,z} \tag{2.15}$$

## 2.2 Finite Element Method

The Finite Element Method (FEM) is a numerical method for solving PDEs. It is a widely used tool to study problems in engineering and mathematical physics. The formulation of this method is not part of this work, but some of the main aspects are hereafter recalled.

### 2.2.1 Space Discretization

The basic idea of FEM is to divide the entire domain in a finite number of elements, that can be arbitrary polygons (generally triangles in 2D). The domain divided in this way is called *mesh* and has a set of rules to run this process of discretization properly.

For every element, like the one in figure 2-2, a set of *shape functions* can be written, so that $U^{e_j}$ can be found:

$$U^{e_j} = \sum_{i=1}^{n_{e_j}} U_i\varphi_i \tag{2.16}$$

where $n_{e_j}$ is the number of nodes of the element $e_j$ and $\varphi_i$ are the interpolating functions.

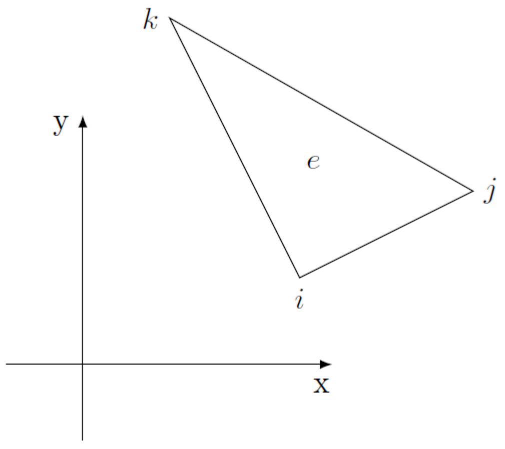


**Figure 2-2:** Triangular element.

In this case, the shape functions are:

$$\begin{cases} \varphi_i^e(x,y) = a_i + b_i x + c_i y \\ \varphi_j^e(x,y) = a_i + b_j x + c_j y \\ \varphi_k^e(x,y) = a_k + b_k x + c_k y \end{cases} \tag{2.17}$$

The value of the function $\varphi_i$ in the node $i$ is the union of all the contributions of all the elements that present the node $i$ as a vertex (as shown in figure 2-3).

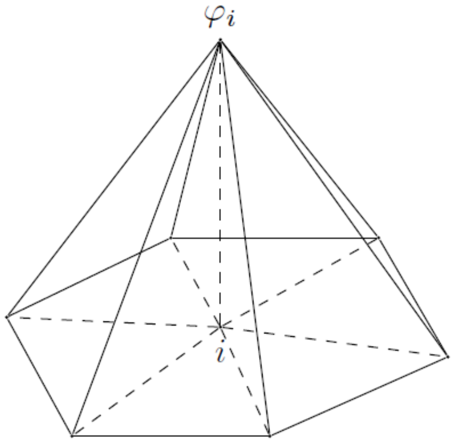


**Figure 2-3:** Basis function of the *i* node.

### 2.2.2 Stiffness Matrix

The stiffness matrix represents the system of linear equations that must be solved in order to find an approximate solution to the system of differential equation.

The **local stiffness matrix** is defined element by element:

$$K_{ij} = \int_\Omega \nabla\varphi_i \cdot \mu \nabla\varphi_j d\Omega \tag{2.18}$$

The **full stiffness matrix** is computed by the union of all the local matrices. It is a symmetric N by N very sparse matrix that depends only by the mesh of the system and by the material property of every element. Moreover, the final system derive still presents the nonlinearity, due to the terms containing the reluctivity. This problem is typically solved using Newton-Raphson; in this case the derivation of the algorithm is not straightforward. In fact, to apply Newton-Raphson it is necessary to build up the Jacobian matrix, which presents a complicated structure. Moreover, the linearized system obtained is nonsymmetric and is tipically solved using GMRES with ILUT preconditioning. Nonetheless, the problem was treated by different authors. The formal derivation of the Newton-Raphson method applied to the HBFEM can be found in [4] and [6].

## 2.3 The Harmonic Balance Method

In order to introduce the methodproperly, some considerations are needed. First of all, we are dealing with nonlinear eddy current problems. In this case, the nonlinearity is given by the dependence of the magnetic permeability on the magnetic field. This relation can be seen in the BH curve in figures 2-4.

In order to apply the HB method, we assume a periodical excitation source but, due to the nonlinearity, the result will not generally present the same harmonic content of the excitation, and it will be approximated by a Fourier series expansion.

Starting from the equation 2.14 previously obtained, we can derive an easier formulation with some hypotheses:

- the field is 2-dimensional
- the problem is quasi-stationary
- the saturated core is isotropic
- the hysteresis is not considered

Because of these statements, the magnetic vector potential results only in $z$ direction: $\mathbf{A} = (0, 0, A_z(x, y))$. The equation 2.14 becomes:

$$-\int_\Omega \nabla\varphi_i \cdot \nabla A_z d\Omega + \int_\Omega \varphi_i \sigma \frac{\partial A_z}{\partial t} d\Omega = \int_\Omega \varphi_i J d\Omega \tag{2.19}$$

We are looking for the steady-state solution, therefore all variables are approximated as a Fourier series expansion:

$$A_z^i(t) = \sum_{n=1}^{\infty} A_{ns}^i sin(n\omega t) + A_{nc}^i cos(n\omega t) \tag{2.20}$$

All the other variables of the previous equation can be approximated in the same way as $A_z$ in equation 2.20: $J(t)$, $v(t)$, $Bx(t)$ and $By(t)$.

The solution to this equation gives rise to a complicated assembly and a large, dense system of nonlinear equations. This happens because all the harmonics of the variables are coupled together by the explicit Fourier Expansion.

The idea hereby expressed represents the so called *strong-coupled approach* to the harmonic Balance. Now we want to formulate a similar method considering each harmonic decoupled from the other in such a way to study each harmonic independently. The nonlinear term containing the permeability (or the reluctivity) couples all the unknowns, so that the decoupling is trivial in the linear case, whereas for nonlinear problems, special techniques are needed.

Starting from the equation obtained from the Maxwell system 2.14:

$$\nabla \times (\nu \nabla \times \mathbf{A}) + \sigma \frac{\partial \mathbf{A}}{\partial t} = \mathbf{J}$$

the system can be rewritten as:

$$\mathbf{K}[\nu(\mathbf{x}(t))]\mathbf{x}(t) + \mathbf{M}(\sigma)\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(t) \tag{2.21}$$

This formulation makes use of the definition of the Stiffness Matrix **K** and the Mass Matrix **M**. The Stiffness matrix depends on $\nu$ that depends on **x** which in turn depends on time and is the nonlinear part of the equation.

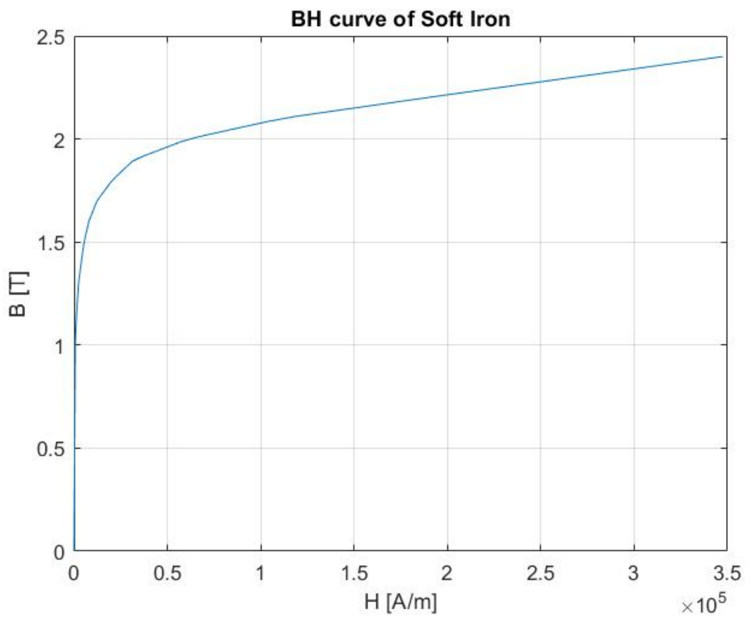


**Figure 2-4:** BH curve of soft iron.

Using a complex Fourier series for the solution, considering N harmonics we obtain this approximation to the solution:

$$\mathbf{x}(t) \approx \mathbf{x}_N(t) = Re\left(\sum_{k=1}^{N} \mathbf{X}_k e^{-jk\omega t}\right) \tag{2.22}$$

where $\mathbf{X}_k$ is the complex Fourier coefficient of the k-th harmonic at the angular frequency $k\omega$ and it can be computed as:

$$\mathbf{X}_k = \mathscr{F}_k(\mathbf{x}) = \frac{1}{T}\int_t \mathbf{x}(t)e^{-jk\omega t}dt \tag{2.23}$$

By implementing these definitions in the equation 2.21 we obtain a system that belongs to the frequency domain instead of the time domain:

$$\mathscr{F}_m\{\mathbf{K}[\nu(\mathbf{x}_N)]\mathbf{x}_N\} + jm\omega\mathbf{M}(\sigma)\mathbf{X}_m = \mathscr{F}_m(\mathbf{f}) \qquad m = 1,\ldots,N \tag{2.24}$$

The time derivative that multiplies the Mass Matrix corresponds to a multiplication by $jm\omega$ in the frequency domain. The nonlinear term, containing the permeability depending on the unknown solution, couples all Fourier coefficients to each other. Therefore, due to the nonlinearity, we cannot solve each harmonic independently.

### 2.3.1 Linear Problems

In the linear case the permeability and the reluctivity do not depend on the solutions, and in this case the decoupling is trivial. The Fourier coefficients related to the Stiffness matrix become:

$$\mathscr{F}_m\{\mathbf{K}(\nu)\mathbf{x}_N\} = \mathbf{K}(\nu)\mathbf{X}_N \tag{2.25}$$

The equation 2.24 can be easily determined because each harmonic is decoupled and results in $N$ independent linear systems:

$$[\mathbf{K}(\nu) + jm\omega\mathbf{M}(\sigma)]\mathbf{X}_m = \mathscr{F}_m(\mathbf{f}) \qquad m = 1,\ldots,N \tag{2.26}$$

### 2.3.2 Nonlinear Problems: Fixed Point Method

In nonlinear problems, each harmonic is coupled with harmonics due to the presence of the permeability. To manage this problem, a fixed point iteration is implemented, as presented in the Biro works [7] and [8]. As previously shown, the nonlinear behavior of the ferromagnetic material is given by the BH-curve and the following relation:

$$\mathbf{B} = \mu \mathbf{H} \tag{2.27}$$

The key idea of the fixed point algorithm is to separate the relation between **B** and **H** into a linear and a nonlinear term:

$$\mathbf{H}(\mathbf{B}) = \frac{1}{\mu_{fp}}\mathbf{B} - \mathbf{M}_{fp}(\mathbf{B}) \tag{2.28}$$

The term $\mathbf{M}_{fp}$ is a magnetization-like quantity which describes the nonlinearity and therefore depends on the flux density **B**. The constant factor $\mu_{fp}$ of the linear term is called the fixed-point permeability.

The starting equation 2.14, together with this definition, can be rewritten as follow:

$$\nabla \times \frac{1}{\mu_{fp}}(\nabla \times \mathbf{A}) - \nabla \times \mathbf{M}_{fp} + \sigma \frac{\partial \mathbf{A}}{\partial t} = \mathbf{J} \tag{2.29}$$

This equation by means of the stiffness and mass matrices becomes:

$$\begin{cases} \mathbf{K}_{fp}\mathbf{A} + jm\omega \mathbf{M}_{mass} = \mathbf{b}_J + \mathbf{b}_M \\ \mathbf{b}_J = \int N_i \cdot \mathbf{J}_{sources} \\ \mathbf{b}_M = -\int N_i \nabla \times \mathbf{M} \end{cases} \tag{2.30}$$

In this system, $\mathbf{b}_J$ is the right hand side referring to the sources, while $\mathbf{b}_M$ is referring to the magnetization as seen in equation 2.28. The minus sign comes from properties of the nabla operator:

$$\nabla \times (N_i \mathbf{M}) = N_i \nabla \times \mathbf{M} + \nabla N_i \times \mathbf{M} \tag{2.31}$$

but in our case of study, the term $\nabla \times (N_i \mathbf{M})$ is null, so the equation becomes:

$$N_i \nabla \times \mathbf{M} = -\nabla N_i \times \mathbf{M} \tag{2.32}$$

and so the minus sign is explained.

The whole procedure of the fixed point iteration results in:

1. Choose the value for $\mu_{fp}$.

2. Set **M** to zero for the first iteration.

3. Solve the system 2.30 to find **A** and so the field **B**. The field **H** can be found entering **B** in the BH-curve.

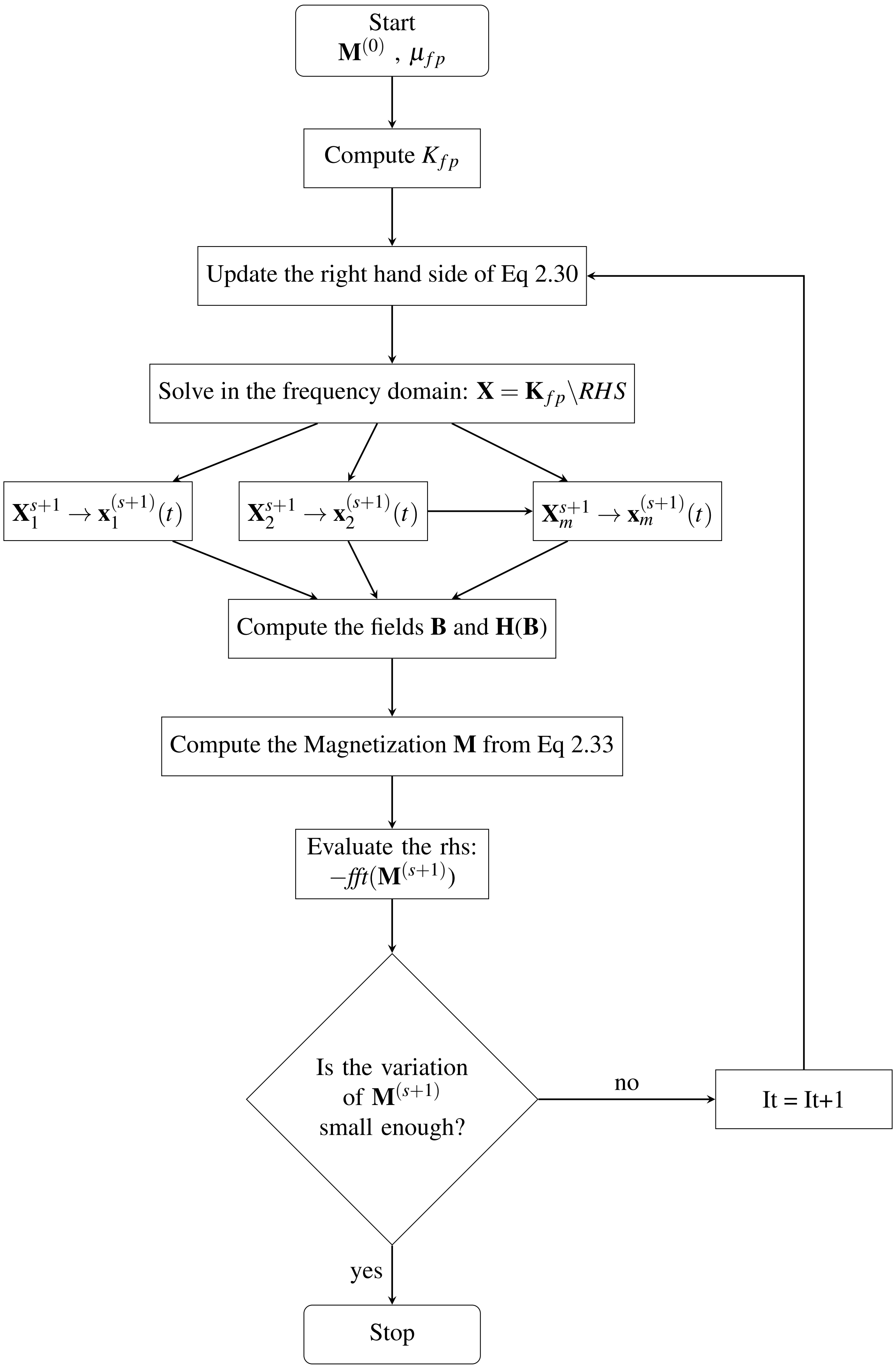
4. Find the magnetization **M** reversing the equation 2.28:

$$\mathbf{M} = \frac{1}{\mu_{fp}}\mathbf{B} - \mathbf{H}(\mathbf{B}) \tag{2.33}$$

5. Update the right-hand side of 2.30.

6. Repeat steps 3 and 4 until a certain criterion is fulfilled .

The right choice for $\mu_{fp}$ is fundamental: for $\mu_{fp} < 2$ the convergence of the method is secure but slow, for higher $\mu_{fp}$ the method is faster, but the convergence needs to be checked. The optimal convergence of the fixed-point method is studied in depth in [9]. The following flowchart shows the steps previously presented in an easier way to understand.


Start
$\mathbf{M}^{(0)}$ , $\mu_{fp}$
Compute $K_{fp}$
Update the right hand side of Eq 2.30
Solve in the frequency domain: $\mathbf{X} = \mathbf{K}_{fp} \backslash RHS$
$\mathbf{X}_1^{s+1} \rightarrow \mathbf{x}_1^{(s+1)}(t)$
$\mathbf{X}_2^{s+1} \rightarrow \mathbf{x}_2^{(s+1)}(t)$
$\mathbf{X}_m^{s+1} \rightarrow \mathbf{x}_m^{(s+1)}(t)$
Compute the fields $\mathbf{B}$ and $\mathbf{H}(\mathbf{B})$
Compute the Magnetization $\mathbf{M}$ from Eq 2.33
Evaluate the rhs: $-fft(\mathbf{M}^{(s+1)})$
Is the variation of $\mathbf{M}^{(s+1)}$ small enough?
no
It = It+1
yes
Stop

# Chapter 3

# Harmonic Balance FEM: Static Test Case

In this chapter, the Harmonic Balance method is presented, applied to different test cases, to study the field problem of static devices. Some strings of the code are commented below, while the whole code is reported in Appendix B.

In all the following test cases, the mesh is built in *Comsol* and then imported in *Matlab*. It is important to set in Comsol *linear elements* in the mesh. Comsol is then used to validate the results obtained. A FEM algorithm implemented in Matlab is also used to compare the results and to confront the computational time of the two methods.

## 3.1 Eddy Current-free problem: Ferromagnetic Yoke

In this first test case, the ferromagnetic yoke in figure 3-1 is simulated. The core consists in a non linear Iron (BH curve in fig 3-2) where the conductivity $\sigma$ of the material is set to zero, in order to have an eddy currents free problem. In order to achieve a strong saturation condition, we impose a sinusoidal current density with peak value equal to $1 \cdot 10^6 A/m^2$. The frequency is set to 50 Hz.

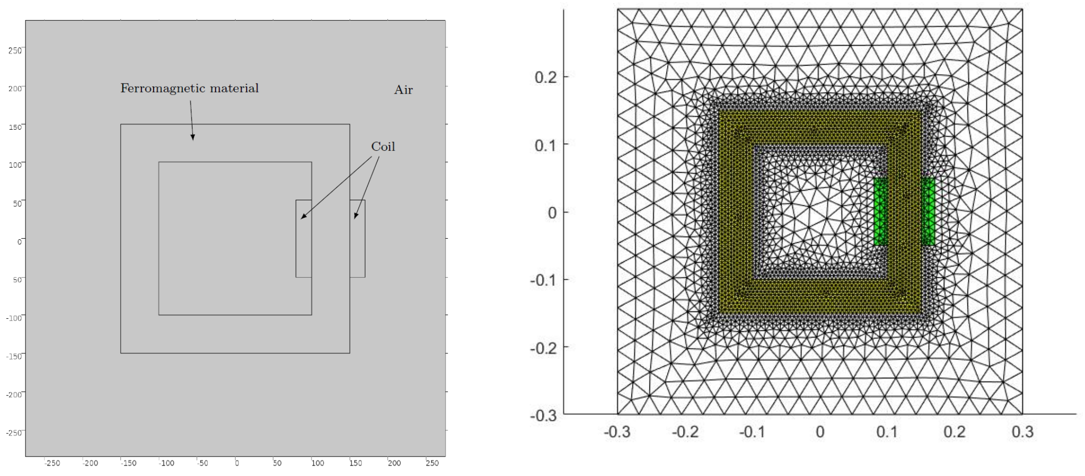


**Figure 3-1:** Geometry of the ferromagnetic yoke and mesh.

The principal aim of this test is to study the non linear behaviour of the iron. The equation 2.33 is implemented in the code, as can be seen below in the listing 3.1. The magnetization has to be evaluated in every time step, therefore a *for* loop is necessary.

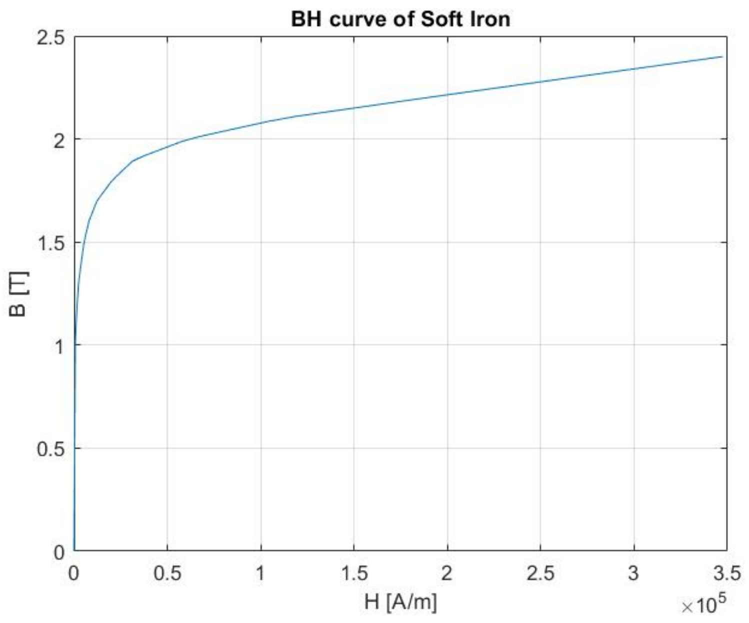


**Figure 3-2:** BH curve of the ferromagnetic material.

```
    for m=1:Nh         % For every time instant find the Magnetization
            xloc=myNull*sol_t(:,m);
            Bx = M1x*xloc;
            By = M1y*xloc;
            normB = sqrt(Bx.^2 + By.^2);
            [indB] = find(normB < 1E-9);          % Approximation of the null numbers
            normB(indB) = 1e-9;
            normH=normB/mu0;
%             Evaluate H from BH only in iron
            normH(IndIron) = H(normB(IndIron));
%             H bounded to B from the real relation BH
            Hx = normH./normB .* Bx;
            Hy = normH./normB .* By;
%             H bounded to B by niFP (only in iron)
            Hx_fp = nimu0*Bx; % air
            Hy_fp = nimu0*By; % air
            Hx_fp(IndIron) = nimu0*niFP*Bx(IndIron); % iron
            Hy_fp(IndIron) = nimu0*niFP*By(IndIron); % iron

            M(:,m) = myNull'*(M2x*(Hx-Hx_fp) + M2y*(Hy-Hy_fp));
            % The minus sign is inside M2y
        end
```

**Listing 3.1:** Fixed point iteration.

The results obtained from the algorithm were validated with the FEM algorithm and in Comsol. The most important result is that, compared to the FEM in which Newton-Raphson is needed to take care of the nonlinearities, HB is much quicker:

- FEM with Newton Raphson algorithm: about 850 seconds as calculation time, 23 NR iterations.

- HB with fixed point algorithm: about 11 seconds as calculation time, 188 FP iterations.

The choice of the $\mu_{fp}$ is critical: if it is below 2, the method always converges but it is slow. In this test for example, with $\mu_{fp} = 2$ it take 125 seconds as computation time and 4385 iterations. If instead $\mu_{fp} = 55$ the method is a lot faster, as presented above. With $\mu_{fp} = 60$ the method does not converge. At the moment, the choice of $\mu_{fp}$ is empirical, and needs to be investigated in future works.

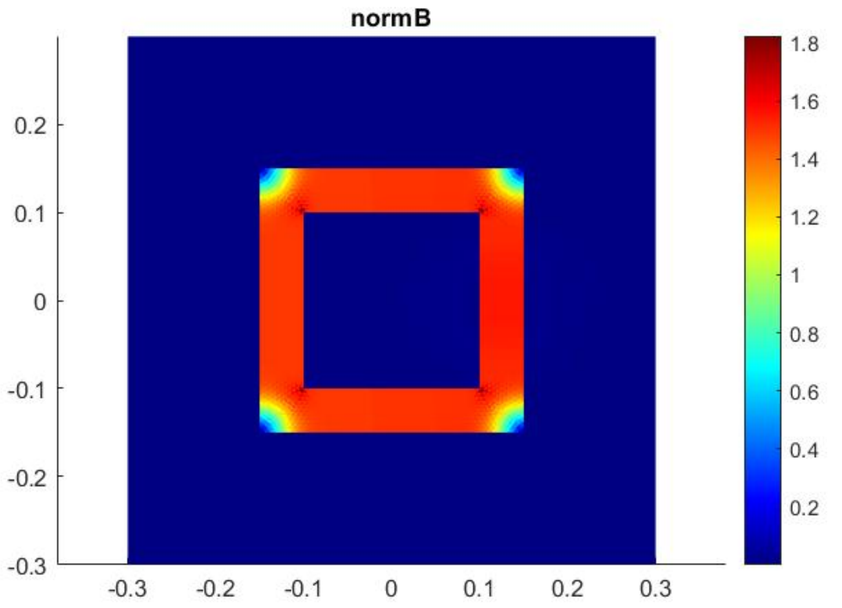


**Figure 3-3:** B field as result of the HB method.

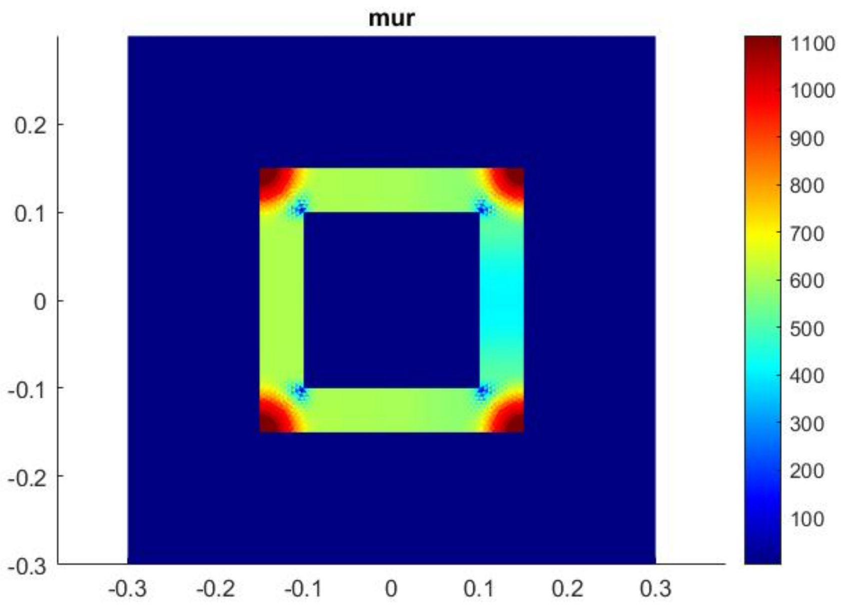


**Figure 3-4:** $\mu_r$ in the domain as result of the HB method.

In the two figures above, the field resulted from the first time step is shown. In these two figures, the behaviour of the non linear material is clear: it can be seen where the material is saturating and where it is not.

In the two figures below, instead, the real strength of the method is shown: the time domain solution. Figure 3-5 is particularly interesting because it shows the error that can be committed if a non-appropriate number of harmonics is used. The solution with 10 harmonics (red line) has a ripple that the 20-harmonics solution (yellow line) does not have.

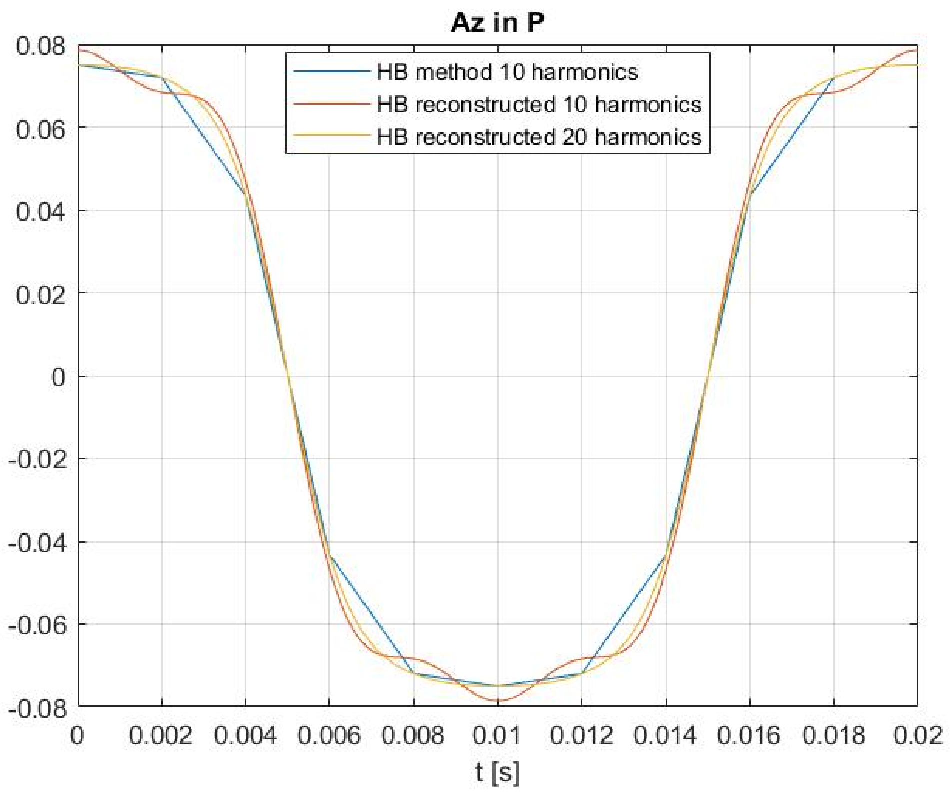


**Figure 3-5:** Time variation of $A_z$ in a point of the domain.

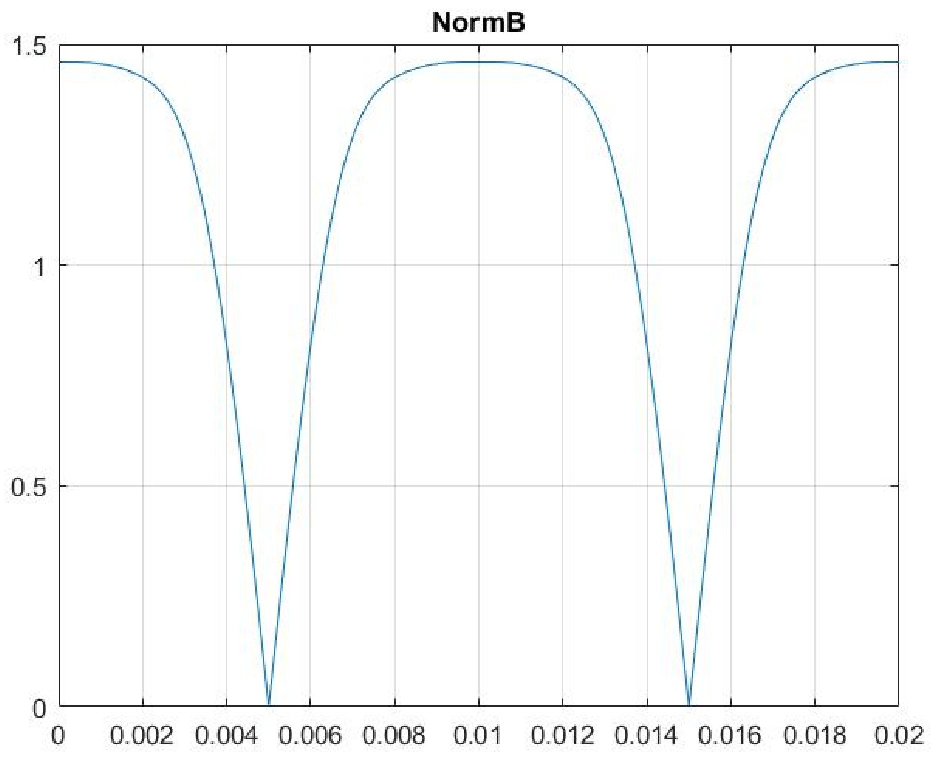


**Figure 3-6:** Time variation of NormB in a point of the domain.

## 3.2 Eddy Current Problem: Induced Currents

This test case is a simple extension of the previous one. A very simple geometry is considered: a plate of iron is excited by a coil above it.

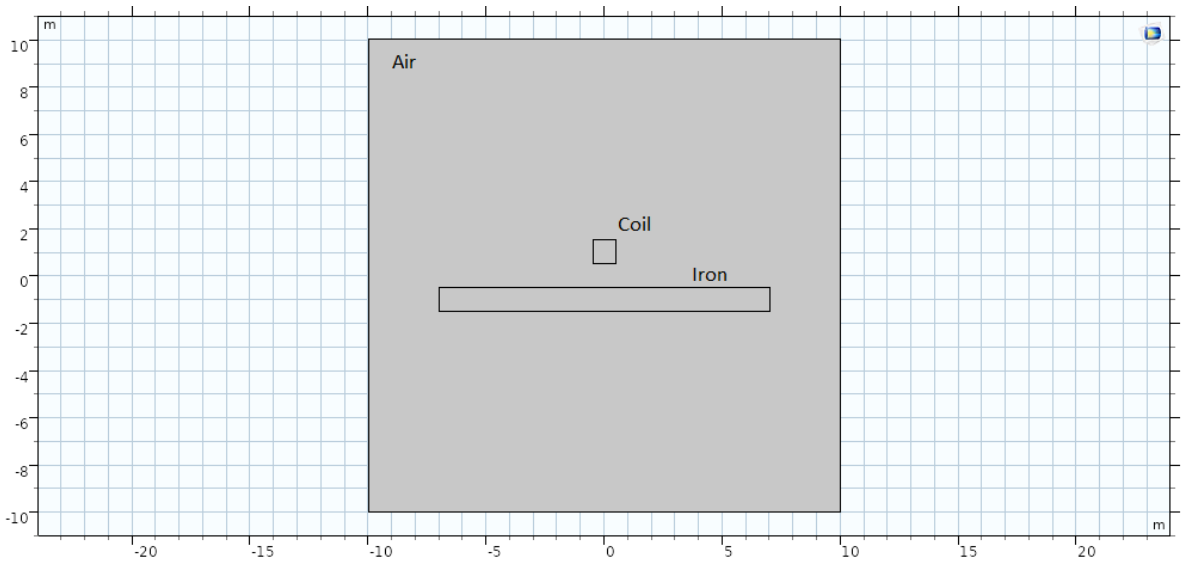


**Figure 3-7:** Geometry of the test case.

Because of the similarity with the eddy currents free problem, I am going to briefly discuss only the part of the code that implements the induced currents. As previously reported, the choice of $\mu_{fp}$ is a key point: it has to be set to 2 to ensure the convergence. The code takes 5 minutes and 7800 iterations to reach the solution.

```
1  while  check > toll && it < it_max
2   %Frequency domain rhs
3          rhs_f = -fft(M,[],2);
4          rhs_f(:,2) = (rhs_sources + rhs_f(:,2));           % Force the sources
5
6   % Frequency domain solution
7          for r = 1:(Nh/2+1)
8              sol_f(:,r) = (K_fp+1j*vec_harm(r)*omega*Mass)\rhs_f(:,r);
9              % Cycle to multiply omega to the number of harmonics
10         end
11         sol_f = [sol_f(:,(1:Nh/2+1)),fliplr(conj(sol_f(:,(2:Nh/2))))];
   % Second half of the vector computed as the conj of the first Nh+1 elements turned L-R
12
13         %Time domain solution
14         sol_t=ifft(sol_f,[],2,'symmetric');
15
16         it=it+1;
17 end
```

**Listing 3.2:** Eddy currents code.

To take care of eddy currents, the term of the **mass matrix** has to be added to the previous case. The system to solve now is the one in the equation 2.30. The *for* loop at the 7th row is needed to calculate the right omega for every different frequency.

The sources are forced into the right hand side at the right frequency. In this case the 50 Hz sources are set as first harmonic, so they are settled in the second column of the RHS vector (the first column is the DC term). Again, particular attention is needed while managing the *fft* coefficients.

In the following images, the results of the method are plotted for two different instants in time. The B field and the profile of the eddy currents are coherent with the Comsol results. The figures are only a pictures of two instants in time, but the method provides the solution in every time step as output.

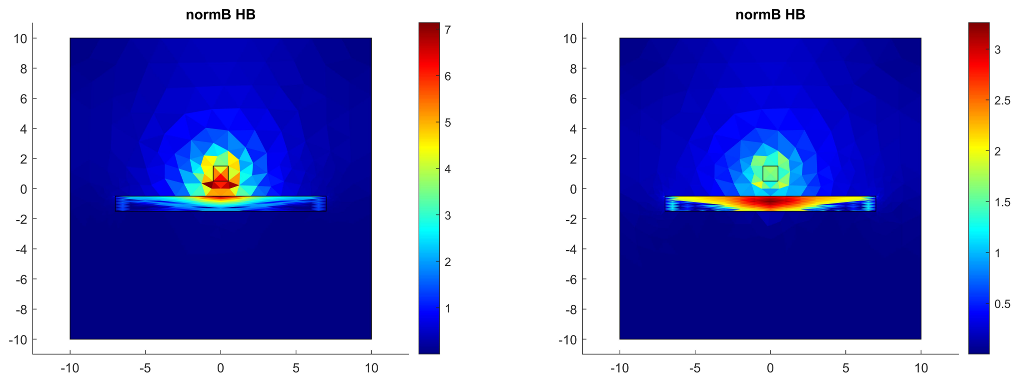


**Figure 3-8:** Norm of Field B in space for two different time instants.

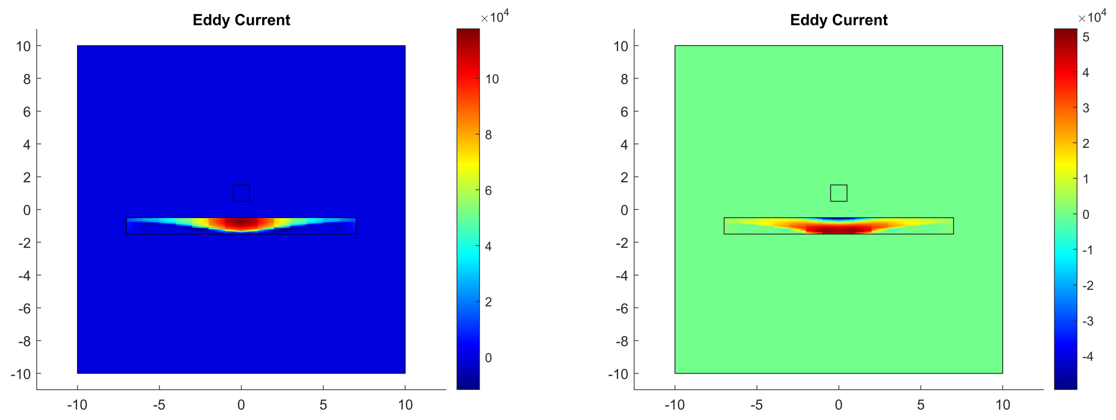


**Figure 3-9:** Eddy currents in the domain for two different time instants.

## 3.3 Permanent Magnet Motor with Locked Rotor

The last test case of this section approaches, for the first time, the real key point of this work: the field problem of a synchronous motor. In this case, we want to use the algorithm proposed above to study a permanent magnet synchronous motor in a locked rotor situation. In figure 3-10 the geometry and the mesh of the motor are shown.

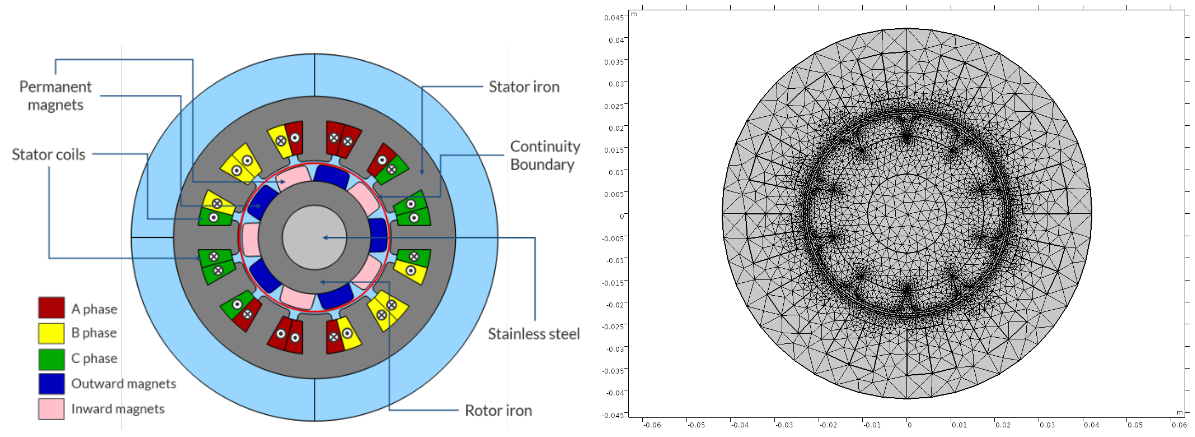


**Figure 3-10:** Geometry of the motor and mesh.

Despite the fact that the geometry is more complicated with respect to the previous problems, the code has to be only slightly adjusted. As first improvement, the permanent magnets need to be taken into account. In this case, magnets are made of NdFeB, with the norm of the remanent flux density equal to 1.47 T. The flux has therefore to be set with radial direction as shown in figure 3-11.

The value of 1.47 T (standard for NdFeB magnet) is a rather high value of induction, that forces the iron into the nonlinear part of the BH-curve. For this reason, and because of the higher number of elements, the fixed point method takes more iterations to reach the solution (figure 3-12) and the $\mu_{fp}$ has to be set to a value near 2 to assure convergence.

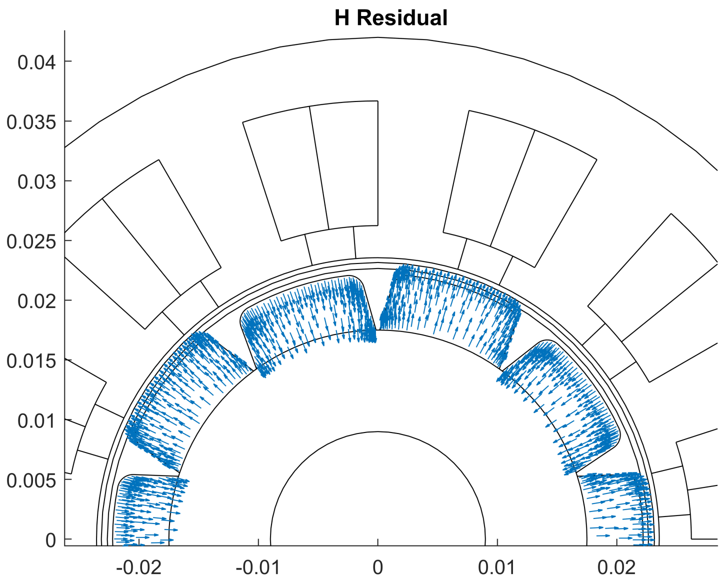


**Figure 3-11:** Quiver plot of the remanent flux density.

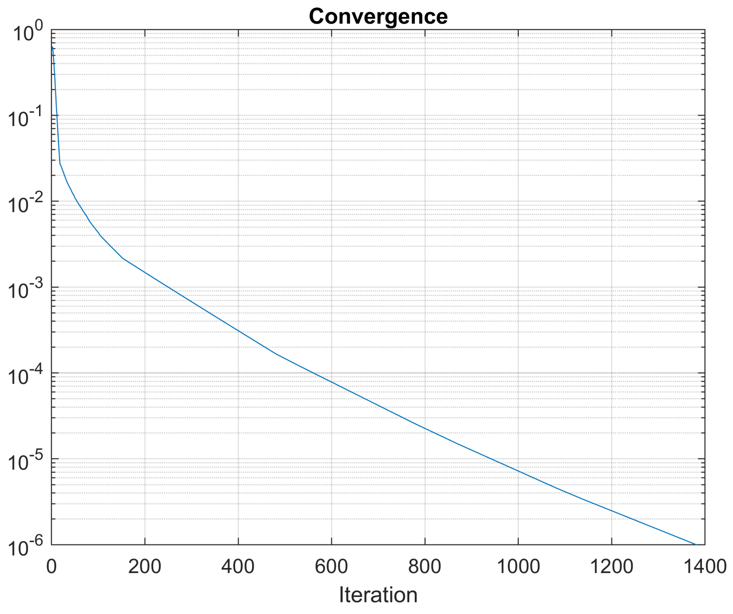


**Figure 3-12:** Plot of the convergence of the fixed point.

A second minor change needs to be made to take into account the three phase current. For sake of simplicity the density of the current $J$ is directly set in the coils. Both currents and magnets give contributions to the RHS, which is defined in the frequency domain and has this shape:

$$\mathbf{RHS} = \begin{bmatrix} b_1^{dc} & b_1^1 & \dots & b_1^{max} \\ b_2^{dc} & b_2^1 & \dots & b_2^{max} \\ \vdots & \vdots & \ddots & \vdots \\ b_n^{dc} & b_n^1 & \dots & b_n^{max} \end{bmatrix} \tag{3.1}$$

In the RHS matrix, each row refers to a node of the mesh and each column refers to a specific frequency weighed by the coefficients of the *fft* algorithm.

As in the previous section, the output of the code is the magnetic vector potential $A_z$ evaluated in different time steps. In figure 3-13 field B is plotted for two different instants in time, while in figure 3-14, the variation of $A_z$ over time as assessed in a point of the domain is directly shown.

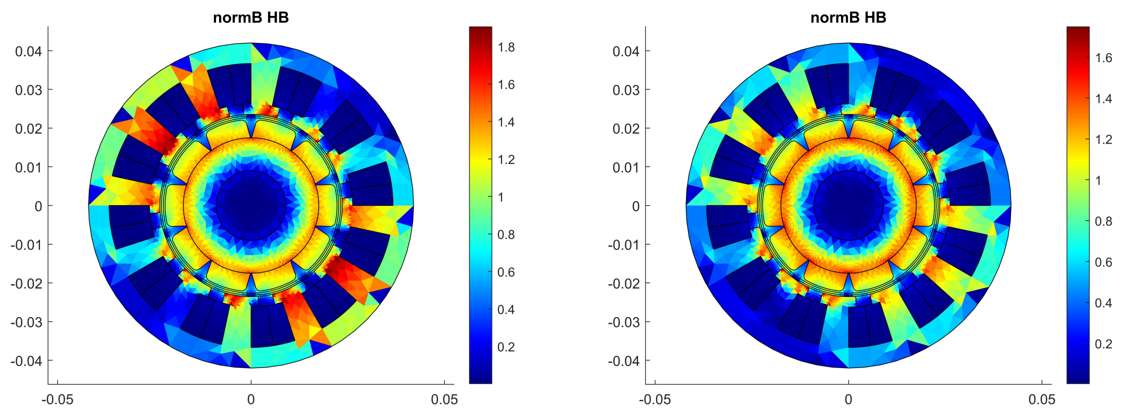


**Figure 3-13:** B field evaluated in two different time steps.

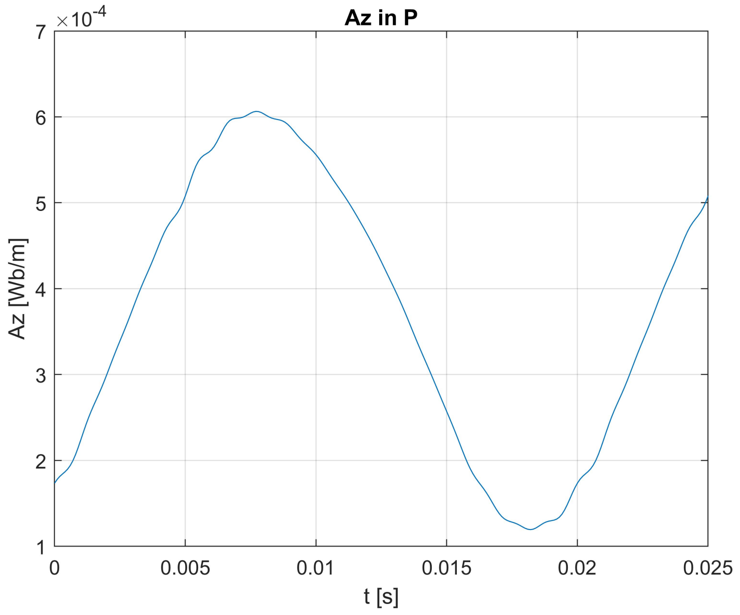


**Figure 3-14:** $A_z$ in a period.

Despite the fact that this way of analysing field problems is very promising, the implementation of a code to study in the frequency domain a rotating machine is very complex and little literature can be found on the subject. An extension of these codes applied to rotating machines, as proposed in [10], might represent an interesting future work.

# Chapter 4

# Time Domain Solution: Rotating Machines

The previous chapter reached the goal of studying the field problem of a synchronous motor in a locked rotor condition. The aim of this chapter is to add motion to the previous case. The time domain solution is reached by means of a *theta method* algorithm, first implemented in a linear situation, then extended to a nonlinear case of study. A different formulation of the method is then discussed and compared with the classical one.

## 4.1 Theta Method: A Brief Recall of the Algorithm

To introduce the theta method, the DAE system in equation 2.21 is recalled.

$$\mathbf{K}[\nu(\mathbf{x}(t))]\mathbf{x}(t) + \mathbf{M}_{mass}(\sigma)\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(t)$$

To solve the DAE system, the initial condition $x(0)$ is necessary. The system can then be written in the form:

$$\mathbf{M}_{mass}\dot{x} = \mathbf{s} - \mathbf{K}\mathbf{x} = \mathbf{fun}(\mathbf{x}) \tag{4.1}$$

where $\mathbf{s}$ is the RHS of the previous equation and is related to the sources. In this formulation, the time derivative of $x(t)$ is a function of $x(t)$ itself.

The previous expression can be integrated in a time interval, and considering $\mathbf{M}$ and $\mathbf{K}$ constant in the $\Delta t$, it results in:

$$\int_{t_n}^{t_{n+1}} \mathbf{M}\dot{x}dt = \mathbf{M}_{mass}(\mathbf{x}_{n+1} - \mathbf{x}_n) \tag{4.2}$$

while the right-hand side, with the help of the Mean value theorem for definite integrals, becomes:

$$\int_{t_n}^{t_{n+1}} \mathbf{fun}(\mathbf{x})dt = \Delta t \cdot \mathbf{fun}(\mathbf{x}^*) \tag{4.3}$$

The key point of the Theta Method is the assumption that $\mathbf{fun}(\mathbf{x}^*)$ can be written as the weighted average of the values that it has at the beginning and at the end of the interval:

$$\mathbf{fun}(\mathbf{x}^*) \approx \theta\mathbf{fun}_{n+1} + (1-\theta)\mathbf{fun}_n \qquad 0 < \theta < 1 \tag{4.4}$$

Applying these definitions to the first system we obtain:

$$\mathbf{M}_{mass}(\mathbf{x}_{n+1} - \mathbf{x}_n) = \Delta t(\theta(\mathbf{s}_{n+1} - \mathbf{K}\mathbf{x}_{n+1}) + (1-\theta)(\mathbf{s}_n - \mathbf{K}\mathbf{x}_n)) \tag{4.5}$$

that can be rearranged in the typical form:

$$(\theta\mathbf{K} + \frac{1}{\Delta t}\mathbf{M}_{mass})\mathbf{x}_{n+1} = (-(1-\theta)\mathbf{K} + \frac{1}{\Delta t}\mathbf{M}_{mass})\mathbf{x}_n + \theta\mathbf{s}_{n+1} + (1-\theta)\mathbf{s}_n \tag{4.6}$$

This formulation is the one that is used in the codes where:

$$\mathbf{M} = (\theta\mathbf{K}_{n+1} + \frac{1}{\Delta t}\mathbf{M}_{mass,n+1}) \tag{4.7}$$

$$\mathbf{MM} = -(1-\theta)\mathbf{K}_n + \frac{1}{\Delta t}\mathbf{M}_{mass,n} \tag{4.8}$$

The Theta-Method can be:

- **asymptotically conditionally stable** if $0 < \theta < \frac{1}{2}$. This means that the method is asymptotically stable if and only if some conditions on the left hand side are met. Since such conditions are very difficult to check in general, in practice asymptotic stability is guaranteed only if $\Delta t$ is small enough. Very small time steps may result in unrealistically long simulation times.
- **asymptotically unconditionally stable** if $\frac{1}{2} < \theta < 1$. This means that the method is certainly asymptotically stable without any requirements on the LHS. However accuracy still depends on the time step size, which should be kept small.

For these reasons theta is set equal to 0.51.

## 4.2 Spatial Discretization of the Moving Band

In order to allow the rotation of the rotor, the procedure to build the mesh needs to change. Until now, the entire procedure to build the mesh was demanded to Comsol. Now, instead, the procedure reported below has to be followed:

1. Build in Comsol the mesh in the entire domain except for the moving band, as can be seen in figure 4-1. On both sides of the moving band, the number of nodes must be defined and equal. In this case the same number of nodes is imposed in both the two circles, with equal space imposed between nodes.

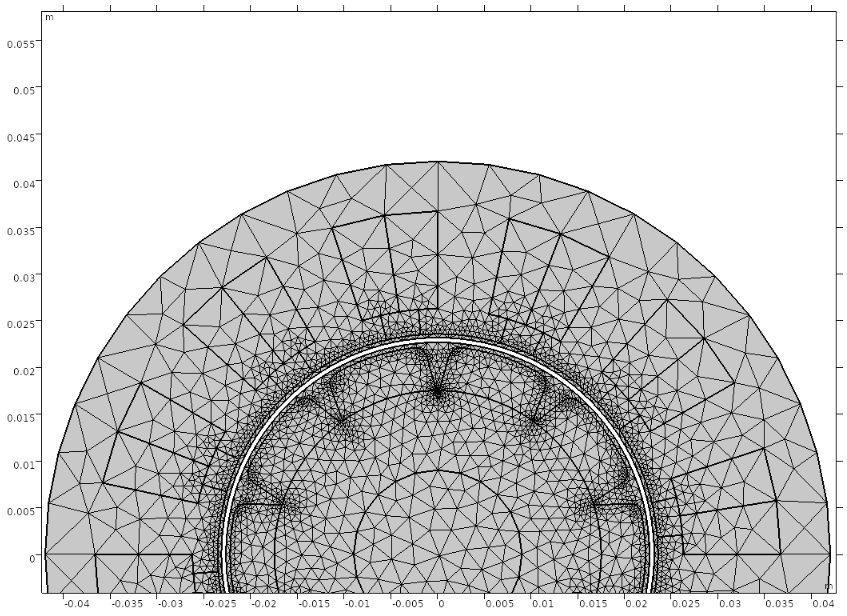


**Figure 4-1:** Particular of the mesh built in Comsol.

2. In Matlab, the mesh of the airgap is built, knowing the indices of the nodes of the two boundary regions.

3. When the rotor spins, the triangle of an element of the mesh of the airgap needs to change his shape to adapt to the rotation (first two plots in figure 4-2).
4. If the angle of rotation is great enough, all the triangles of the moving band have to change. In figure 4-2 three nodes are crossed to see how the moving of the rotor changes the mesh.
5. The mesh of the rotor does not change with the rotation because all the nodes of the rotor spin together.

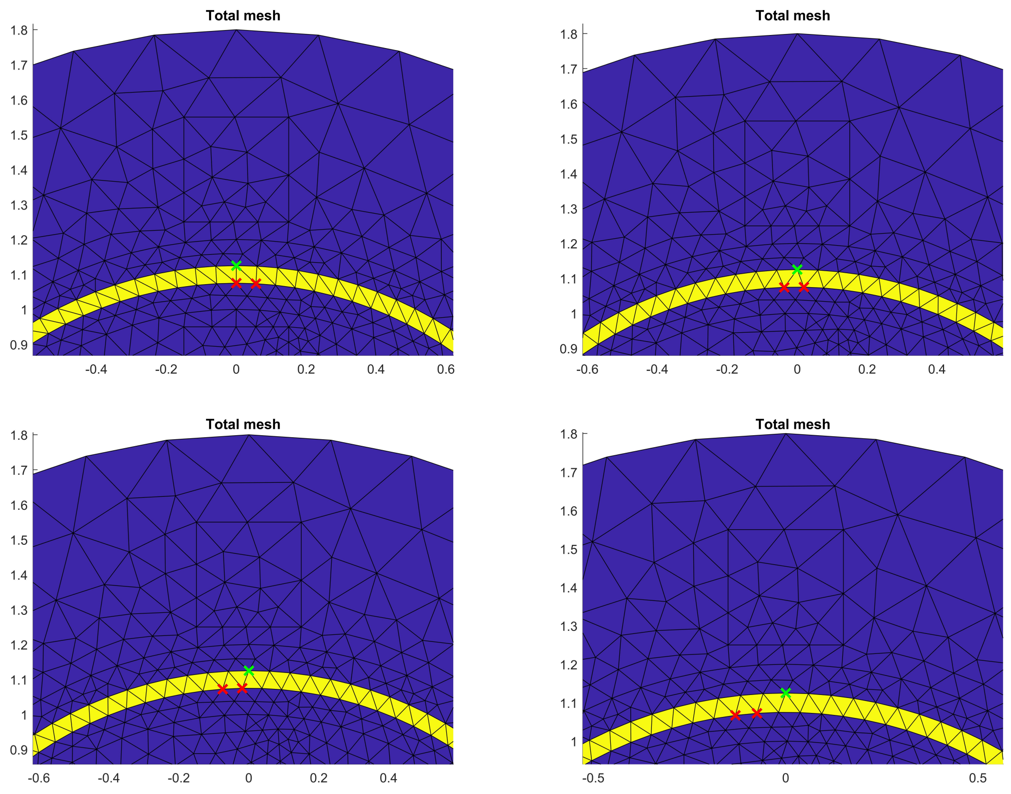


**Figure 4-2:** Mesh of the moving band.

In Appendix C.1 the function that build the mesh in the moving band is reported. As input the function needs the number of nodes to move the mesh. So the routine to find the number of nodes for the rotation as function of the degrees of rotations is:

```
1  numb_gap_nod = length(IndInt);   % Number of nodes ind the inner circle of the moving band
2  gradi_per_node = 360/numb_gap_nod;
3  rotaz = floor(grad/gradi_per_node);
4 [Tri] = renumerate_node(Pstart,IndInt,IndExt,rotaz);
```

At line three it is determined at how many nodes the degrees of rotation correspond. The *floor* operator assures that the mesh does not change until a right angle is reached.

## 4.3 Theta Method: Linear Field Problem

The first test case, for the theta method algorithm presented above, is the same synchronous motor studied with the harmonic balance method in section 3.3. Previously, the motor was studied in a locked rotor situation. Now, instead, the algorithm is easily implemented taking into account the rotation of the rotor.

The problem that needs to be studied is linear and, instead of the iron, there is air with the conductivity of iron. Despite the fact that, in a real motor, iron is laminated in order to not have eddy currents, this test considers the iron as a single block, to evaluate the eddy currents into the entire geometry.

In the code below the iterative cycle implementing the theta method is reported. The matrices **M** and **MM** are the same matrices introduced in the equations 4.7. The loop starts with the rotation of the rotor and the building of the new mesh. Then, currents and magnets need to be updated (this part is not reported in this scheme). After building the theta method matrices, the system is resolved and the solution stored. The complete code can be found in appendix C.2.

```
for iter = 1:nDt
    disp(['iteration = ',num2str(iter)])
    P = Pstart;                   % Restart from angle=0

    % Rotation
     angle = steps(iter)*rps;      % Angle in radians for the rotor rotation function
     grad = angle/pi*180;          % Angle in degrees that will be divided
     rotaz = floor(grad/gradi_per_node);     % Number of nodes of rotation
     [Tri] = renumerate_node(P,IndInt,IndExt,rotaz);    % Function to mesh the moving band
     [P] = Rot_Rotation(P,IndRot,angle);                % Function to rotate the rotor
     Ttot = [Told;Tri];

    for i = 1:length(Ttot(:,1))           % Find new center of gravity
        xyzb(i,1:2) = (P(Ttot(i,1),1:2)+P(Ttot(i,2),1:2)+P(Ttot(i,3),1:2))/3;
    end

    % Set of the currents
        [...]

    % Set of the magnets
        [...]

    [M1] = fun_stiff_matrix(Ttot,P,mu_fp,myNull);     % New stiffness matrix
    [M2] = fun_mass_matrix(sigma,Ttot,P,myNull);      % New mass matrix
    % Theta method
    M = (theta*M1+(1/Dt)*M2);
    MM = (-( 1-theta)*M1old+(1/Dt)*M2old);
    rhs = MM*xold+theta*snew+(1-theta)*sold;
    xnew = M\rhs;                                     % New solution
    X(:,iter) = xnew;                                 % Store the solution
    xold = xnew;
    sold = snew;
    M1old = M1;
    M2old = M2;
end
```

**Listing 4.1:** Theta Method iterative code

As in the harmonic balance chapter, the output of the code is the magnetic vector potential in different time steps, *nDt* in this case. The results that can be found below were checked in Comsol. This code, in case of linear materials, is particularly easy to implement and quick to run (120 seconds for 60 samples of time in half rotation).

In the figure below, the B field is shown. The permanent magnets have a major impact on B, but the contribution of the currents can also be seen in the background.

This figure shows the eddy current induced in the machine. The density of current is rather high in the stator near the airgap, because of the rotating magnetic field induced by the rotation of the rotor. Again, a lower contribution of the three phases can be seen near the coils.

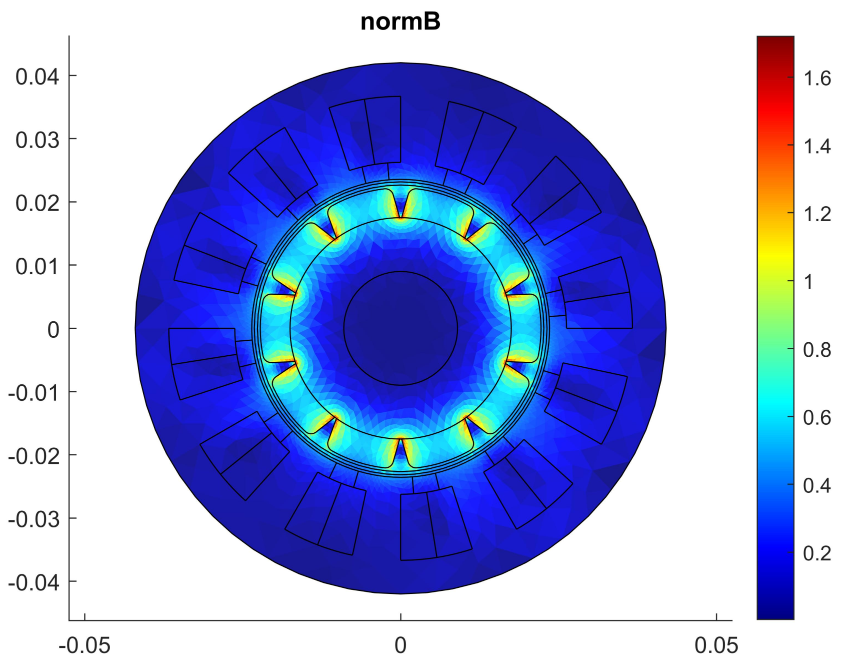


**Figure 4-3:** Norm of the B field.

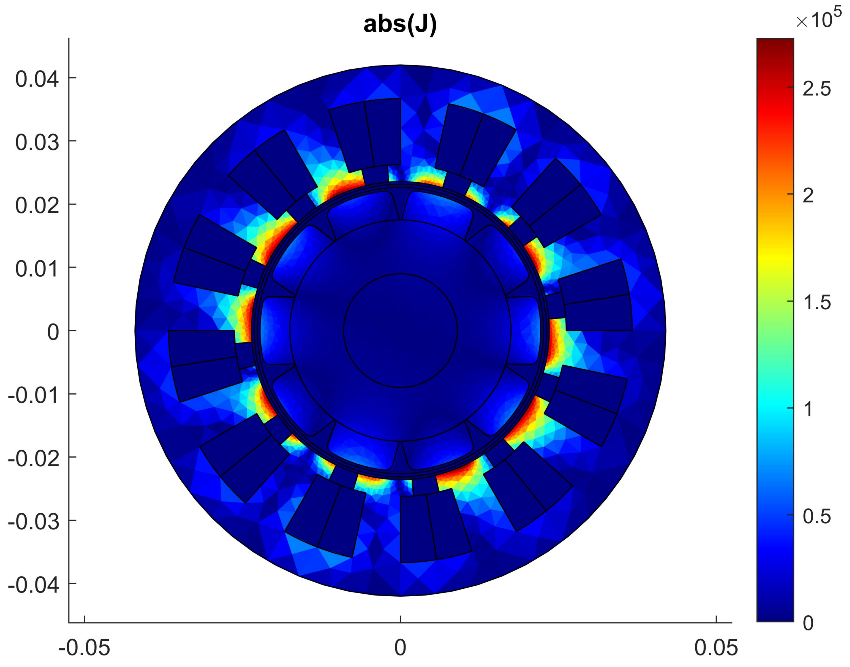


**Figure 4-4:** Abs of the eddy currents.

## 4.4 Time Periodic Solver: Linear Field Problem

The same problem of the previous section is now studied with a different approach, to test it and compare it with the previous one. The aim of this method is to avoid the solution of the field problem at every time step, as it was done in the theta method. Instead, we want to build a big matrix which contains inside all the equations of all the time steps. In this way the solution is found only once, saving time of computation. On the other hand, this method requires to build a very large matrix and to solve an extensive system of equations.

To build the system, we start by recalling the theta method matrices in eq. 4.7 and eq. 4.8. Note that the second equation has a different sign with respect to the equation 4.8, because now it is moved to the left hand side.

$$\mathbf{A} = (\theta \mathbf{K}_{n+1} + \frac{1}{\Delta t}\mathbf{M}_{mass,n+1}) \tag{4.9}$$

$$\mathbf{B} = (1-\theta)\mathbf{K}_{n} - \frac{1}{\Delta t}\mathbf{M}_{mass,n} \tag{4.10}$$

The system to solve is therefore expressed as follows:

$$\begin{cases} \mathbf{A}\mathbf{x}_2 + \mathbf{B}\mathbf{x}_1 = \theta\mathbf{S}_2 + (1-\theta)\mathbf{S}_1 \\ \mathbf{A}\mathbf{x}_3 + \mathbf{B}\mathbf{x}_2 = \theta\mathbf{S}_3 + (1-\theta)\mathbf{S}_2 \\ \quad\vdots \qquad = \qquad \vdots \\ \mathbf{A}\mathbf{x}_N + \mathbf{B}\mathbf{x}_{N-1} = \theta\mathbf{S}_N + (1-\theta)\mathbf{S}_{N-1} \\ \mathbf{A}\mathbf{x}_1 + \mathbf{B}\mathbf{x}_N = \theta\mathbf{S}_1 + (1-\theta)\mathbf{S}_N \end{cases} \tag{4.11}$$

In matrices formulation, the system becomes $\mathbf{Mx} = \mathbf{S}$ where the matrices are:

$$\begin{bmatrix} \mathbf{A} & \dots & & & & \dots & \mathbf{B} \\ \mathbf{B} & \mathbf{A} & \dots & & & & \\ \dots & \mathbf{B} & \mathbf{A} & \dots & & & \\ & & & \dots & & & \\ & & & \dots & \mathbf{B} & \mathbf{A} & \dots \\ & & & & \dots & \mathbf{B} & \mathbf{A} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \vdots \\ \mathbf{x}_{N-1} \\ \mathbf{x}_N \end{bmatrix} = \begin{bmatrix} \theta\mathbf{S}_1 + (1-\theta)\mathbf{S}_N \\ \theta\mathbf{S}_2 + (1-\theta)\mathbf{S}_1 \\ \theta\mathbf{S}_3 + (1-\theta)\mathbf{S}_2 \\ \vdots \\ \theta\mathbf{S}_{N-1} + (1-\theta)\mathbf{S}_{N-2} \\ \theta\mathbf{S}_N + (1-\theta)\mathbf{S}_{N-1} \end{bmatrix} \tag{4.12}$$

In Matlab, the system of matrices can be easily solved by:

$$\mathbf{x} = \mathbf{M}\backslash\mathbf{S} \tag{4.13}$$

It is important to notice that, for this method to work, the problem needs to be periodical. Looking at the first row of the matrix, it is clear that the solution to the last time step needs to be equal to the solution to the first time step, otherwise the problem is no longer periodical and the system looses the sense.

```matlab
        %% Iter 1:nDt-1
disp('---------------------------------------------------------------')
disp('                 Building the matrices of the rotation')
disp('---------------------------------------------------------------')

for iter=1:nDt-1
    P = Pstart;
    % Rotation
        [...]
    % Set of the currents
        [...]
    btot = myNull'*(M2_mat*(newJext));
    % Set of the magnets
        [...]
    snew = myNull.'*(M2x*Hresx + M2y*Hresy) + btot;

    [M1] = fun_stiff_matrix(Ttot,P,mu_fp,myNull);    % Stiffness matrix
    [M2] = fun_mass_matrix(sigma,Ttot,P,myNull);     % Mass matrix
    % Theta method
    A = (theta*M1+(1/Dt)*M2);
    B = ((1-theta)*M1-(1/Dt)*M2);  % B saved for the next time step

    Matrix(((iter-1)*dim+1):(iter*dim),((iter-1)*dim+1):(iter*dim)) = A;
% Writing A on the complete matrix, step N
    Matrix((iter*dim+1):((iter+1)*dim),(((iter-1)*dim+1):(iter*dim))) = B;
% Writing B on the next step

    Stot_n2(((iter-1)*dim+1):(iter*dim)) = snew;              % Sources of the step k+1
    Stot_n1((iter*dim+1):((iter+1)*dim)) = snew;              % Sources of the step k
end

%% Iter for the last step: nDt
    P = Pstart;
       % Rotation
        [...]
    % Set of the currents
        [...]
    btot = myNull'*(M2_mat*(newJext));
    % Set of the magnets
        [...]
    snew = myNull.'*(M2x*Hresx + M2y*Hresy) + btot;

    [M1] = fun_stiff_matrix(Ttot,P,mu_fp,myNull);      % Stiffness matrix
    [M2] = fun_mass_matrix(sigma,Ttot,P,myNull);       % Mass matrix
    % Theta method
    A = (theta*M1+(1/Dt)*M2);
    B = (( 1-theta)*M1-(1/Dt)*M2);  % B saved for the first time step

    Matrix(((nDt-1)*dim+1):(nDt*dim),((nDt-1)*dim+1):(nDt*dim)) = A;
% Writing A on the complete matrix, step N
    Matrix((1:dim),(((nDt-1)*dim+1):(nDt*dim))) = B;
% Writing B on the first step

    Stot_n2(((nDt-1)*dim+1):(nDt*dim)) = snew;       % Sources of the step k+1
    Stot_n1(1:dim) = snew;                           % Sources of the step k

  %% Solutions
disp('---------------------------------------------------------------')
disp('                     Solving the system')
disp('---------------------------------------------------------------')
  Stot = theta.* Stot_n2 + (1-theta).*Stot_n1;     % Matrix of the sources

  Solution = Matrix\Stot;   % Solving the problem
  toc
```

**Listing 4.2:** Building of the matrix of the system

The part of the code shown above is the implementation of the algorithm explained in this chapter in Matlab. As it can be seen, there is a *for* loop in order to build the matrices of the LHS and of the sources. The procedure inside the loop needs to to be repeated one last time outside the loop in order to set in the right position the matrix **B** at the first row. The LHS matrix results as a large sparse square matrix (fig.4-5), which dimensions are $(N_{nodes} \cdot N_{Dt}) \times (N_{nodes} \cdot N_{Dt})$. In this test case it results a $256740 \times 256740$ matrix, but despite the dimensions, the procedure to build the matrix is rather quick and takes about 100 seconds.

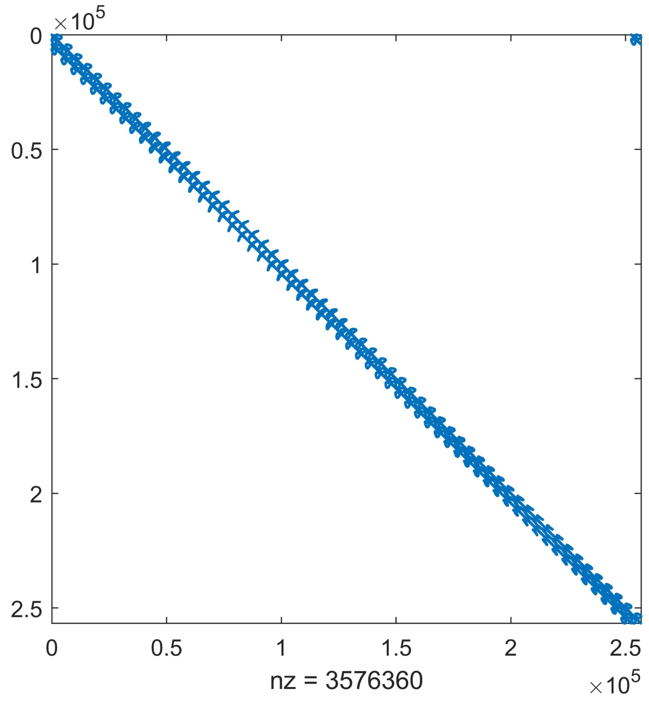


**Figure 4-5:** Matrix of the system: it is a block sparse matrix.

Although this approach seems very similar to the theta method, the computational time of the two is very different. This solver in fact employs about 10 minutes to find the solution because of the big matrices that has to manage. However, it should be noted that no optimization was performed on the code to improve its speediness.

This algorithm is interesting also for another reason: few changes need to be implemented to study a non-linear problem, as it can be seen in the next section.

## 4.5 Time Periodic Solver: Non-linear Field Problem

To introduce nonlinearities in the previous algorithm, the code presented before needs to be slightly changed. A **Newton Raphson** iterative method is added to solve the nonlinear system, but most of the code remains unchanged. Explaining the functioning of the **NR** code is not part of this work, only how to apply it to the problem is now explained. The Matlab's *fsolve* algorithm can be used instead of Newton Raphson, obtaining the same results in a slightly larger time of calculation. The structure of the codes is the same for both solvers and can be found in Appendix C.

The Newton Raphson code asks as input a *handle function* that represents the function to minimize. Inside this function there is the loop with the rotation of the rotor to build the matrices **A** and **B**, and the rows of the code to calculate the magnetic permeability, according to the BH-curve. To improve the speediness of the code, the system is first resolved in a linear case, then this solution is used as a starting point for the NR iterative cycle. In the listing below, the parts of the code where there is the calling to NR, or to the solve algorithm are shown.

```
    %% Solutions of the non-linear system: Newton Raphson
disp('----------------------------------------------------------------')
disp('                     Solving the system with NR                 ')
disp('----------------------------------------------------------------')

  fun = @(arr) myfun_with_jac(H,arr,Told,Pstart,Ntri,rps,nDt,Dt,dim,steps,mu0,IndIron,...
    IndInt,IndExt,IndRot,IndMagN,IndMagS,IndPhasA,IndPhasAn,IndPhasB,IndPhasBn,...
    IndPhasC,IndPhasCn,normH_resid,dni_dB2,myNull,sigma,Jext,f,M2);

xx =  Matrix\Stot;   % first linear solution

options = optimset('TolX',1e-17);                     % set TolX
[xx, resnorm, f, exitflag, output, jacob,resnormstore,xstore] = newtonraphson_mod(fun, xx, options);

% options = optimoptions('fsolve','SpecifyObjectiveGradient',true);
% xx = fsolve(fun,xx,options);     % fsolve as alternative to Newton Raphson

Solution = xx;
```

**Listing 4.3:** The calling to Newton Raphson in the main code.

In the following code, the part of the code that takes into account the saturation is shown. The update of the saturation is made for every time step by using the field computed in the previous time step as input of the BH-curve.

```
    function [ff,jaco] = myfun_with_jac(H,arr,Told,Pstart,Ntri,rps,nDt,Dt,dim,steps,mu0,IndIron,...
    IndInt,IndExt,IndRot,IndMagN,IndMagS,IndPhasA,IndPhasAn,IndPhasB,IndPhasBn,...
    IndPhasC,IndPhasCn,normH_resid,dni_dB2,myNull,sigma,Jext,f,M2)
                [...]
    for iter=1:nDt-1
                [...]
            % Updating the saturation
        a = myNull* arr(((iter-1)*dim+1):(iter*dim));
        Bx = M1x*a;
        By = M1y*a;
        normB = sqrt(Bx.^2 + By.^2);
        normH = normB/mu0;
        normH(IndIron) = H(normB(IndIron));
        mu = normB./normH/mu0;       % Total magnetic permeabiliti

        % Computing the jacobian
        dni_dB2_tot = zeros(Ntri,1);
        dni_dB2_tot(IndIron) = dni_dB2(normB(IndIron).^2);
        [K2jacTOT] = fun_my_grad_grad_jac3(Ttot,P,a,M1x,M1y,dni_dB2,IndIron,dni_dB2_tot);
        K2jac = myNull.'*K2jacTOT*myNull;
        Matrix_Jac(((iter-1)*dim+1):(iter*dim),((iter-1)*dim+1):(iter*dim)) = theta*K2jac;
        Matrix_Jac((iter*dim+1):((iter+1)*dim),(((iter-1)*dim+1):(iter*dim))) = (1-theta)*K2jac;
              [...]
     end
```

**Listing 4.4:** Updating saturation and computing the jacobian.

The problem simulated in this section is much closer to a real synchronous motor than the previous one: the electrical conductivity is set to zero everywhere, except in the magnets. In the iron's region in fact, the induced currents are null because of the lamination of the material and the only region where induced currents can be found are the permanent magnets.

The simulation is carried out in only 24 time steps, that means one every 15 degrees of rotation of the rotor. This low precision is due to the fact that the NR solver is very slow to solve such large matrices and it takes about 55 minutes. The computational time grows exponentially with the growing of the time steps. Despite the problems expressed above, the solution found in Matlab reflects the solution founded in Comsol as well, as can be seen in figure 4-6.

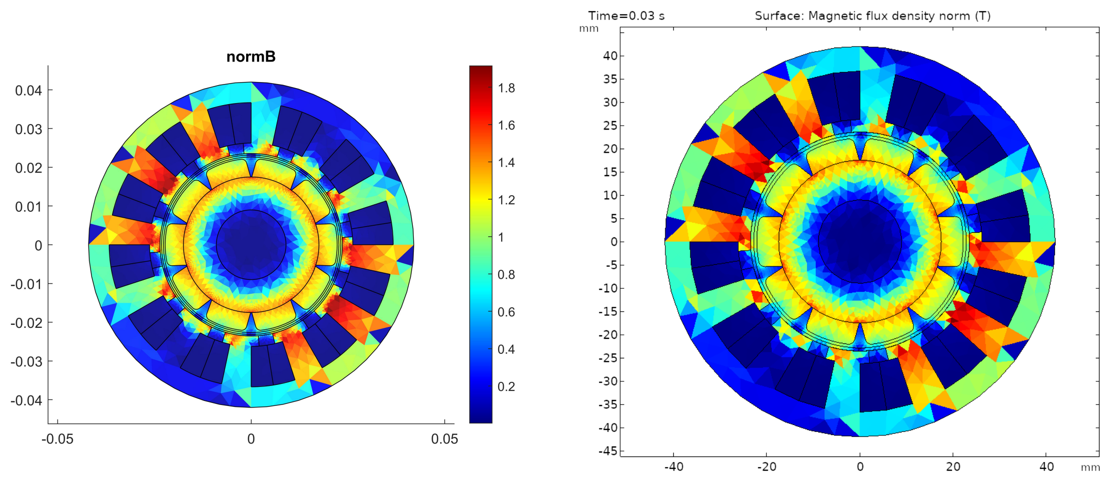


**Figure 4-6:** Comparison between the solution founded in Matlab (left) and Comsol (right).

### 4.5.1 The Problem of Eddy Currents

The eddy currents in the machine are plotted in figure 4-8. As expected the only region of space where there is current is inside the permanent magnets. As known from the theory of the rotating machine, in the permanent magnets, a current is induced by the rotating electromagnetic field. This current creates a loop inside the magnet (figure 4-7), because the magnet itself is isolated from the rest of the motor. The net current, in every time instant, for every section of the magnet perpendicular to the axis of the motor, must to be zero. In this case, because the simulation is 2D and not 3D, the condition above mentioned is not automatically true as can be seen in figure 4-8. A boundary condition has to be imposed in the magnets as proposed in equation 24 of [11] or in [12] to avoid this problem, but at the moment it is not performed in this code. Since machine performance is not measured, eddy currents relatively affect the quality of the code and are currently ignored. These considerations also apply to the next section on the Theta Method.

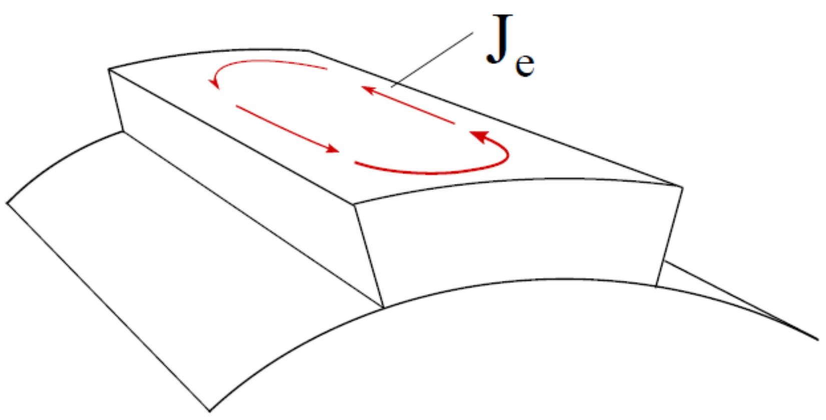


**Figure 4-7:** Loop of induced currents in a permanent magnet.

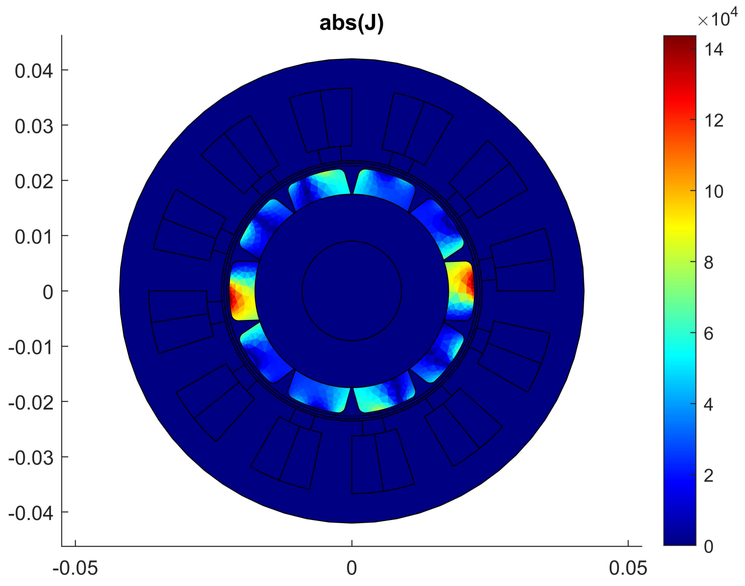


**Figure 4-8:** Induced currents in the permanent magnets.

In conclusion, the algorithm works despite the slowness, and needs some optimization and improvements but it is rather easy to build also in its extension to nonlinear problems.

## 4.6 Theta Method: Nonlinear Field Problems

The extension of the Theta Method to nonlinear problems is slightly more problematic than in the previous case, but the key idea is the same. The nonlinear system is again solved with the Newton Raphson algorithm. The theta method involves finding the solution at each iteration, so the NR code is called at the end of every cycle. This time, the rotation of the rotor and the sets of currents and magnets are performed outside the *handle function*, so some variables, like **M1, M1old** and **xold**, have to be defined as *global* variables (see the listing 4.5). The global variables exist in all the workspaces of Matlab without the need to pass them as inputs of the functions. The whole code can be found in appendix C.5, while below only the calling to Newton Raphson is reported.

The handle function for Newton Raphson performs only the calculation of the Jacobian and the updating of the saturation, then gives as input to NR the function to minimize. To speed up the calculation, two different option are set to the NR code: for the first iteration a very low tolerance is set. The first solution found is the most important because all other solutions depend on the first one. From the second solution to the last, a higher tolerance can be chosen while still having precise solutions. The speediness of the code is therefore slightly improved.

```
   %% Theta Method
global M1old
global sold
global M1
    [...]
for iter=1:nDt
    P = Pstart;
    % Rotation
        [...]
    % Set of the currents
        [...]
    % Set of the magnets
        [...]

      % Solutions of the non-linear system: Newton Raphson
    disp('|----------------------------------------------------------------------|')
    disp(['                      Solving the system with NR, iter = ',num2str(iter)])

      fun = @(arr) myfun_with_jac2(H,arr,xold,Ntri,Ttot,P,M1x,M1y,mu0,IndIron,dni_dB2,...
                           myNull,snew,theta,Dt,sigma,M2);

              if iter ==1
                 options = optimset('TolX',1e-17);
              else
                  options = optimset('TolX',1e-8,'MAXITER',15);
              end
   [xx, resnorm, f_newt, exitflag, output, jacob,resnormstore,xstore] = newtonraphson(fun, xold, options);

    % Updating the global function
    xnew = xx;
    X(:,iter) = xnew;
    xold = xnew;
    M1old = M1;
    sold = snew;
end
```

**Listing 4.5:** Theta method main code.

The solution found is equal to the solution found in the previous section, but this approach to the time domain problem has a strong advantage: the calculation time **linearly** depends on the number of time steps. In the simulation performed, for example, NR takes one minute and a half to find the solution for a time step, so in order to have 60 time steps, the code takes 1 hour and a half to complete the calculus. Once the mesh of the problem is given, it is easy to know the computational time of the solver.

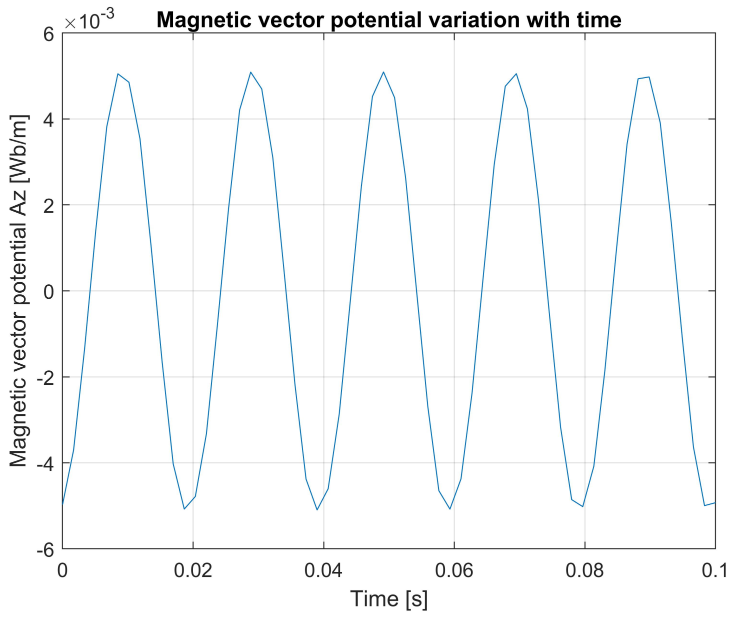


**Figure 4-9:** Magnetic vector potential in a point of the stator.

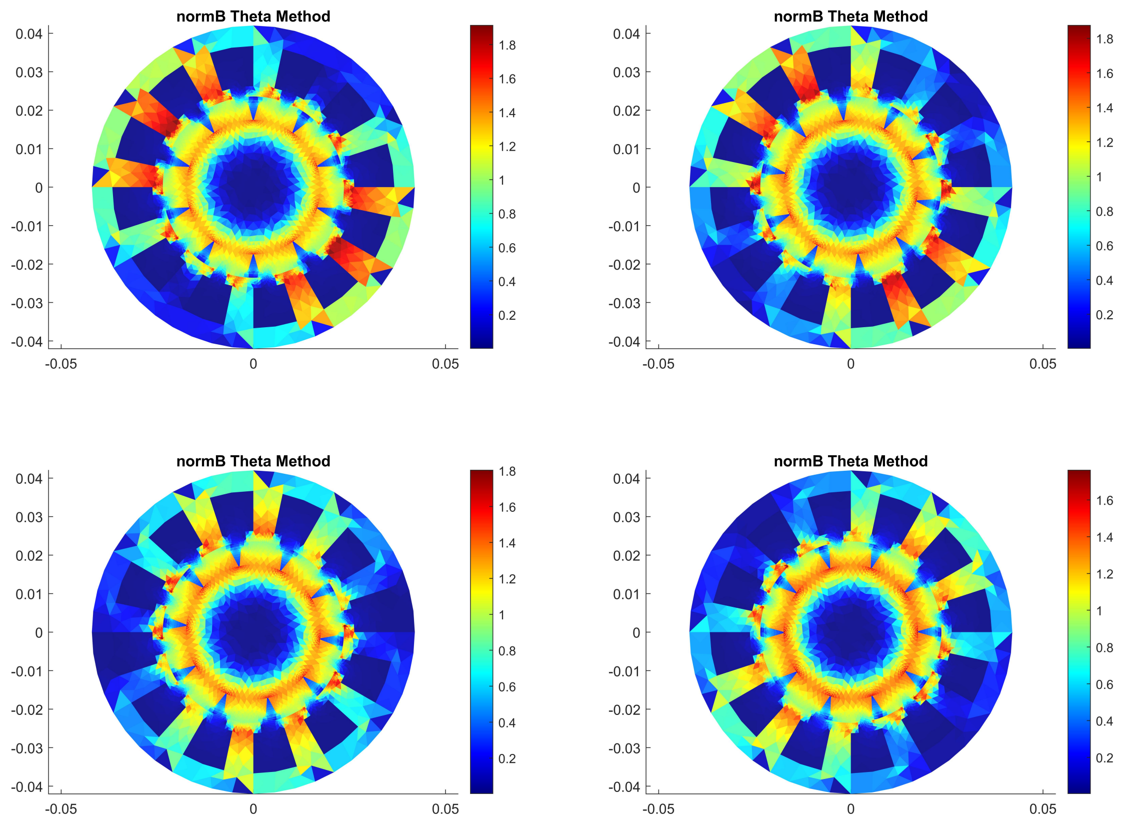


**Figure 4-10:** Magnetic field in four consecutive time steps.

### 4.6.1 Comparison between the two Time Periodic Solvers

In the table below, a brief recap comparing the key aspects of the two outlined algorithms is presented.

| | Theta Method | Time Periodic Solver |
|---|---|---|
| **Calling to NR** | $N_{steps}$ | 1 |
| **Matrices Dimensions** | $N_{node} \times N_{node}$ | $(N_{node} \cdot N_{steps}) \times (N_{node} \cdot N_{steps}) = N_{node}^2 \times N_{steps}^2$ |
| **Calculation time for 1 NR calling** | 1.5 minutes | 55 minutes |
| **Times** | Calculation time linearly dependent on $N_{steps}$ | Calculation time is exponentially dependent on $N_{steps}$ |
| **Other aspects** | | Storage problems because of the sizes of the matrices |

# Chapter 5

# Conclusions

In this thesis, a number of algorithms and codes are covered to analyze increasingly complex problems, starting from electrical circuits, where there is no space distretization, moving to rotating devices in 2D domain. The promising Harmonic Balance Method is introduced and implemented for 2D static devices, while its extension to rotating devices might be an interesting development for future works.

The main subject of the thesis is the study of Rotating electrical machines, carried out with the Theta Method and a Time Periodical method. The complexities of a changing mesh and nonlinear materials are overcome, although new improvements may be implemented in the future, especially regarding the speed of computation. The solutions of the two methods are compared to find the advantages and the disadvantages of each of the two methods, comparing them with a commercial software as COMSOL. All the codes are implemented in MATLAB, trying to make the most of its matrix computing capacity.

This work is meant as a starting point for future developments, with the aim of improving its efficiency and speediness, in order to make it competitive with other softwares. As previously reported, the application of harmonic balance to rotating machines is a technique that must be investigated and which promises interesting results for the steady state solution.

# Appendix A

# Harmonic Balance Code for Nonlinear Circuits

The Matlab code used to study nonlinear circuits with HB is presented in this section. The theoretical aspects are discussed in Chapter 1, while in this appendix the main code and some useful tools needed to study the circuit are reported.

## A.1 Main Code

```
% Main script for non-linear harmonic balance for a half wave rectifier
close all
clear all
clc
tic

% Open the netlist
fid = fopen('netlist.txt');
F = textscan(fid,'%f %f %s %f %s');   % cell of the netlist
fclose(fid);

%% Input
h = 20;                       % Number of harminics: It has to be even
w_o = 20*10^3*2*pi;           % Omega

%% Exitation
Components = char(F{3});              % Components of the netlist
gen = find(Components=='V');
Vs = F{4}(gen,1);                 % Sources from the netlist
V_s = sparse(h+1,1);
V_s(2,1) = Vs;

% Diode
Id = 100*10^-12;
vt = 0.025;
gmin = 10^-12;

%% Admittance Matrix
[Ys] = BuildYs(F,w_o,h,Vs);
[Y12,NL] = buildY12(F,w_o,h);

%% Solving HB

v_t = ones(2*h,1);
v_f = 1/h.*fft(v_t);

it_max = 10000;
lambda = 0.001;
d_lambda = lambda;
i_diodo = zeros(2*h,1);
H = zeros(2*h,2*h);

it = 1;
norma(it) = 1;
while norma(it)>10^-12 && it<it_max

    % Voltage in frequency domain
    v_f = v_f(1:h+1,1);

    % Current in time domain
    i_diodo = Id.*(exp(v_t./vt)-1)+v_t.*gmin;
    di_diodo = Id/vt.*(exp(v_t/vt))+gmin;
    i_t = i_diodo;

    % Current in frequency domain
    i_f = 1/h.*fft(i_t);
    i_f = i_f(1:h+1,1);

    % f in frequency domain
    f = lambda.*Ys*V_s+Y12*v_f+i_f ;
    norma(it+1) = norm(f);

    %  Jacobian in frequency domain
    H = h.*ifft(1/h.*fft(diag(di_diodo)).').';
    H = H(1:h+1,1:h+1);
    J = Ys+H;

    % Update v_f
    v_f = v_f-(J)\f;

    % From v_f to v_t
    v_fc = flipud(conj(v_f(2:h,1)));
    v_f = [v_f ;v_fc];            % Couple of valori real and imag value for Fourier
    v_t = h.*ifft(v_f);
    V(:,it) = v_t;        %  V in time domain

    % New step
    if lambda≤1
      lambda = lambda+d_lambda;
    end

    it = it+1;
end
```

## A.2 Admittance Matrix

```
    function [Y12,NL] = buildY12(F,w,h)
    %% Build the admittance matrix between non linear ports
    % Incident Matrix
    [x,y] = size(F);
    N = [F{1,1},F{1,2}];      % Node matrix
    [l,p] = size(N);
    n = max(max(N));          % Number of nodes

    %% Shortcircuit exitation
    Cr1 = char(F{y-2});
    for i=1:l
        if Cr1(i,1)=='V'
            F{y-1}(i,1) = 0;
            F{y-2}(i,1) = {'R'};
        end
    end

    %% Shortcircuit non linearities
    Cr = char(F{y});
    k = 1;
    for i = 1:l
        if Cr(i,1) == 'N'
           F{y-1}(i,1) = 0;
           F{y-2}(i,1) = {'V'};
           num_NL(k,1) = i;
           k = k+1;
           NL = k-1;
        end
    end

  %% Build the matrices
  T1 = zeros(NL*(h+1),NL);
  Y12 = sparse(h+1,h+1);
  for r = 1:NL
      F{y-1}(num_NL(r,1),1) = 1;

      [A,R,G,L,C,b,M1,M2,RHS] = funcomponenti(F);  % Function of the components for the Tableau analysis

      for k = 0:h
          M12 = M1+1j*k*w*M2;
          x = M12\RHS;
          I = x(1:l,1);
          V = x(l+1:l+1,1);

          T((k+1),:) = I(num_NL);
      end

      T1(:,r) = reshape(T,NL*(h+1),1);
      F{y-1}(num_NL(r,1),1) = 0;
  end

        %% Build the admittance matrix
        for g = 1:NL
            for m = 1:NL
                for nn = 1:h+1
                    Y12(nn+(m-1)*(h+1),nn+(g-1)*(h+1)) = ...
                        - T1(nn+(g-1)*(h+1),m);
                end
            end
        end

 end
```

## A.3 FFT reconstruction

Hereafter, a useful code to reconstruct a signal from the fft coefficients is presented.

```
    function [s,t] = FourierPlot(Y,Tmax,L,M)
        % L = number of samples in fourier
        % M = number of sample to recontruct
        % Y = fft coefficient to reconstruct

        t = linspace(0,Tmax-Tmax/L,L);
        dt=t(2)-t(1);
        Fs = 1/dt;          % Sampling frequency

        %% FOURIER

        f = linspace(0,Fs/2,L/2+1);

        mag = abs(Y/L);
        mag(2:end-1) = mag(2:end-1)*2;
        phas = angle(Y);

        %% Reconstructing
        t = linspace(0,Tmax-Tmax/M,M);
        s = zeros(1,size(t,2));
        for jj=1:L/2+1
            s=s+mag(jj).*cos(2*pi*f(jj)*t+phas(jj));
        end
    end
```

# Appendix B

# Harmonic Balance: Codes for Static Simulation

In this section, some of the Matlab codes implemented for the Harmonic Balance FEM in the static case are reported. The code outlined below comes from the test case of the permanent magnet motor in a locked rotor situation, since it is the most complex and complete test case studied in this work for this kind of situation.

## B.1 Import Mesh Data from Comsol

This script is necessary in order to import the data of the mesh, created in Comsol, into Matlab. The key idea is to set the areas with the same materials with different named materials, so that every geometry can be identified by its material.

```
mphopen PM_motor2.mph    % Open the comsol file
%% mesh
info_mesh = mphxmeshinfo(model);   % Import the comsol mesh data
P = info_mesh.nodes.coords';
T = info_mesh.elements.tri.nodes' + 1;
Ntri=size(T,1)      % Number of triangols
Nnode=size(P,1)     % Number of nodes
xyzb = zeros(length(T(:,1)),3);
for i = 1:length(T(:,1))    % center of gravity of the elements
    xyzb(i,1:2) = (P(T(i,1),1:2)+P(T(i,2),1:2)+P(T(i,3),1:2))/3;
end
mphgeom(model)     % Plot of the geometry
%% Find of the elements of the geometry from their material
id1=fun_find_tri_from_material_numb(model,1,xyzb);
id2=fun_find_tri_from_material_numb(model,2,xyzb);
id3=fun_find_tri_from_material_numb(model,3,xyzb);
id4=fun_find_tri_from_material_numb(model,4,xyzb);
id5=fun_find_tri_from_material_numb(model,5,xyzb);
id6=fun_find_tri_from_material_numb(model,6,xyzb);
id7=fun_find_tri_from_material_numb(model,7,xyzb);
id8=fun_find_tri_from_material_numb(model,8,xyzb);
id9=fun_find_tri_from_material_numb(model,9,xyzb);
id10=fun_find_tri_from_material_numb(model,10,xyzb);

%% Plot of the elements with different materials
figure
hold on
patch('Faces',T(id1,1:3),'Vertices',P,'Facecolor','r','FaceAlpha',0.9)
patch('Faces',T(id2,1:3),'Vertices',P,'Facecolor','b','FaceAlpha',0.9)
patch('Faces',T(id3,1:3),'Vertices',P,'Facecolor','y','FaceAlpha',0.9)
patch('Faces',T(id4,1:3),'Vertices',P,'Facecolor','g','FaceAlpha',0.9)
patch('Faces',T(id5,1:3),'Vertices',P,'Facecolor',[0.9290 0.6940 0.1250],'FaceAlpha',0.9)
patch('Faces',T(id6,1:3),'Vertices',P,'Facecolor',[0.4940 0.1840 0.5560],'FaceAlpha',0.9)
patch('Faces',T(id7,1:3),'Vertices',P,'Facecolor',[0.3010 0.7450 0.9330],'FaceAlpha',0.9)
patch('Faces',T(id8,1:3),'Vertices',P,'Facecolor',[0.9290 0.6940 0.1250],'FaceAlpha',0.9)
patch('Faces',T(id9,1:3),'Vertices',P,'Facecolor',[0.4940 0.1840 0.5560],'FaceAlpha',0.9)
patch('Faces',T(id10,1:3),'Vertices',P,'Facecolor',[0.3010 0.7450 0.9330],'FaceAlpha',0.9)
axis equal
view(2)
%%
IndAir=id1;
IndIron=id2;
IndMagN=id3;
IndMagS=id4;
```

```
IndCoilaP=id5;
IndCoilbP=id6;
IndCoilcP=id7;
IndCoilaM=id8;
IndCoilbM=id9;
IndCoilcM=id10;
save data_PM_motor_2d.mat P T IndIron IndAir  IndMagN  IndMagS  IndCoilaP  IndCoilbP
IndCoilcP  IndCoilaM  IndCoilbM  IndCoilcM
%%
return
```

## B.2 Main Code

```
    %% Script HBFEM permanent magnet motor
clc
close all
clear
tic
%%
cd ..; cd('FEM'); addpath(pwd); cd ..; cd('PM Motor');
main_from_comsol_to_matlab;    % Script to update mesh data 'data_PM_motor_2d'
load('data_PM_motor_2d.mat');

Nh = 30;             % Number of Harmonics: it has to be even
%% Find the center of gravity of the triangles
xyzb = zeros(length(T(:,1)),3);
for i = 1:length(T(:,1))
    xyzb(i,1:2) = (P(T(i,1),1:2)+P(T(i,2),1:2)+P(T(i,3),1:2))/3;
end
%%  Find the boundary nodes
Ntri=size(T,1)
Nnode=size(P,1)
if 1
[g,c]=gcd_tri(double(T.'),Ntri);
Nedge=size(g,2);
Csp=sparse(1:Ntri,abs(c(1,:)),sign(c(1,:)),Ntri,Nedge);
Csp=Csp+sparse(1:Ntri,abs(c(2,:)),sign(c(2,:)),Ntri,Nedge);
Csp=Csp+sparse(1:Ntri,abs(c(3,:)),sign(c(3,:)),Ntri,Nedge);
ind_boundary_edge=find(full(sum(abs(Csp)))==1);
ind_boundary_node=unique([g(1,ind_boundary_edge),g(2,ind_boundary_edge)]);
ind_internal_node=setdiff(1:Nnode,ind_boundary_node);
Nnode_int=length(ind_internal_node); % Ndofs
myNull=sparse(ind_internal_node,1:Nnode_int,...
              ones(Nnode_int,1),Nnode,Nnode_int);
end

%% Build of the matrices to project quantity from nodes to triangles and vice versa
if size(P,2)==2
    P=[P,zeros(size(P,1),1)];
end
[M1x,M1y] = M1_creation_bis_xy(P,T);
[M2x,M2y] = M2_creation_bis_NL(P,T);
[M2] = M2_creation_bis(P,T);
[M_Tn] = M2_creation_Tn(P,T);

%% set of the coil's currents
Jext=5E6;
newJext=zeros(Ntri,1);
newJext(IndCoilaP)=Jext*exp(1j*0);
newJext(IndCoilaM)=-Jext*exp(1j*0);
newJext(IndCoilbP)=Jext*exp(1j*2/3*pi);
newJext(IndCoilbM)=-Jext*exp(1j*2/3*pi);
newJext(IndCoilcP)=Jext*exp(1j*4/3*pi);
newJext(IndCoilcM)=-Jext*exp(1j*4/3*pi);
% plot of the currents
figure
patch('Faces',T,'Vertices',P,'CData',abs(newJext),'Facecolor','flat','FaceAlpha',1.0)
axis equal
```

```
colormap jet
colorbar
title('Jext')
drawnow
% from tri to nodes (build rhs)
btot = myNull'*(M2*(newJext));

%% load BH and interpolation
load BH_comsol.mat
H = fit(BH(:,2),BH(:,1),'linearinterp');

%% Data for HB
f = 50;           % Frequency
omega = 2*pi*f;
IndCond=[IndMagN;IndMagS];             % Indices of conductive materials
sigma = zeros(Ntri,1);
sigma(IndCond) = 8.41e3;

mufp = 5;                 % mufp<2 always converges but it is slow, mufp>2, it is a lot faster.
mu_fp = ones(Ntri,1);     % Number empirically founded for this  specific load conditions
mu_fp(IndIron) = mufp;    % Set of mu_fp in the iron elements
niFP = 1/mufp;
mu0 = 4*pi*10^-7;
nimu0=1/mu0;

%% Set the magnets
normH_resid = 0.5/mu0;      %[T]
newH_res = zeros(Ntri,1);
Hresx = newH_res;
Hresy = Hresx;
newH_res(IndMagN) = normH_resid;
newH_res(IndMagS) = -normH_resid;
% plot of H residual
figure
patch('Faces',T,'Vertices',P,'CData',newH_res,'Facecolor','flat','FaceAlpha',1.0)
axis equal
colormap jet
colorbar
title('normH Residual')
drawnow

%% Set of the vector for the magnet
dist_magN = sqrt(xyzb(IndMagN,1).^2+xyzb(IndMagN,2).^2);  % Distance of COG from (0,0) for every
dist_magS = sqrt(xyzb(IndMagS,1).^2+xyzb(IndMagS,2).^2);  % elements of the magnets N e S

Hresx(IndMagN) = normH_resid*(xyzb(IndMagN,1)./dist_magN);
% Set of the radial direction of the
Hresy(IndMagN) = normH_resid*(xyzb(IndMagN,2)./dist_magN);
% magnetic field for every elements
Hresx(IndMagS) = -normH_resid*(xyzb(IndMagS,1)./dist_magS);
Hresy(IndMagS) = -normH_resid*(xyzb(IndMagS,2)./dist_magS);


mag_res = myNull'*(M2x*Hresx + M2y*Hresy);

%% Initialization HB
Mold = myNull.'*zeros(Nnode,Nh);
rhs_t = zeros(length(btot),Nh);
rhs_f = zeros(length(btot),Nh);
[K_fp] = fun_stiff_matrix(T,P,mu_fp,myNull);      % Stiffness Matrix of the fixed point
[Mass] = fun_mass_matrix(sigma,T,P,myNull);       % ass Matrix

% Scale the rhs to match the Fourier's coefficients
mag_sources = Nh*mag_res;         %  DC term do not need the /2 !
cur_sources = Nh/2*btot;

rhs_f(:,1) = mag_sources;                % DC Component in the RHS
rhs_f(:,2) = cur_sources;                % First harmonic in the RHS
rhs_f(:,end) = conj(rhs_f(:,2));         % Complex conjugate of the first harmonic
sol_f = K_fp\rhs_f;                      % First solution with magnetization = 0
sol_t = ifft(sol_f,[],2,'symmetric');    % First solution in time domain
vec_harm = (0:Nh/2);                     % Vector of the number of the harmonics
```

```
it = 1;
Tmax=20e-3;
t = [0:Tmax/Nh:Tmax-Tmax/Nh];

%% Iterative cycle
disp('---------------------------------------------------------------')
disp('                      HB Iteration')
disp('---------------------------------------------------------------')
disp('Iter  Check')

check = 1;
toll = 1E-6;
it_max = 10000;
convergenza = zeros(it_max,2);

while  check > toll && it < it_max
        % Compute the magnetization for the solution previously obtained
        for m=1:Nh        % Compute magnetization in every time step
            %
            xloc=myNull*sol_t(:,m);
            Bx = M1x*xloc;
            By = M1y*xloc;
            normB = sqrt(Bx.^2 + By.^2);
            [indB] = find(normB < 1E-9);         % Approximation where normB = 0
            normB(indB) = 1e-9;
            normH=normB/mu0;
%            Evaluate H from BH only in iron
            normH(IndIron) = H(normB(IndIron));

%             H bounded to B from BH, with direction
            Hx = normH./normB .* Bx;
            Hy = normH./normB .* By;
%             H bounded to B from niFP (only for iron)
            Hx_fp = nimu0*Bx; % air
            Hy_fp = nimu0*By; % air
            Hx_fp(IndIron) = nimu0*niFP*Bx(IndIron); % iron
            Hy_fp(IndIron) = nimu0*niFP*By(IndIron); % iron

            M(:,m) = myNull'*(M2x*(Hx-Hx_fp) + M2y*(Hy-Hy_fp));   % Minus sign is inside M2y

        end
     check = max(max(sqrt((1/(nimu0*niFP))*full(M-Mold).^2)));
     convergenza(it,1) = it;
     convergenza(it,2) = check;
     Mold = M;
     disp([num2str(it),':  ',num2str(check)])

        % Frequency domain rhs
        rhs_f = -fft(M,[],2);
        rhs_f(:,1) = (mag_sources + rhs_f(:,1));                % DC known term
        rhs_f(:,2) = (cur_sources + rhs_f(:,2));                % First harmonic known term
        rhs_f(:,end) = conj(cur_sources + rhs_f(:,end));        % First harmonic known term

        % Frequency domain solution
        for r = 1:(Nh/2+1)       % For cycle to multiply omega for the harmonic number
            sol_f(:,r) = (K_fp+1j*vec_harm(r)*omega*Mass)\rhs_f(:,r);
        end
        % The second half of sol_f is the complex conj of the first Nh/2
        % elements (without the dc term and the max term)
        sol_f = [sol_f(:,(1:Nh/2+1)),fliplr(conj(sol_f(:,(2:Nh/2))))];

        %Time domain solution
        sol_t=ifft(sol_f,[],2,'symmetric');

        it=it+1;
end
convergenza(it:end,:) = [];
disp('---------------------------------------------------------------')
disp('                      End Iteration')
disp('---------------------------------------------------------------')
   disp(['Numero di armoniche:  ',num2str(Nh)])
toc
```

```
%% Plot of the convergence
figure
semilogy(convergenza(:,1),convergenza(:,2));
title('Convergence')
xlabel('Iteration')
grid on

%% Plot of the fields
xloc=myNull*sol_t(:,1);
Bx = M1x*xloc;
By = M1y*xloc;
normB = sqrt(Bx.^2 + By.^2);
figure
patch('Faces',T,'Vertices',P,'CData',normB,'Facecolor','flat','FaceAlpha',1,'EdgeColor','none')
axis equal
colormap jet
colorbar
title('normB HB')
drawnow

%% Plotting solution
internal_P = P;
internal_P(ind_boundary_node,:)=[];    % Eliminate boundary nodes
p = [-0.015 0.009];
[ind]=find(abs(internal_P(:,1)-p(1))<1e-5 & abs(internal_P(:,2)-p(2))<1e-5 );
figure
plot(t(1:Nh),sol_t(ind,:));
xlabel('t [s]');
ylabel('Az [Wb/m]')
title('Az in P');
grid on

%%   Recontruction of Az in time thanks to the fft
[Az_t,temp] = fft_plot(sol_f(ind,:),Tmax);
figure
hold on
plot(temp,Az_t)
xlabel('t [s]');
ylabel('Az [Wb/m]')
title('Az in P');
grid on


%% Plot of the point of evaluation in the geometry
figure
patch('Faces',T,'Vertices',P,'CData',normB./normB,'Facecolor','flat','FaceAlpha',1.0)
axis equal
colormap jet
colorbar
hold on
plot(internal_P(ind,1),internal_P(ind,2),'rx','markersize',20,'linewidth',2)
drawnow
title('Point of evaluation')
```

## B.3 Other Useful Codes

### B.3.1 [M2x,M2y] = M2_creation_bis_NL(P,T)

Function to project a value of an element on his nodes. The other matrices are built similarly to these.

```
    function [M2x,M2y] = M2_creation_bis_NL(P,T)
n_T = length(T(:,1));
n_n = length(P(:,1));
loc = [1 2 3
    2 3 1
    3 1 2];
% M2 = sparse(n_n,n_T);
```

```
M2bisx=zeros(3,n_n*15);
M2bisy=zeros(3,n_n*15);
triang = zeros(3,3);
count=1;
for i = 1:n_T
    ind = T(i,:);
    for j = 1:3
        triang(1:3,:) = P(T(i,loc(j,:)),:);
        area = fun_my_area_triang(triang);% fun_my_vol_prisma(tetra); % compute triangle area
        area = abs(area); % to be shure int(curl(phi)_x)=Area*curl(phi)_x(computed,in the centre of gravity)
        [valx, valy] = funNonBarnormB(triang);
        M2bisx(1,count)=ind(j);
        M2bisx(2,count)=i;
        M2bisx(3,count)=area*valx;
        M2bisy(1,count)=ind(j);
        M2bisy(2,count)=i;
        M2bisy(3,count)=area*valy;
        count=count+1;
    end
end
M2x = sparse(M2bisx(1,1:count-1),M2bisx(2,1:count-1),M2bisx(3,1:count-1),n_n,n_T);
M2y = sparse(M2bisy(1,1:count-1),M2bisy(2,1:count-1),M2bisy(3,1:count-1),n_n,n_T);

end
```

### B.3.2 [K] = fun_stiff_matrix(T,P,mu,myNull)

Function to compute the the stiffness matrix from the mesh and the materials of the problem.

```
    function [K] = fun_stiff_matrix(T,P,mat,myNull)
% Attention, mat = vector of mu of the domain

Ntri = size(T,1);
Nnode = size(P,1);
ni = 1./(mat*4*pi*1e-7);
loc = [1 2 3
    2 3 1
    3 1 2];
KBIS = zeros(3,Nnode*15);
count = 1;
for ii = 1:Ntri
    triang = P(T(ii,:),:);
    area = fun_my_area_triang(triang);% fun_my_vol_prisma(tetra); % Compute area triangles
    ind = T(ii,:); %extract node indexes
    for hh = 1:3
       [ghh1,ghh2] = funNonBarnormB(P(T(ii,loc(hh,:)),:)); %compute grad. function on centroid
       for kk = 1:3
           [gkk1,gkk2] = funNonBarnormB(P(T(ii,loc(kk,:)),:)); %compute grad. function on centroid
            KBIS(1,count)=ind(hh);
            KBIS(2,count)=ind(kk);
            KBIS(3,count)= area*(ghh1*gkk1+ghh2*gkk2)*ni(ii);
            count = count+1;
       end
    end
end
% K
Ktemp = sparse(KBIS(1,1:count-1),KBIS(2,1:count-1),KBIS(3,1:count-1),Nnode,Nnode);
K = myNull.'*Ktemp*myNull;     % Delete rows and columns of the boundary nodes

end
```

### B.3.3 [Mass] = fun_mass_matrix(sigma,T,P,myNull)

Function to compute the the mass matrix from the mesh and the materials of the problem.

```
    function [Mass] = fun_mass_matrix(sigma,T,P,myNull)
```

```

Ntri = size(T,1);
Nnode = size(P,1);
Mass = sparse(Nnode,Nnode); % initialize the global matrix A

                      %% Start the assembly process
for kk = 1:Ntri
% loop over all elements
    sigmae=sigma(kk);
    triang = P(T(kk,:),:);
    area = fun_my_area_triang(triang);
    Me=(area/12)*(sigmae)*[2 1 1;1 2 1;1 1 2];
    for ii = 1:3         % loop over the local nodes of each element
        ig = T(kk,ii);                              % global node corresponding to ii
        for jj = 1:3                                % loop over the local nodes of each element
            jg = T(kk,jj);                          % global node corresponding to jj
            Mass(ig,jg) = Mass(ig,jg) + Me(ii,jj);    % fill the global matrix
        end
    end
end
Mass = myNull.'*Mass*myNull;    % Deleting boundary rows and columns
end
```

# Appendix C

# Theta Method: Codes for Rotating Machines

In this section, the codes used to study the field problem of a rotating machine in time domain are reported. All the codes refer to the synchronous motor described in chapter 3 and 4.

## C.1 Mesh of the Moving Band

This is the code to mesh the moving band. The input variable "*rotaz*" is a quantity that indicates of how many elements the moving band has to rotate. The transformation from degree to element of rotation is done in the main code and it depends on how many nodes there are in the boundary of the moving band.

```
function [Tri] = renumerate_node(P,IndInt,IndExt,rotaz)
    %% Nodes of the inner circle
    [out_Int,oldind_int] = sort((P(IndInt,1)));
    index_pos_int = find(P(IndInt(oldind_int),2)≥0);
    index_neg_int = flipud(find(P(IndInt(oldind_int),2)<0));
    renum_int = [P(IndInt(oldind_int(index_pos_int)),1),P(IndInt(oldind_int(index_pos_int)),2);...
        P(IndInt(oldind_int(index_neg_int)),1),P(IndInt(oldind_int(index_neg_int)),2)];

    %% Nodes of the outer circle
    [out_Ext,oldind_ext] = sort((P(IndExt,1)));
    index_pos_ext = find(P(IndExt(oldind_ext),2)≥0);
    index_neg_ext = flipud(find(P(IndExt(oldind_ext),2)<0));
    renum_ext = [P(IndExt(oldind_ext(index_pos_ext)),1),P(IndExt(oldind_ext(index_pos_ext)),2);...
        P(IndExt(oldind_ext(index_neg_ext)),1),P(IndExt(oldind_ext(index_neg_ext)),2)];

    %% Meshing the airgap
    Np = length(renum_int);
    Nel = 2*Np;
    Tri = zeros(Nel,3);
    vec1 = [1+rotaz:Np,1:rotaz];
    vec2 = Np+1:Nel;
    vec_par = 2:2:Nel-2;
    vec_disp = 1:2:Nel-1;

    % column 1
    Tri(vec_disp,1) = vec1;
    Tri(vec_par,1) = vec1(2:end);
    Tri(end,1) = vec1(1);

    % column 3
    Tri(vec_disp,2) = vec2;
    Tri(vec_par,2) = vec2(2:end);
    Tri(end,2) = vec2(1);

    % column 2
    Tri(vec_disp,3) = [vec1(2:end),vec1(1)];
    Tri([vec_par,end],3) = vec2;

    %% Local numbering to global numbering
    IndGlobal = [IndInt(oldind_int(index_pos_int));IndInt(oldind_int(index_neg_int));IndExt(oldind_ext(index_p
    Tri = IndGlobal(Tri);

end
```

## C.2 Main Code: Theta Method for Linear Problem

Despite the fact that this code is for the most part similar to the main code reported in the previous section, the code is now reported in all of its parts.

```
    %% Script Theta Method Linear - permanent magnet motor
clc
close all
clear
tic
%%
cd ..; cd('FEM'); addpath(pwd); cd ..; cd('Sync_rotation');
main_from_comsol_to_matlab;    % Script to update the file test.mat
load('data_test.mat');

Told = T;
Pstart = [P,zeros(size(P,1),1)];
numb_gap_nod = length(IndInt);
gradi_per_node = 360/numb_gap_nod;

%% rot check
grad = 0;
P = Pstart;
angle = grad/180*pi;
rotaz = floor(grad/gradi_per_node);
[Tri] = renumerate_node(Pstart,IndInt,IndExt,rotaz);
Ttot = [Told;Tri];
[P] = Rot_Rotation(P,IndRot,angle);
cc=zeros(size(Ttot,1),1);
cc(size(T,1)+1:end)=1;
figure
patch('Faces',Ttot,'Vertices',P,'Cdata',cc,'Facecolor','flat','FaceAlpha',1.0)
hold on
plot(P(1434,1),P(1434,2),'rx','markersize',10,'linewidth',2)
plot(P(1441,1),P(1441,2),'rx','markersize',10,'linewidth',2)
plot(P(168,1),P(168,2),'gx','markersize',10,'linewidth',2)
title('Total mesh')
axis equal

%% Check quiver

L1 = P(T(:,2),:) - P(T(:,1),:);
L2 = P(T(:,1),:) - P(T(:,3),:);
vz = cross(L1,L2);
vz=vz./sqrt(vz(:,1).^2+vz(:,2).^2+vz(:,3).^2);
figure
quiver3(P(T(:,2),1),P(T(:,2),2),P(T(:,2),3),vz(:,1),vz(:,2),vz(:,3),0,'filled')

%%
T = Ttot;
%% Find the center of gravity of the triangles
xyzb = zeros(length(T(:,1)),3);
for i = 1:length(T(:,1))
    xyzb(i,1:2) = (P(T(i,1),1:2)+P(T(i,2),1:2)+P(T(i,3),1:2))/3;
end
%%  Find the boundary nodes
Ntri=size(T,1)
Nnode=size(P,1)
if 1 % lento ma funzia
[g,c]=gcd_tri(double(T.'),Ntri);
Nedge=size(g,2);
Csp=sparse(1:Ntri,abs(c(1,:)),sign(c(1,:)),Ntri,Nedge);
Csp=Csp+sparse(1:Ntri,abs(c(2,:)),sign(c(2,:)),Ntri,Nedge);
Csp=Csp+sparse(1:Ntri,abs(c(3,:)),sign(c(3,:)),Ntri,Nedge);
ind_boundary_edge=find(full(sum(abs(Csp)))==1);
ind_boundary_node=unique([g(1,ind_boundary_edge),g(2,ind_boundary_edge)]);
%
ind_boundary_node=setdiff(ind_boundary_node,IndInt);
ind_boundary_node=setdiff(ind_boundary_node,IndExt);
%
```

```
ind_internal_node=setdiff(1:Nnode,ind_boundary_node);
Nnode_int=length(ind_internal_node); % Ndofs
myNull=sparse(ind_internal_node,1:Nnode_int,...
              ones(Nnode_int,1),Nnode,Nnode_int);
end

%% Build of the matrices to project quantity from nodes to triangles and vice versa
if size(P,2)==2
    P=[P,zeros(size(P,1),1)];
end
[M1x,M1y] = M1_creation_bis_xy(P,T);
[M2x,M2y] = M2_creation_bis_NL(P,T);
[M2_mat] = M2_creation_bis(P,T);
[M_Tn] = M2_creation_Tn(P,T);


%% load BH and interpolation
load BH_comsol.mat
H = fit(BH(:,2),BH(:,1),'linearinterp');

%% Data
f = 50;           % Frequency
omega = 2*pi*f;
Jext = 5e7;
IndIron = [IndIron_stat;IndIron_rot];
IndCond = [IndIron;IndMagN;IndMagS];            % Indices of the conductive materials
sigma = zeros(Ntri,1);
sigma(IndCond) = 57e4;


mu0 = 4*pi*10^-7;
nimu0 = 1/mu0;
mufp = 1;
mu_fp = ones(Ntri,1);
mu_fp(IndIron) = mufp;
niFP = 1/mufp;


%% Set the magnets
normH_resid = 1.47/mu0;      %[T]
newH_res = zeros(Ntri,1);
Hresx = newH_res;
Hresy = Hresx;
newH_res(IndMagN) = normH_resid;
newH_res(IndMagS) = -normH_resid;
% plot della magnetizzazione residua
figure
patch('Faces',T,'Vertices',P,'CData',newH_res,'Facecolor','flat','FaceAlpha',1.0)
axis equal
colormap jet
colorbar
title('normH residuo')
drawnow

%% Set of the vector for the magnet
dist_magN = sqrt(xyzb(IndMagN,1).^2+xyzb(IndMagN,2).^2);  % Distance of COG from (0,0) for every
dist_magS = sqrt(xyzb(IndMagS,1).^2+xyzb(IndMagS,2).^2);  % elements of the magnets N e S

Hresx(IndMagN) = normH_resid*(xyzb(IndMagN,1)./dist_magN);
% Set of the radial direction of the
Hresy(IndMagN) = normH_resid*(xyzb(IndMagN,2)./dist_magN);
% magnetic field for every elements
Hresx(IndMagS) = -normH_resid*(xyzb(IndMagS,1)./dist_magS);
Hresy(IndMagS) = -normH_resid*(xyzb(IndMagS,2)./dist_magS);

mag_res = myNull'*(M2x*Hresx + M2y*Hresy);

%% Speed
p = 5;                  % Polar couples
rps = 2*pi*f/p;         % rad per second
rpm = 60*f/p;           % Round per minute
period = 60/rpm;
nDt = 60;               % Number of time steps
```

```

steps = linspace(0,period/2,nDt);
Dt = steps(2)-steps(1);
%% Theta Method
theta = 0.51; X = [];

[M1] = fun_stiff_matrix(Ttot,P,mu_fp,myNull);
[M2old] = fun_mass_matrix(sigma,Ttot,P,myNull);
M1old = M1;
sold = mag_res;
xold = M1\sold;

% Iterative theta method

for iter=1:nDt
        disp(['iteration = ',num2str(iter)])
    P = Pstart;

    % Rotation
    angle = steps(iter)*rps;   % Angle in radians for the rotor rotation function
    grad = angle/pi*180;       % Angle in degrees that will be divided
    rotaz = floor(grad/gradi_per_node);        % Elements of rotation
    [Tri] = renumerate_node(P,IndInt,IndExt,rotaz);
    [P] = Rot_Rotation(P,IndRot,angle);
    Ttot = [Told;Tri];

    for i = 1:length(Ttot(:,1))
        xyzb(i,1:2) = (P(Ttot(i,1),1:2)+P(Ttot(i,2),1:2)+P(Ttot(i,3),1:2))/3;
    end

                % Set of the currents
        newJext=zeros(length(Ttot(:,1)),1);
        newJext(IndPhasA)=Jext*sin(2*pi*f*steps(iter));
        newJext(IndPhasAn)=-Jext*sin(2*pi*f*steps(iter));
        newJext(IndPhasB)=Jext*sin(2*pi*f*steps(iter)+2/3*pi);
        newJext(IndPhasBn)=-Jext*sin(2*pi*f*steps(iter)+2/3*pi);
        newJext(IndPhasC)=Jext*sin(2*pi*f*steps(iter)+4/3*pi);
        newJext(IndPhasCn)=-Jext*sin(2*pi*f*steps(iter)+4/3*pi);

        % from tri to nodes (build rhs)
        btot = myNull'*(M2_mat*(newJext));

    % Set of the magnets
    dist_magN = sqrt(xyzb(IndMagN,1).^2+xyzb(IndMagN,2).^2);
% Distance of COG from (0,0) for every
    dist_magS = sqrt(xyzb(IndMagS,1).^2+xyzb(IndMagS,2).^2);  % elements of the magnets N e S
    Hresx(IndMagN) = normH_resid*(xyzb(IndMagN,1)./dist_magN);
% Set of the radial direction of the
    Hresy(IndMagN) = normH_resid*(xyzb(IndMagN,2)./dist_magN);
% magnetic field for every elements
    Hresx(IndMagS) = -normH_resid*(xyzb(IndMagS,1)./dist_magS);
    Hresy(IndMagS) = -normH_resid*(xyzb(IndMagS,2)./dist_magS);
    [M1x,M1y] = M1_creation_bis_xy(P,Ttot);
    [M2x,M2y] = M2_creation_bis_NL(P,Ttot);
    snew = myNull.'*(M2x*Hresx + M2y*Hresy)+ btot;



    [M1] = fun_stiff_matrix(Ttot,P,mu_fp,myNull);
    [M2] = fun_mass_matrix(sigma,Ttot,P,myNull);
    % Theta method
    M = (theta*M1+(1/Dt)*M2);
    MM = (-( 1-theta)*M1old+(1/Dt)*M2old);
    rhs = MM*xold+theta*snew+(1-theta)*sold;
    xnew = M\rhs;
    X(:,iter) = xnew;
    xold = xnew;
    sold = snew;
    M1old = M1;
    M2old = M2;
end
```

## C.3 Main Code for the Time Periodic Solver

```matlab
disp('---------------------------------------------------------------')
disp('                 Building the matrices of the rotation         ')
disp('---------------------------------------------------------------')

for iter=1:nDt-1
    P = Pstart;

    % Rotation
    angle = steps(iter)*rps;   % Angle in radians for the rotor rotation function
    grad = angle/pi*180;       % Angle in degrees that will be divided
    rotaz = floor(grad/gradi_per_node);        % Elements of rotation
    [Tri] = renumerate_node(P,IndInt,IndExt,rotaz);
    [P] = Rot_Rotation(P,IndRot,angle);
    Ttot = [Told;Tri];

    for i = 1:length(Ttot(:,1))
        xyzb(i,1:2) = (P(Ttot(i,1),1:2)+P(Ttot(i,2),1:2)+P(Ttot(i,3),1:2))/3;
    end
    % Set of the currents
        [...]
        btot = myNull'*(M2_mat*(newJext));

    % Set of the magnets
        [...]
        snew = myNull.'*(M2x*Hresx + M2y*Hresy) + btot;

    [M1] = fun_stiff_matrix(Ttot,P,mu_fp,myNull);
    [M2] = fun_mass_matrix(sigma,Ttot,P,myNull);
    % Theta method
    A = (theta*M1+(1/Dt)*M2);
    B = ((1-theta)*M1-(1/Dt)*M2);  % B saved for the next steps

    Matrix(((iter-1)*dim+1):(iter*dim),((iter-1)*dim+1):(iter*dim)) = A; % Writing A on the complete matrix, step N
    Matrix((iter*dim+1):((iter+1)*dim),(((iter-1)*dim+1):(iter*dim))) = B; % Writing B on the next step

    Stot_n2(((iter-1)*dim+1):(iter*dim)) = snew;              % S al passo k+1
    Stot_n1((iter*dim+1):((iter+1)*dim)) = snew;              % S al passo k
end
%% Iter nDt
    P = Pstart;
    % Rotation
    angle = steps(iter)*rps;   % Angle in radians for the rotor rotation function
    grad = angle/pi*180;       % Angle in degrees that will be divided
    rotaz = floor(grad/gradi_per_node);        % Elements of rotation
    [Tri] = renumerate_node(P,IndInt,IndExt,rotaz);
    [P] = Rot_Rotation(P,IndRot,angle);
    Ttot = [Told;Tri];

    for i = 1:length(Ttot(:,1))
        xyzb(i,1:2) = (P(Ttot(i,1),1:2)+P(Ttot(i,2),1:2)+P(Ttot(i,3),1:2))/3;
    end

       % Set of the currents
        [...]
        btot = myNull'*(M2_mat*(newJext));

       % Set of the magnets
        [...]
    snew = myNull.'*(M2x*Hresx + M2y*Hresy) + btot;

    [M1] = fun_stiff_matrix(Ttot,P,mu_fp,myNull);
    [M2] = fun_mass_matrix(sigma,Ttot,P,myNull);
    % Theta method
    A = (theta*M1+(1/Dt)*M2);
    B = (( 1-theta)*M1-(1/Dt)*M2);    % B saved for the next steps

    Matrix(((nDt-1)*dim+1):(nDt*dim),((nDt-1)*dim+1):(nDt*dim)) = A;
% Writing A on the complete matrix, step N
    Matrix((1:dim),(((nDt-1)*dim+1):(nDt*dim))) = B;
% Writing B on the next step

    Stot_n2(((nDt-1)*dim+1):(nDt*dim)) = snew;
    Stot_n1(1:dim) = snew;

  %% Solutions
disp('----------------------------------------------------------------')
disp('                     Solving the system                         ')
disp('----------------------------------------------------------------')
  Stot = theta.* Stot_n2 + (1-theta).*Stot_n1;

  Solution = Matrix\Stot;
```

## C.4 Codes for non-linear Time Periodic Solver

In this section only the part of the code that changes to take into account nonlinearities is reported. Instead of the Newton Raphson algorithm (not shown here) the Matlab's function *fsolve* can be easily used without changing the codes.

### C.4.1 Main Code

```
        %% Iter 1:nDt-1

disp('----------------------------------------------------------------')
disp('                      First linear solution')
disp('----------------------------------------------------------------')

   [...]   % Linear time periodic solver non reported here. It is the same of the previus section

  %% Solutions of the non-linear system: Newton Raphson
disp('----------------------------------------------------------------')
disp('                    Solving the system with NR                  ')
disp('----------------------------------------------------------------')

  fun = @(arr) myfun_with_jac(H,arr,Told,Pstart,Ntri,rps,nDt,Dt,dim,steps,mu0,IndIron,...
    IndInt,IndExt,IndRot,IndMagN,IndMagS,IndPhasA,IndPhasAn,IndPhasB,IndPhasBn,...
    IndPhasC,IndPhasCn,normH_resid,dni_dB2,myNull,sigma,Jext,f,M2);

xx =  Matrix\Stot;

        options = optimset('TolX',1e-17);                     % set TolX
        [xx, resnorm, f, exitflag, output, jacob,resnormstore,xstore] = newtonraphson_mod(fun, xx, options);

%         options = optimoptions('fsolve','SpecifyObjectiveGradient',true);
%         xx = fsolve(fun,xx,options);

Solution = xx;
```

### C.4.2 Function: myfun_with_jac

This function is necessary for the functioning of the Newton Raphson algorithm. It has to be introduced like a *handle function* in the main code and it is given as input to the **NR** code. The set of the currents and of the magnet is not shown because it is the same of the previous codes.

```
    function [ff,jaco] = myfun_with_jac(H,arr,Told,Pstart,Ntri,rps,nDt,Dt,dim,steps,mu0,IndIron,...
    IndInt,IndExt,IndRot,IndMagN,IndMagS,IndPhasA,IndPhasAn,IndPhasB,IndPhasBn,...
    IndPhasC,IndPhasCn,normH_resid,dni_dB2,myNull,sigma,Jext,f,M2)


    Matrix = sparse(dim*nDt,dim*nDt);
    Matrix_Jac = sparse(dim*nDt,dim*nDt);
    Stot_n2 = zeros(dim*nDt,1);
    Stot_n1 = Stot_n2;
    theta = 0.51;
                        %% Iterative cycle to build the big matrix
    for iter=1:nDt-1
        P = Pstart;

        % Rotation
        [...]

        for i = 1:length(Ttot(:,1))
            xyzb(i,1:2) = (P(Ttot(i,1),1:2)+P(Ttot(i,2),1:2)+P(Ttot(i,3),1:2))/3;
        end

        % Set of the currents
        [...]
        % from tri to nodes (build rhs)
        btot = myNull'*(M2*(newJext));

        % Set of the magnets
        [...]

        [M1x,M1y] = M1_creation_bis_xy(P,Ttot);
        [M2x,M2y] = M2_creation_bis_NL(P,Ttot);
        snew = myNull.'*(M2x*Hresx + M2y*Hresy) + btot;

        % Updating the saturation
        a = myNull* arr(((iter-1)*dim+1):(iter*dim));
        Bx = M1x*a;
        By = M1y*a;
        normB = sqrt(Bx.^2 + By.^2);
        normH = normB/mu0;
        normH(IndIron) = H(normB(IndIron));
        mu = normB./normH/mu0;       % Total magnetic permeability

        dni_dB2_tot = zeros(Ntri,1);
        dni_dB2_tot(IndIron) = dni_dB2(normB(IndIron).^2);
        [K2jacTOT] = fun_my_grad_grad_jac3(Ttot,P,a,M1x,M1y,dni_dB2,IndIron,dni_dB2_tot);
        K2jac = myNull.'*K2jacTOT*myNull;
        Matrix_Jac(((iter-1)*dim+1):(iter*dim),((iter-1)*dim+1):(iter*dim)) = theta*K2jac;
        Matrix_Jac((iter*dim+1):((iter+1)*dim),(((iter-1)*dim+1):(iter*dim))) = (1-theta)*K2jac;

        [M1] = fun_stiff_matrix(Ttot,P,mu,myNull);
        [M2_t] = fun_mass_matrix(sigma,Ttot,P,myNull);
        % Theta method
        A = (theta*M1+(1/Dt)*M2_t);
        B = ((1-theta)*M1-(1/Dt)*M2_t);  % B viene salvato per il passo successivo

        Matrix(((iter-1)*dim+1):(iter*dim),((iter-1)*dim+1):(iter*dim)) = A;
% Writing A on the complete matrix, step N
        Matrix((iter*dim+1):((iter+1)*dim),(((iter-1)*dim+1):(iter*dim))) = B;
% Writing B on the next step

        Stot_n2(((iter-1)*dim+1):(iter*dim)) = snew;              % S al passo k+1
        Stot_n1((iter*dim+1):((iter+1)*dim)) = snew;              % S al passo k

    end

    %% Iter nDt
    P = Pstart;

       % Rotation
    [...]

    for i = 1:length(Ttot(:,1))
        xyzb(i,1:2) = (P(Ttot(i,1),1:2)+P(Ttot(i,2),1:2)+P(Ttot(i,3),1:2))/3;
    end

            % Set of the currents
        [...]
        % from tri to nodes (build rhs)
        btot = myNull'*(M2*(newJext));

       % Set of the magnets
    [...]
    snew = myNull.'*(M2x*Hresx + M2y*Hresy) + btot;

            % Updating the saturation
        a = myNull* arr(((nDt-1)*dim+1):(nDt*dim));
        Bx = M1x*a;
        By = M1y*a;
        normB = sqrt(Bx.^2 + By.^2);
        normH = normB/mu0;
        normH(IndIron) = H(normB(IndIron));
        mu = normB./normH/mu0;      % Total magnetic permeability

        dni_dB2_tot = zeros(Ntri,1);
        dni_dB2_tot(IndIron) = dni_dB2(normB(IndIron).^2);
        [K2jacTOT] = fun_my_grad_grad_jac3(Ttot,P,a,M1x,M1y,dni_dB2,IndIron,dni_dB2_tot);
        K2jac = myNull.'*K2jacTOT*myNull;
        Matrix_Jac(((nDt-1)*dim+1):(nDt*dim),((nDt-1)*dim+1):(nDt*dim)) = theta*K2jac;
        Matrix_Jac((1:dim),(((nDt-1)*dim+1):(nDt*dim))) = (1-theta)*K2jac;

    [M1] = fun_stiff_matrix(Ttot,P,mu,myNull);
    [M2_t] = fun_mass_matrix(sigma,Ttot,P,myNull);
    % Theta method
    A = (theta*M1+(1/Dt)*M2_t);
    B = (( 1-theta)*M1-(1/Dt)*M2_t);

    Matrix(((nDt-1)*dim+1):(nDt*dim),((nDt-1)*dim+1):(nDt*dim)) = A;
% Writing A on the complete matrix, step N
    Matrix((1:dim),(((nDt-1)*dim+1):(nDt*dim))) = B;
% Writing B on the next step

    Stot_n2(((nDt-1)*dim+1):(nDt*dim)) = snew;
    Stot_n1(1:dim) = snew;

%% jacobian
    Stot = theta.* Stot_n2 + (1-theta).*Stot_n1;
    ff = Matrix*arr-Stot;
    % jacobian assemble
    jaco = Matrix + Matrix_Jac;
end
```

## C.5 Theta Method: Non-linear problems

In this section the complete code for the nonlinear problem, studied with the theta method, is found.

### C.5.1 Main Code

```
    %% Theta Method for nonlinear problems
theta = 0.51; X = [];
global M1old
global sold
global M1
[M1] = fun_stiff_matrix(Ttot,P,mu_fp,myNull);
[M2] = fun_mass_matrix(sigma,Ttot,P,myNull);
dim = length(M1);
M1old = M1;
sold = mag_res + btot;
xold = M1\sold;
    % Plot of the linear solution used as starting point for the method
    xloc = myNull*xold;
    Bx = M1x*xloc;
    By = M1y*xloc;
    normB = sqrt(Bx.^2 + By.^2);
    figure
    patch('Faces',Ttot,'Vertices',P,'CData',normB,'Facecolor','flat','FaceAlpha',0.9,'EdgeColor','none')
    axis equal
    colormap jet
    colorbar
    title('normB as starting point for the theta method')
    drawnow


for iter=1:nDt
    P = Pstart;
    % Rotation
    [...]
    for i = 1:length(Ttot(:,1))
        xyzb(i,1:2) = (P(Ttot(i,1),1:2)+P(Ttot(i,2),1:2)+P(Ttot(i,3),1:2))/3;
    end

        % Set of the currents
            newJext=zeros(length(Ttot(:,1)),1);
        newJext(IndPhasA)=Jext*sin(2*pi*f*steps(iter));
        newJext(IndPhasAn)=-Jext*sin(2*pi*f*steps(iter));
        newJext(IndPhasB)=Jext*sin(2*pi*f*steps(iter)+2/3*pi);
        newJext(IndPhasBn)=-Jext*sin(2*pi*f*steps(iter)+2/3*pi);
        newJext(IndPhasC)=Jext*sin(2*pi*f*steps(iter)+4/3*pi);
        newJext(IndPhasCn)=-Jext*sin(2*pi*f*steps(iter)+4/3*pi);

                % from tri to nodes (build rhs)
        btot = myNull'*(M2_mat*(newJext));

    % Set of the magnets
    dist_magN = sqrt(xyzb(IndMagN,1).^2+xyzb(IndMagN,2).^2);
    dist_magS = sqrt(xyzb(IndMagS,1).^2+xyzb(IndMagS,2).^2);
    Hresx(IndMagN) = normH_resid*(xyzb(IndMagN,1)./dist_magN);
    Hresy(IndMagN) = normH_resid*(xyzb(IndMagN,2)./dist_magN);
    Hresx(IndMagS) = -normH_resid*(xyzb(IndMagS,1)./dist_magS);
    Hresy(IndMagS) = -normH_resid*(xyzb(IndMagS,2)./dist_magS);
    [M1x,M1y] = M1_creation_bis_xy(P,Ttot);
    [M2x,M2y] = M2_creation_bis_NL(P,Ttot);
    snew = myNull.'*(M2x*Hresx + M2y*Hresy) + btot;

      % Solutions of the non-linear system: Newton Raphson
    disp('|----------------------------------------------------------------------|')
    disp(['                     Solving the system with NR, iter = ',num2str(iter)])

      fun = @(arr) myfun_with_jac2(H,arr,xold,Ntri,Ttot,P,M1x,M1y,mu0,IndIron,dni_dB2,...
                          myNull,snew,theta,Dt,sigma,M2);
              if iter ==1
                 options = optimset('TolX',1e-17);
              else
                  options = optimset('TolX',1e-8,'MAXITER',20);
              end
    [xx, resnorm, f_newt, exitflag, output, jacob,resnormstore,xstore] = newtonraphson(fun, xold, options);

    xnew = xx;
    X(:,iter) = xnew;
    xold = xnew;
    M1old = M1;
    sold = snew;
end
toc
```

### C.5.2 myfun_with_jac2

The handle function for the Newton Raphson script. It is different from the previous one but the functioning is the same.

```
    function [ff,jaco] = myfun_with_jac2(H,arr,arrold,Ntri,T,P,M1x,M1y,mu0,IndIron,dni_dB2,...
    myNull,snew,theta,Dt,sigma,M2)
    global M1old
    global sold
    global M1

    % Update the saturation
    a = myNull*arr;
    Bx = M1x*a;
    By = M1y*a;
    normB = sqrt(Bx.^2 + By.^2);
    normH = normB/mu0;
    normH(IndIron) = H(normB(IndIron));
    ni=normH./normB;
    ni(isnan(ni))=1;
    mu = 1./ni./mu0;

    [M1] = fun_stiff_matrix(T,P,mu,myNull);   % Stiffness matrix
    % Theta method
    M = (theta*M1+(1/Dt)*M2);
    MM = (-( 1-theta)*M1old+(1/Dt)*M2);
    % Function to minimize
    ff = M*arr - MM*arrold - theta*snew - (1-theta)*sold;
    % Jacobian
    dni_dB2_tot = zeros(Ntri,1);
    dni_dB2_tot(IndIron) = dni_dB2(normB(IndIron).^2);
    [K2jacTOT] = fun_my_grad_grad_jac3(T,P,a,M1x,M1y,dni_dB2,IndIron,dni_dB2_tot);
    K2jac = myNull.'*K2jacTOT*myNull;
    jaco = M+K2jac*theta;
end
```

# Bibliography

[1] L. Cesari, “Functional analysis and periodic solutions of nonlinear differential equations,” *Contributions to differential equations*, vol. 1, pp. 149–187, 1963.

[2] M. Urabe, “Galerkin’s procedure for nonlinear periodic systems,” tech. rep., WISCONSIN UNIV MADISON MATHEMATICS RESEARCH CENTER, 1964.

[3] S. Yamada and K. Bessho, “Harmonic field calculation by the combination of finite element analysis and harmonic balance method,” *IEEE Transactions on Magnetics*, vol. 24, no. 6, pp. 2588–2590, 1988.

[4] J. Gyselinck, P. Dular, C. Geuzaine, and W. Legros, “Harmonic-balance finite-element modeling of electromagnetic devices: a novel approach,” *IEEE Transactions on Magnetics*, vol. 38, no. 2, pp. 521–524, 2002.

[5] S. A. Maas, *Nonlinear microwave and RF circuits*. Artech house, 2003.

[6] F. Bachinger, U. Langer, and J. Schöberl, “Efficient solvers for nonlinear time-periodic eddy current problems,” *Computing and Visualization in Science*, vol. 9, no. 4, pp. 197–207, 2006.

[7] O. Bíró, G. Koczka, and K. Preis, “Finite element solution of nonlinear eddy current problems with periodic excitation and its industrial applications,” *Applied Numerical Mathematics*, vol. 79, pp. 3–17, 2014.

[8] S. Außerhofer, O. Biro, and K. Preis, “A strategy to improve the convergence of the fixed-point method for nonlinear eddy current problems,” *IEEE transactions on magnetics*, vol. 44, no. 6, pp. 1282–1285, 2008.

[9] G. Koczka, S. Auberhofer, O. Biro, and K. Preis, “Optimal convergence of the fixed-point method for nonlinear eddy current problems,” *IEEE Transactions on Magnetics*, vol. 45, no. 3, pp. 948–951, 2009.

[10] J. Gyselinck, L. Vandevelde, P. Dular, C. Geuzaine, and W. Legros, “A general method for the frequency domain fe modeling of rotating electromagnetic devices,” *IEEE Transactions on Magnetics*, vol. 39, no. 3, pp. 1147–1150, 2003.

[11] F. Dubas and A. Rahideh, “Two-dimensional analytical permanent-magnet eddy-current loss calculations in slotless pmsm equipped with surface-inset magnets,” *IEEE Transactions on Magnetics*, vol. 50, no. 3, pp. 54–73, 2014.

[12] M. Filippini, “Magnetic gears numerical modelling and optimization,” 2019.

[13] H. De Gersem, S. Vandewalle, and K. Hameyer, “Krylov subspace methods for harmonic balanced finite element methods,” in *Scientific Computing in Electrical Engineering*, pp. 387–396, Springer, 2001.

[14] J. D. García-Saldaña and A. Gasull, “A theoretical basis for the harmonic balance method,” *Journal of Differential Equations*, vol. 254, no. 1, pp. 67–80, 2013.

[15] N. Bianchi, *Calcolo delle Macchine Elettriche col Metodo degli Elementi Finiti*. Cleup, 2001.

[16] G. Gambolati and M. Ferronato, *Lezioni di Metodi Numerici per l'Ingegneria*. Progetto, 2014.