ESCOLA TÈCNICA SUPERIOR D'ENGINYERS DE
TELECOMUNICACIÓ DE BARCELONA

PROJECTE FINAL DE CARRERA

Integral Equation MEI (IE-MEI) analysis of 2D homogeneous lossy dielectric and plasmonic scatterers

AUTHOR: Gerard Planes Conangla
DIRECTOR: Juan Manuel Rius Casals

Departament de Teoria de Senyal i Comunicacions
AntennaLab - Grup d'Antenes i Sistemes Ràdio

Barcelona, 2015

Acknowledgements

To Juan Manuel Rius, for his help and guidance and for introducing me to research with this project.

Resum

Aquest treball revisita el mètode IE-MEI, aplicat fins ara únicament a conductors perfectes, i proposa una nova formulació per a dielèctrics d'altes pèrdues en medis homogenis a trossos. L'objectiu és determinar si la ràpida atenuació dels camps en medis amb pèrdues ajuda a millorar el rendiment del IE-MEI, problemàtic en dielèctrics sense pèrdues, amb l'objectiu de reduir els requeriments de memòria per a emmagatzemar la matriu del sistema d'equacions, resultant de la discretització de les equacions integrals de l'electromagnetisme amb el mètode dels moments. Els resultats són prometedors i la dispersió en les matrius obtingudes és excel·lent, si bé el problema d'obtenció de matrius mal condicionades és comú i resta per solucionar.

Resumen

Este trabajo revisita el método IE-MEI, aplicado hasta ahora únicamente en conductores perfectos, y propone una nueva formulación para dieléctricos de altas pérdidas en medios homogéneos a trozos. El objetivo es determinar si la rápida atenuación de los campos en medios con altas pérdidas ayuda a mejorar el rendimiento del IE-MEI, problemático en dieléctricos sin pérdidas, con el objetivo de reducir los requerimientos de memoria para almacenar la matriz del sistema de ecuaciones, resultante de discretizar las ecuaciones integrales del electromagnetismo con el método de los momentos. Los resultados son prometedores y la dispersión en las matrices obtenidas es excelente, si bien el problema de obtención de matrices mal condicionadas es común y todavía no se ha solucionado.

Abstract

This work revisits the IE-MEI method, previously only applied to perfect conductors, and proposes a new formulation for high-loss dielectric piecewise-homogeneous media. The aim is to assess if the fast attenuation of fields in highly lossy media helps in improving the previously not successful performance of the IE-MEI for lossless dielectric scatterers, with the goal of reducing the memory requirements to store the matrix of the system of equations, obtained after discretization with the method of moments of the integral equations of electromagnetism. The results show promise and the obtained sparsification is usually excellent, but the problem of ill-conditioned matrices is common and has yet to be solved.

Contents

1 Introduction **5**

1.1 Motivation 5

1.2 Overview of the methods for electromagnetic problems 7

1.2.1 Numerical approach to the solution 7

1.2.2 Classification of the numerical methods 8

1.3 Goals 9

2 Method of moments **12**

2.1 Surface equivalence 12

2.2 Integro-differential equations 14

2.3 Boundary conditions 14

2.4 Equations for a penetrable dielectric scatterer 15

2.5 Discretization with the MoM 16

3 The MEI method **19**

3.1 Finite differences 19

3.2 Differential MEI 21

3.2.1 The postulates 21

3.2.2 Metrons 22

3.2.3 Consequences and correctness of the postulates 23

4 Integral formulation of the MEI (IE-MEI) **24**

4.1 Derivation from the reciprocity theorem 24

4.2 Derivation as a CFIE from the MoM 26

4.2.1 EFIE and MFIE for PEC 26

4.2.2 Linear combination and equivalence 27

4.3 Coefficient computation 28

4.4 Metrons . . . 29
4.4.1 Coefficient computation . . . 29
4.4.2 Types of metrons . . . 30

5 Implementation of the code in the 2D case 32
5.1 Development and routines . . . 33
5.1.1 MoM for 2D dielectric scatterer . . . 33
5.1.2 MoM in 2Nx2N and NxN form . . . 35
5.1.3 IE-MEI implementation . . . 36
5.1.4 Reduced zero forcing for the IE-MEI . . . 39
5.1.5 Green's function integral in field computation . . . 41
5.1.6 2D Geometry section . . . 42
5.2 Outline of the code . . . 43

6 Graphical user interface 45
6.1 2D Scattering simulator . . . 45
6.1.1 Description of the GUI . . . 45
6.1.2 A guided example . . . 48
6.2 IE-MEI NxN GUI . . . 49
6.2.1 Description of the GUI . . . 49
6.2.2 A guided example . . . 51

7 Results 53
7.1 General considerations . . . 53
7.2 2Nx2N IE-MEI . . . 56
7.3 NxN IE-MEI . . . 57
7.4 Zero-forcing schemes figures . . . 59
7.5 MoM vs IE-MEI figures . . . 66

8 Conclusions 78

9 Annexes 81
9.1 List of publications . . . 81
9.2 Review of electromagnetism . . . 81
9.2.1 Maxwell's equations . . . 81
9.2.2 Wave equation . . . 82

9.2.3 Boundary conditions between media 83
9.2.4 Green's functions 83
9.3 Code 85
9.3.1 Main MATLAB routines 85
9.3.2 Geometry routines 129
9.3.3 Others 133

Chapter 1

Introduction

1.1 Motivation

Numerical analysis, as opposed to general symbolic manipulations, is the design and analysis of techniques to give approximate but accurate solutions to mathematical problems. Algebraic and symbolic manipulations, and thus exact solutions to problems, were insufficient with the advent of hard problems (usually physical models via ordinary and partial differential equations), and in broad terms the study of these equations subdivided in qualitative and numerical analysis: the goal of the first was to obtain global, regularity and convergence results for a given equation; numerical analysis, on the other hand, served as a means to obtain particular solutions, if approximate, to specific problems, and hence became a fundamental tool in applied mathematics and engineering.

With the advent of computing and the automatization of calculations that were traditionally performed manually, the field of numerical methods experienced an authentic revolution, specially from the second half of the twentieth century on. Fields as diverse as weather forecasting, finances and investment, internet queueing, chemical reactions in the farmaceutical industry or antenna and radar theory have benefited immensely from numerical methods.

Scattering problems (scattering of incident light by and object) are a subfield of the latter, and can be modelled with Maxwell's equations (therefore forming a system of partial differential equations). Electromagnetic waves are one of the best known and most commonly encountered forms of radiation that undergo scattering, so the study of light and radio waves behaviour is of fundamental importance in telecommunications. Major forms of elastic light scattering (involving negligible energy transfer) are Rayleigh

and Mie scattering. This models are used to explain, for instance, the atmospheric and cloud scattering. For modelling of scattering in cases where the Rayleigh and Mie models do not apply (such as irregularly shaped particles), there are many numerical methods that can be used.

Recently, surface plasmon polaritons (SPPs) have been brought into light for the possibility of innovative technological implementations. SPPs are electromagnetic waves confined to a metal-dielectric interface, so they are excitations that can be explained, essentially, from a classical point of view, and therefore analysed with Maxwell's equations. As shown in recent experiments, SPPs can be manipulated using waveguides and resonators, and hence hold promise for very diverse applications (sensing, photovoltaics, telecommunications, opto-electronic circuit integration...), due to their ability to concentrate and guide electromagnetic energy at the nanoscale. As such, there is a huge interest in the performance of the simulations involving plasmonic resonances.

As will be explained later in more detail, a problem of interest is the simulation of colloidal depositions of a huge number of small "particles", each of which acting as a plasmonic resonator. These particles can be spheres, nanorods or have other, more exotic, geometries, and the general scattering problem is solved by the method of moments (MoM) in conjunction with the multilevel fast multipole method (MLFMM). The MoM is a numerical computational method for solving linear partial differential equations which have been formulated as integral equations, via discretization in a mesh of boundary elements; the MLFMM is also based on the MoM, but reduces the scaling with respect to memory and run-time from $\mathcal{O}(n^3)$ to $\mathcal{O}(n \log n)$, where n represents the number of unknowns. This method subdivides the boundary element mesh into different clusters and if two clusters are in each other's far field, all calculations that would have to be made for every pair of nodes can be reduced to the midpoints of the clusters with almost no loss of accuracy. Thus, MLFMM introduces different levels of clustering (clusters made out of smaller clusters) to additionally enhance computation speed. However, for clusters not in the far field, the traditional MoM has to be applied.

Although for conventional scatterers the MLFMM can solve large problems of more than one billion (10^9) unknowns, for plasmonic scatterers the discretization must be much finer, thus increasing the size of near field matrices and leading to a possible collapse of the available storage memory. This is why obtaining a solution for the near field, if n is big enough, can increase the time and memory requirements to a point where the problem is not solvable in practice. And this is precisely where the IE-MEI (Integral equation

of the Measured Equation of Invariance) can make a difference, because it is aimed at sparsifying the near field matrices. The main goal of the project has been the application and analysis of the IE-MEI to homogeneous dielectric scatterers with complex relative permittivities ϵ_r, thus being able to represent a broad class of materials (metals, lossy dielectrics and plasmonic materials).

1.2 Overview of the methods for electromagnetic problems

1.2.1 Numerical approach to the solution

Some of the most common numerical methods for electromagnetic problems are finite-element methods and the method of moments, which solve Maxwell's equations to find the distribution of the scattered electromagnetic field by discretizing the geometry of the scatterer (meshing all the scatterer, in the former, or only the boundary, in the latter). To apply any numerical method, the sequence of actions is always the following:

1. *Formulation of the problem*: the starting point are the Maxwell's equations, the wave equation or some derived integro-differential equations.

2. *Discretization of the mathematical formulation*: computations can only be performed with numbers, so the functions have to be discretized to vectors in some (finite) basis, to which the integro-differential operators will be applied. This way, a system of functional equations is transformed into a system of algebraic equations of finite dimension, which can be solved with a computer. Usually, as in Maxwell's equations case, the operators are linear and the resulting system is of the matrix form.

3. *Discretization of the boundary conditions*: the surface must be discretized except in a few canonical geometries. Then, boundary conditions in the elements must be imposed. This can be performed in several ways:

   (a) Imposing boundary conditions in exactly a finite number of points (point-matching).

(b) Making some measure of the error (usually quadratic error) minimal:

$$\min\left(\int |CC(\vec{r}_m)_{\text{calc}} - CC(\vec{r}_m)_{\text{theor}}|^2 \mathrm{d}s\right) \tag{1.1}$$

(c) Making zero the residual with some weight functions W_m

$$\int (CC(\vec{r})_{\text{calc}} - CC(\vec{r})_{\text{theor}}) \cdot W_m(\vec{r}) \mathrm{d}s \tag{1.2}$$

It is the base of the weighted residuals or *moments*. It is a more general method than the other two.

4. It is also necessary to assure that the Sommerfeld radiation condition is met. This condition imposes a restriction to the fields, making them propagate from the source to the exterior, and not to the inside of it, and conserving the power in the wavefront. This is achieved by one of several indirect methods:

   (a) Formulation of the radiated fields via convolution of the currents with Green functions.

   (b) Closing the discretized space with a surface with special boundary radiation/absorption conditions.

   (c) Making the solution behave as some fields radiated by arbitrary currents existing inside the discretized space (as in the MEI method).

1.2.2 Classification of the numerical methods

Numerical methods are usually classified by the electromagnetic equations they solve. There are two families big of methods

1. *Time domain methods (TD)*: temporal formulation of the electromagnetic problem.
2. *Frequency domain methods (FD)*: formulation in terms of phasors of the steady state.

Depending on the type of electromagnetic equations we can also differentiate between:

1. *Differential methods*: they discretize directly Maxwell's equations or the wave equation, representing the derivatives with finite differences. This discretization of the derivatives and functions leads to a linear algebraic system, in which a matrix has to be inverted to solve the problem. To make sure the Sommerfeld radiation conditions is met in open problems, some further constraints in the boundary are required.

2. *Integral methods*: they are based on the equivalence theorem for the calculus of the field in the exterior or interior of the object, by using the integral of radiation of the equivalent currents and the appropriate boundary conditions. If some surface geometry is canonical, a Green function can be used in the integral. This assures that the solution fulfills the radiation and boundary conditions and, therefore, it is only necessary to discretize and impose the boundary conditions to the other surfaces.

   We can distinguish between the method of moments or the conjugated gradient method.

3. *Variational methods*: based on the definition of a stationary functional, a scalar function of the fields with a stationary point when the fields are the solution of the problem. A typical example would be a minimum energy system.

4. *High frequency methods*: the computational cost (in memory and time requirements) required for a numerical method increases enormously with the frequency, as the discretization is performed at constant electric size (from $\lambda/10$ in metallic scatterers to much smaller fractions in plasmonic resonators) and therefore is directly proportional. This imposes a practical upper limit to the electric size of the structures that can be analysed with the available computing power. For this reason, a number of valid approximations in the region of high frequency have been developed, that allow the analysis of big structures.

1.3 Goals

Scattering from homogeneous dielectric media can be numerically modelled either with finite difference or with integral equations methods. The former needs the discretization of the whole domain under analysis, but leads to a sparse linear system that can be stored and solved very efficiently. The latter discretizes only the domain boundary, resulting in a linear system with a much lower number of unknowns (N), but having a full matrix that requires high computational resources of order N^2 and N^3 for storage and solution, respectively.

In the last 20 years there has been intensive research to accelerate the solution of the full linear system resulting from the integral equations discretized by the MoM. The most successful methods are based on iterative solvers and matrix compression schemes

(such as the MLFMM previously discussed). An alternative approach, or complementary in the case of near field, was explored in the early 90s: the possibility of sparsifying the full IE-MoM matrix in order to achieve the best of the IE methods (boundary discretization) together with the best of finite difference methods (sparse linear system matrix). The main idea was to use sets of basis and testing functions that radiate narrow beams and thus produce impedance matrices containing many small elements, which could be dropped to zero. This techniques includes, for instance, the Impedance Matrix Localization (IML) method and the wavelets expansions. Later, the Integral Equation formulation of the Measured Equation of Invariance (IE-MEI) was developed as an integral equation discretization scheme for perfectly conducting (PEC) boundaries that, using the Measured Equation of Invariance (MEI) concept, produced a sparse linear system for convex scatterers, which additionally allowed frequency extrapolation of the matrix coefficients.

The IE-MEI is in fact a special case of the Combined Field Integral Equation (CFIE) discretized by the Method of Moments (MoM), in which the choice of different testing functions for the electric and/or the magnetic fields results in an approximately sparse linear system to solve for the induced current, where most of the matrix elements can be neglected. These new testing functions are numerically derived by solving small linear systems as in the MEI method. A significant feature of numerically derived testing functions is that they are adaptive, i.e., specific to the particular shape of the scatterer boundary and to the location of the function in the boundary. The computational cost and storage requirements of IE-MEI for 2D perfectly conducting scatterers of arbitrary convex shape are of an order between $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$: the linear system to invert has a band cyclic matrix of very small bandwidth independent from n, that can be stored and solved in $\mathcal{O}(n)$ memory and time in the best case. The IE-MEI matrix coefficients can be computed at a frequency lower than the operating frequency and then extrapolated to the higher frequency, so that for electrically very large objects the time required to compute the coefficients (corresponding to the smaller problem at a lower frequency) is negligible compared to the linear system solution, resulting in an overall operation count proportional to the number of unknowns.

Almost 20 years later, we revisit the IE-MEI idea presenting an improved formulation with a few variations and applying it to high-loss 2D dielectric homogeneous penetrable scatterers like plasmonic media. The aim of this work is to assess if highly lossy scatterers help in overcoming the problems previously encountered with lossless dielectric scatterers.

Most of the code here is developed in MATLAB so as to be able to use the efficient

matrix calculations implemented (very adequate to numerical methods), as well as its plotting and GUI capabilities with ease.

Chapter 2

Method of moments

This chapter is dedicated to the most representative approach to discretize integral equations, the method of moments. This method has been and will be continuously referred to in this work, so a good exposition of its characteristics in some detail is useful and necessary. After all, the IE-MEI method is a modification of the MoM. For this reason, and because no exact solutions exist or are known for the majority of the problems analysed, the unmodified method of moments will be the reference used for comparison and will be compared in terms of speed, memory and behaviour of the solution. Furthermore, the code of the MoM is necessary as a first step to develop the IE-MEI equations.

The method of moments, as the rest of the integral methods, calculates the radiated fields from the induced currents in the scatterer and the boundary conditions. Because of the complexity of performing calculations with the presence of the object, the calculation is performed via the reciprocity theorem in vacuum. This is done in three steps: application of the equivalence theorem, formulation of the integral equations and discretization of the problem.

2.1 Surface equivalence

The equivalence theorem is a means of formulating an equivalent problem to the given original; that is, formulating a problem with the exact same solution. The idea is to substitute the media by a set of equivalent currents (not real) that radiate in the free space the same field as the one in the original problem. This allows the use of integrals with Green functions in free space, vastly simplifying the calculations.

There are two ways to apply the equivalence principle, depending on where the equiv-

alent currents are defined: volumetric and surface equivalence. The latter is used here: the principle of surface equivalence is useful when the medium is piecewise homogeneous (for example, in the case of a homogeneous scatterer in vacuum). In this case, as shown in the 2.1, the currents are defined at the boundaries between homogenous regions of the medium (this boundaries are 2D surfaces in general 3D problems, or 1D figures in the case of symmetry along an axe).

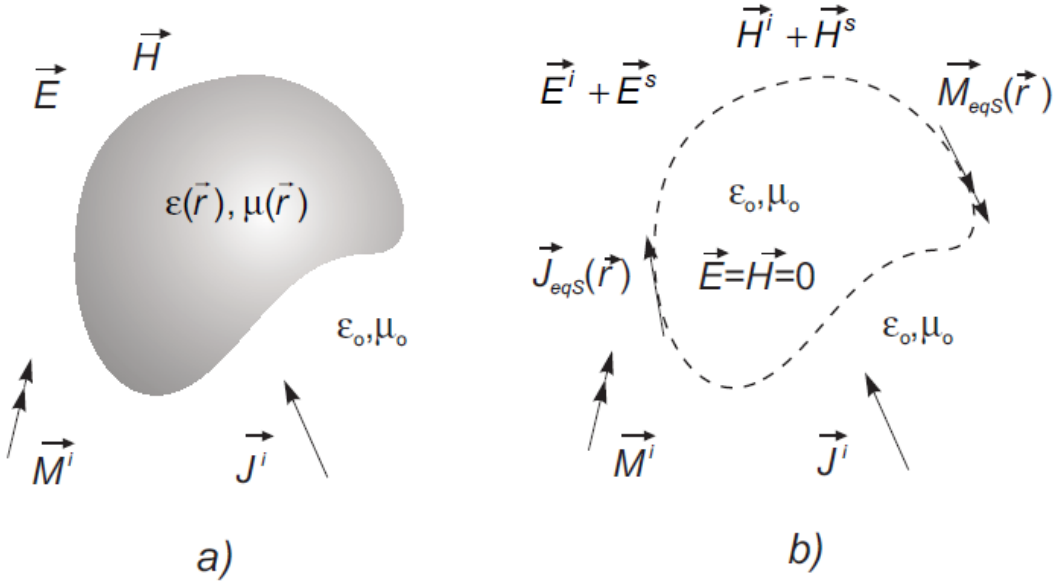

Figure 2.1: Equivalence theorem

Let V be an arbitrary homogeneous (bounded an oriented) region –representing a dielectric scatterer–, embedded in a homogeneous (unbounded) medium, $\overline{V}$. According to the equivalence principle, electric and magnetic fields can be decomposed as $\vec{E} = \vec{E}_i + \vec{E}_s$ and $\vec{H} = \vec{H}_i + \vec{H}_s$, where $\vec{E}_i$ and $\vec{H}_i$ are the fields that would exist in the absence of the scatterer (also called incident fields) and

$$\vec{E}_s = \mathcal{L}_{EJ}\vec{J}_{eq} + \mathcal{L}_{EM}\vec{M}_{eq} \tag{2.1}$$

$$\vec{H}_s = \mathcal{L}_{HJ}\vec{J}_{eq} + \mathcal{L}_{HM}\vec{M}_{eq} \tag{2.2}$$

are the electric and magnetic fields due to the electrical and magnetic equivalent currents:

$$\vec{J}_{eq} = \vec{n} \times \vec{H} \tag{2.3}$$

$$\vec{M}_{eq} = -\vec{n} \times \vec{E} \tag{2.4}$$

The $\mathcal{L}_{ij}$ in the previous equations are the integro-differential operators that relate the scattered field with the equivalent currents, and $\vec{n}$ is the vector normal to the surface pointing outwards (towards the embedding medium). From this point on we will drop the "eq" sub-index from the currents.

2.2 Integro-differential equations

At any point r, the total field of an electromagnetic scattering problem defined by V is given by the integral equations

$$\vec{E}(r) = \theta(r)(\vec{E}_i(r) - \Lambda\vec{J} + \Omega\vec{M}) \tag{2.5}$$

$$\vec{H}(r) = \theta(r)(\vec{H}_i(r) - \Omega\vec{J} - \frac{1}{\eta^2}\Lambda\vec{M}) \tag{2.6}$$

where $\theta(r)$[1] is a constant with different values depending on if $r \in V$, E_i and H_i the incident electromagnetic field, J and M (unknown) surface current densities, $\eta = \sqrt{\frac{\mu}{\epsilon}}$ is the impedance of the medium and Λ and Ω linear integro-differential operators. Given a surface field $\vec{X}(r)$, Λ and Ω are defined by

$$\Lambda\vec{X}(r) = \int_{\partial V} \left(j\omega\mu\vec{X}(r') + \frac{j}{\omega\epsilon}\nabla(\nabla'\vec{X}(r')))G(r-r') \right) \mathrm{d}s' \tag{2.7}$$

and

$$\Omega\vec{X}(r) = \int_{\partial V} \vec{X}(r) \times \nabla G(r-r')\mathrm{d}s' \tag{2.8}$$

where $G(r-r')$ is the corresponding Green function (that varies depending on the problem). Thus, the previous $\mathcal{L}_{ij}$ are easily identified.

2.3 Boundary conditions

Working with the equivalence theorem requires the use of operators 2.7 and 2.8, thus evaluating the radiation integrals in the points of ∂V. When the source point and the field point are coincident, $r' = r$, the Green function is singular and the integrals become improper. To calculate the correct value, the integral is divided between Cauchy principal value (coincident with numerical integration) and the singularity value, calculated

[1] See next subsection.

analytically

$$\int_S \mathrm{d}s = \lim_{a\to 0} \int_{S_a} \mathrm{d}s + \int_{S_{vp}} \mathrm{d}s \tag{2.9}$$

To account for the singularity, the value of $\theta(r)$ can be given as follows

$$\theta(r) = \begin{cases} \frac{4\pi}{\Omega_0} \text{ if } r \in V \\ 0 \text{ otherwise} \end{cases} \tag{2.10}$$

where Ω_0 is the solid angle covered by the exterior of the surface at r. If the surface presents no discontinuities, $\Omega_0 = 2\pi$ and thus $\theta(r) = 2$.

2.4 Equations for a penetrable dielectric scatterer

Applying the equivalent currents for an arbitrary dielectric scatterer, we get in the embedding medium $\overline{V}$:

$$\vec{J_1} = \vec{n}_1 \times \vec{H}_1 \tag{2.11}$$

$$\vec{M_1} = -\vec{n}_1 \times \vec{E}_1 \tag{2.12}$$

where n_1 is normal to ∂V pointing to $\overline{V}$, and in the object V

$$\vec{J_2} = \vec{n}_2 \times \vec{H}_2 \tag{2.13}$$

$$\vec{M_2} = -\vec{n}_2 \times \vec{E}_2 \tag{2.14}$$

where $\vec{n}_2$ is the normal pointing inwards. Therefore, $\vec{n}_2 = -\vec{n}_1$, so we can define $\vec{n}_1 \equiv \vec{n}$, and imposing the boundary conditions[2] we get

$$\vec{J_2} = -\vec{n} \times \vec{H}_2 = -\vec{n} \times \vec{H}_1 = -\vec{J_1} \tag{2.15}$$

$$\vec{M_2} = \vec{n} \times \vec{E_2} = -\vec{n} \times \vec{E}_1 = -\vec{M}_1 \tag{2.16}$$

so we can rename $\vec{J_1} \equiv \vec{J}$ and $\vec{M_1} \equiv \vec{M}$. Performing the cross product by $\vec{n}$ on 2.5 both in V and in $\overline{V}$ and substituting with $\vec{n} \times \vec{E} = -\vec{M}$ we obtain the EFIE equations

$$\vec{n} \times \vec{E}_i = \vec{n} \times \Lambda_1 \vec{J} - \vec{n} \times \Omega_1 \vec{M} - \frac{\vec{M}}{2} \tag{2.17}$$

$$0 = \vec{n} \times \Lambda_2 \vec{J} - \vec{n} \times \Omega_2 \vec{M} + \frac{\vec{M}}{2} \tag{2.18}$$

[2] $\vec{n} \times (\vec{E}_1 - \vec{E}_2) = 0$ and $\vec{n} \times (\vec{H}_1 - \vec{H}_2) = 0$

where in V $\vec{E}_i = 0$. Analogously, on 2.6 we get the MFIE equations

$$\vec{n} \times \vec{H}_i = \vec{n} \times \Omega_1 \vec{J} + \frac{1}{\eta_1^2} \vec{n} \times \Lambda_1 \vec{M} + \frac{\vec{J}}{2} \tag{2.19}$$

$$0 = \vec{n} \times \Omega_2 \vec{J} + \frac{1}{\eta_2^2} \vec{n} \times \Lambda_2 \vec{M} - \frac{\vec{J}}{2} \tag{2.20}$$

Finally, operating again with $-\vec{n} \times \cdot$, we get the tangential vectors to ∂V and the equations that will be used for discretization:

1. EFIE:

$$\vec{E}_i^{tan} = \Lambda_1^{tan} \vec{J} - \Omega_1^{tan} \vec{M} + \vec{n} \times \frac{\vec{M}}{2} \tag{2.21}$$

$$0 = \Lambda_2^{tan} \vec{J} - \Omega_2^{tan} \vec{M} - \vec{n} \times \frac{\vec{M}}{2} \tag{2.22}$$

2. MFIE:

$$\vec{H}_i^{tan} = \Omega_1^{tan} \vec{J} + \frac{1}{\eta_1^2} \Lambda_1^{tan} \vec{M} - \vec{n} \times \frac{\vec{J}}{2} \tag{2.23}$$

$$0 = \Omega_2^{tan} \vec{J} + \frac{1}{\eta_2^2} \Lambda_2^{tan} \vec{M} + \vec{n} \times \frac{\vec{J}}{2} \tag{2.24}$$

It is worth noting that these equations permit solving the equivalent currents, J and M, in terms of the incident fields. This won't be, in general, possible analytically, but its discretization is simple.

Because of the given derivation of the EFIE and MFIE, these equations are valid, in general, only for closed surfaces. However, the EFIE can also be applied to open PEC surfaces.

2.5 Discretization with the MoM

Because of the linearity of equations 2.21 to 2.24, the equations can be represented by the general form

$$LX = Y \tag{2.25}$$

where L is one of the linear operators, X is the unknown (the equivalent currents) and Y the independent term (the incident field). To solve 2.25 numerically, it's necessary to

discretize the functions and operators and convert the functional equation to a matrix equation[3]. The first step is to approximate the unknown X by a linear combination of base functions x_j:

$$X \simeq X_N = \sum_j^N a_j x_j \tag{2.26}$$

where a_j are the elements of the discretization of X and constitute the unknowns of the numerical problem. Thus, its approximation X_N will be a vector of N elements. For X_N to be a good approximation of X (which, in general, will be infinite dimensional), the basis functions must belong to the operator domain.

By replacing 2.26 into 2.25 we get

$$L \sum_j^N a_j x_j = \sum_j^N a_j L x_j = Y_N \simeq Y \tag{2.27}$$

which is a functional equation with N unknowns a_j. Again, it is necessary that a linear combination of Lx_j functions be a good approximation of Y, so Lx_j must be a basis of the field.

The error in the boundary conditions, or residual, is

$$R = Y - Y_N = Y - \sum_j^N a_j L x_j \tag{2.28}$$

This residual is made zero by weighting it with M weight functions w_i, so

$$\langle w_i, R \rangle = 0 \qquad i = 1 \dots M \tag{2.29}$$

with the inner product defined in the Hilbert space sense

$$\langle f, g \rangle = \int_\Omega f^* \cdot g \tag{2.30}$$

Hence, the system of M linear equations with N unknowns becomes

$$\sum_j^N a_j \langle w_i, L x_j \rangle = \langle w_i, Y \rangle \qquad i = 1 \dots M \tag{2.31}$$

that in matrix form becomes

$$Ma = b \tag{2.32}$$

[3]Because of the linearity of the differential operator.

where M is a $M \times N$ matrix with $M_{ij} = \langle w_i, Lx_j \rangle$, a is the unknowns vector formed by the a_j coefficients and b is the field vector, with $b_i = \langle w_i, Y \rangle$. To solve the system a matrix inversion (or equivalent procedure) is performed if M is square, not singular

$$a = M^{-1}b \tag{2.33}$$

If the system is overdetermined, usually a least squares method (or other similar criteria) is used. However, in the rest of this work, $M = N$ and the obtained matrices will all be square.

Chapter 3

The MEI method

Finite difference methods (FD) discretize directly the Maxwell's equations, representing derivatives by an appropriate use of finite differences that lead to sparse linear systems that can be efficiently solved. In scattering problems, FD methods raise the necessity of approximating the boundary conditions in the infinite, for the fields to verify Sommerfelds' condition, by imposing some specific constraints, and therefore have a volume mesh between the scatterer surface and the mesh truncation boundary.

In this chapter some basic concepts of differential methods are introduced, and the differential MEI method (original idea for the IE-MEI method) is also discussed.

3.1 Finite differences

In finite differences methods, direct discrete approximations on partial derivatives are made, with different orders of finite differences used according to particular needs. Finite differences, therefore, can be used in Maxwell's equations, the wave equation or any other partial differential equation of interest.

Assuming the function whose derivatives are to be approximated is properly-behaved, by Taylor's theorem, we can create a Taylor Series expansion

$$f(x_0 + h) = f(x_0) + \frac{f'(x_0)}{1!}h + \frac{f^{(2)}(x_0)}{2!}h^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}h^n + R_n(x) \tag{3.1}$$

where $R_n(x)$ is a remainder term, denoting the difference between the Taylor polynomial of degree n and the original function. We will derive an approximation for the first derivative of the function f by first truncating the Taylor polynomial

$$f(x_0 + h) = f(x_0) + f'(x_0)h + R_1(x) \tag{3.2}$$

Setting $x_0 = a$ we have

$$f(a+h) = f(a) + f'(a)h + R_1(x) \tag{3.3}$$

Dividing across by h gives:

$$\frac{f(a+h)}{h} = \frac{f(a)}{h} + f'(a) + \frac{R_1(x)}{h} \tag{3.4}$$

Solving for $f'(a)$:

$$f'(a) = \frac{f(a+h) - f(a)}{h} - \frac{R_1(x)}{h} \tag{3.5}$$

Assuming that $R_1(x)$ is sufficiently small, the approximation of the first derivative of f is:

$$f'(a) \approx \frac{f(a+h) - f(a)}{h} \tag{3.6}$$

Of course, approximations of higher order can be made by following the same scheme.

Using the 2D wave equation for the scattering problem (the incident field in TM or TE polarisation)

$$(\nabla^2 + k^2)\phi(r) = 0 \tag{3.7}$$

in the frequency domain, where $\phi(r)$ is the scattered field and k is the *wavenumber*. The numerical solution can be found by approximating with the finite difference method. A grid mesh (see 3.1) with constant distance h between nodes must be used, and thus we obtain the finite differences equation

$$\phi_1 + \phi_2 - (4 - k^2h^2)\phi_0 + \phi_3 + \phi_4 = 0 \tag{3.8}$$

The resolution of a boundary problem like 3.8 requires the following previous steps

1. *Defining the domain, or space that contains to object to be analysed*: this is the domain that will be meshed. Ideally this domain would be infinitely large but, of course, this is impossible in numerical methods. Boundary conditions verifying the Sommerfeld radiation condition are set.

2. *Stating the equation in each point in the mesh*, as defined previously.

3. *Solving the obtained system.* In finite differences, the system matrix is of big dimensions (every point of the mesh is an unknown ϕ_i, and a lot of mesh points must be used outside the scatterer in order to get a good solution), but as a differential equation, each point is only related to the points in a neighbourhood, and the matrix is very sparse.

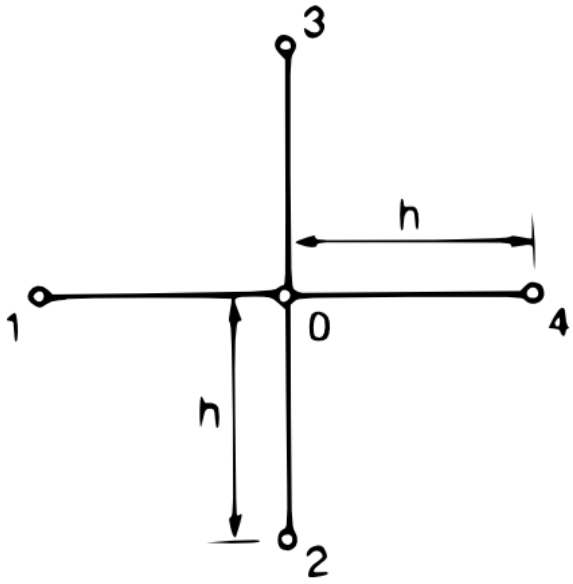

Figure 3.1: A typical finite difference mesh point and local numbering

3.2 Differential MEI

The main problem in terms of quality of the solution in the finite difference method is the proximity of the object to analyse to the ending of the mesh. Logically, the farther away this is, the better the boundary constraint will be and the obtained solution. However, as the mesh grows bigger in volume, the number of points increases as $\mathcal{O}(n^3)$, therefore increasing dramatically the memory and computing time. There is, thus, a trade-off –as often in engineering problems– between the accuracy and the practical implementation of the method.

3.2.1 The postulates

K. Mei introduced the concept of Measured Equation of Invariance (MEI) in [8] as a series of "postulates" to sort out this problem. He advanced that there existed a set of equations for every point in the mesh, analogous as the equations in the finite difference method, that were

1. Location dependent.
2. Geometry specific
3. Invariant to the field of excitation

Therefore, in this case and in contrast with the FDM, the equations at every node wouldn't be the same, as they would depend on the point and geometry of the scatterer. However, the solution would be the same. This idea could be used to terminate the mesh almost anywhere and, in particular, close to the scatterer, hence dramatically reducing the size of the system.

"Postulate" 3 suggests that the coefficients of the equations (that will be used later to solve the problem) can be found by introducing $n-1$ linearly independent solutions of the problem and solving

$$\sum_{i=0}^{n-1} a_i \phi_1 = 0 \tag{3.9}$$

for every solution. Mei uses the example of complex plane waves of the form e^{jkh} as a set of linearly independent solutions.

It is worth noting that Mei didn't provide any proof of the existence and correctness of his postulates.

3.2.2 Metrons

However, a more general method of finding "solutions" is required, as plane waves are clearly not geometry specific. This is where Mei introduced the concept of *metrons*: a set of current densities on the surface of the scatterer, $J_k(r)$ for $k = 1, 2 \ldots, n-1$, from which the solutions to be introduced in the previous systems (this solutions are also called *measures*) are derived via the integral with the corresponding Green function

$$\phi^k(\vec{r}) = \int_{\partial V} G(\vec{r}|\vec{r}')J_k(\vec{r})\mathrm{d}s \tag{3.10}$$

where $G(\vec{r}|\vec{r}')$ is the Green's function of the wave function, k are the nodal points of the $n-1$ neighbours, ∂V is the contour of the scatterer, $\vec{r}$ is the coordinate location of the node and $\vec{r}'$ is the coordinate location of the metron. It should be emphasized that the metrons are not basis functions, so there is no requirement for metrons to resemble the actual current density. Two immediate sets of metrons are

- *Delta metrons*: rectangular shaped current densities, with constant current inside the rectangle and zero current outside.
- *Harmonic metrons*: complex exponential current distributions (only in 2D).

If the third postulate was correct, it should be possible to terminate a finite difference mesh almost anywhere, and in particular, much closer to the object than before. Nonetheless, it is in theory more advantageous to use it in conjunction with the finite difference method, using the MEI method where needed.

3.2.3 Consequences and correctness of the postulates

The interest of the MEI method is evident: it is, in one hand, a differential method, so the resultant matrix is going to be very sparse; also, as the meshing can be terminated very soon, a very reasonable number of nodes and, therefore, unknowns, can be used. Both things contribute to a decrease in computing time and memory requirements.

It would seem, hence, that the MEI method has all the advantages of the differential methods (locality) and all the advantages of integral methods (few unknowns). However, as it turned out, the third postulate was false (see [11]), and no exact MEI method exists.

A different formulation of the MEI, by JM Rius et al, was introduced soon after, and is the method used in this work (it is discussed in the next chapter). Although the method could not provide the same solutions as the finite difference method, the question of whether a good enough approximation could be provided with all the previous advantages remained.

Chapter 4

Integral formulation of the MEI (IE-MEI)

A novel integral formulation of the measured equation of invariance was introduced by J.M. Rius et al in [2], derived from the reciprocity theorem. This formulation uses the electric and magnetic Green's functions of the environment to obtain a matrix equation for the induced surface current with the same number of unknowns as the conventional boundary element - MoM approach. However, the inverted matrix is sparse and circulant, with only some non-zero elements per row close to the diagonal.

4.1 Derivation from the reciprocity theorem

Reciprocity theorem states that two sets of electric and magnetic sources $(\vec{J}_1, \vec{M}_1)$ and $(\vec{J}_2, \vec{M}_2)$ that radiate simultaneously at the same frequency in a linear and isotropic medium, respectively produce electric and magnetic fields $(\vec{E}_1, \vec{H}_1)$ and $(\vec{E}_2, \vec{H}_2)$ that satisfy

$$\int_V \left(\vec{E}_1 \cdot \vec{J}_2 - \vec{H}_1 \cdot \vec{M}_2\right) \mathrm{d}v = \int_V \left(\vec{E}_2 \cdot \vec{J}_1 - \vec{H}_2 \cdot \vec{M}_1\right) \mathrm{d}v \tag{4.1}$$

Let ∂V be the boundary of a perfectly conducting scatterer and ∂V_0 a portion of this boundary . Let the first set of electric and magnetic sources be the induced currents on the surface of the PEC scatterer $\vec{J}_1 = n \times \vec{H}|_C = \vec{J}_s$ and $M_1 = -n \times \vec{E}|_C = 0$, where $\vec{E}$ and $\vec{H}$ are the electric and magnetic fields actually existinc in the presence of the scatterer. According to the equivalence theorem, the induced currents on a PEC surface are equal

to the surface equivalent currents that radiate in free space the scattered fields

$$\vec{E}_s = \vec{E} - \vec{E}_i \tag{4.2}$$

$$\vec{H}_s = \vec{H} - \vec{H}_i \tag{4.3}$$

where the fields with subscript i are fields in the absence of the scatterer (or *incident fields*). Let the second set of electric and magnetic sources $(\vec{J}_2, \vec{M}_2)$ exist only over ∂V_0 and radiate electric field $\vec{E}_2 = \vec{E}_{\text{null}}$. With these sources, equation 4.1 becomes

$$\int_{\partial V_0} \left(\vec{E}^s \cdot \vec{J}_2 - \vec{H}^s \cdot \vec{M}_2 \right) \mathrm{d}l = \int_{\partial V} \vec{E}_{\text{null}} \cdot \vec{J}_s \, \mathrm{d}l = \langle \vec{E}^*_{\text{null}}, \vec{J}_s \rangle \tag{4.4}$$

where $\langle f, g \rangle$ is the Hilbert inner product defined along the scatterer boundary. Since $\vec{J}_2$ and $\vec{M}_2$ are surface currents, they are tangent to the scatterer boundary and

$$\int_{\partial V_0} \left(\vec{E}^s_t \cdot \vec{J}_2 - \vec{H}^s_t \cdot \vec{M}_2 \right) \mathrm{d}l = \langle \vec{E}^*_{\text{null}}, \vec{J}_s \rangle \tag{4.5}$$

This equation 4.5 is the integral equation formulation of the MEI or IE-MEI.

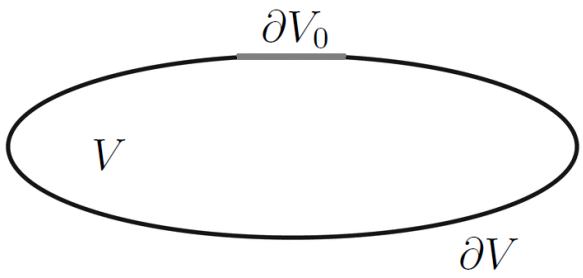

Figure 4.1: Boundary of a perfectly conduction scatterer, and a portion of this boundary.

Its discretization is only presented for 2D problems. The boundary ∂V and its portion ∂V_0 are discretized into respectively N and M segments of equal length h. Let ∂V_0 be centered at node n of ∂V (figure 4.1). The currents $\vec{J}_2$ and $\vec{M}_2$ are expanded along ∂V_0 into M pulse basis functions as

$$\vec{J}_2(l) = \sum_{m=n-\frac{M-1}{2}}^{n+\frac{M-1}{2}} \vec{J}_2(l_m) \Pi(l - l_m) \tag{4.6}$$

$$\vec{M}_2(l) = \sum_{m=n-\frac{M-1}{2}}^{n+\frac{M-1}{2}} \vec{M}_2(l_m) \Pi(l - l_m) \tag{4.7}$$

where $0 \leq l \leq L$ is the arclength along the scatterer boundary and the pulse basis is

$$\Pi(l - l_m) = \begin{cases} 1 \text{ if } |l - l_m| < h/2 \\ 0 \text{ otherwise} \end{cases} \tag{4.8}$$

Substituting the expansion of the currents into equation 4.5 and interchanging the order of summation and integral operators, one obtains

$$\sum_{m=n-\frac{M-1}{2}}^{n+\frac{M-1}{2}} \left(\vec{J}_2(l_m) \cdot \int_{\partial V_0} \vec{E}_t^s(l)\Pi(l - lm)\,\mathrm{d}l - \vec{M}_2(l_m) \cdot \int_{\partial V_0} \vec{H}_t^s(l)\Pi(l - lm)\,\mathrm{d}l \right) = \\ = \langle \vec{E}^*_{\text{null}}, \vec{J}_s \rangle \tag{4.9}$$

Since h is small and $\vec{E}^2$ is smooth along the PEC scatterer boundary, we can use $\vec{E}^s(l) \simeq \vec{E}^s(l_m)$. If the segments do not contain corners in the TM case, $\vec{H}^s$ is also smooth inside them and $\vec{H}^s(l) \simeq \vec{H}^s(l_m)$ for $|l - l_m| < h/2$. If we then define $a_{nm} = h\vec{J}_2(l_m)$ and $b_{nm} = h\vec{M}_2(l_m)$, we obtain

$$\sum_{m=n-\frac{M-1}{2}}^{n+\frac{M-1}{2}} \Big(a_{nm} \cdot \vec{E}^s(l_m) - (b_{nm} \cdot \vec{H}^s(l_m) \Big) = \langle \vec{E}^*_{\text{null}}, \vec{J}_s \rangle \tag{4.10}$$

which is the discrete version of the IE-MEI for 2D PEC.

If the residual of the IE-MEI is zero, which can be approximately achieved in the 2D case for convex scatterer shapes with appropriate a_{nm} and b_{nm} coefficients, in two dimensions we have

$$\sum_{m=n-\frac{M-1}{2}}^{n+\frac{M-1}{2}} \Big(a_{nm} \cdot \vec{E}^s(l_m) - (b_{nm} \cdot \vec{H}^s(l_m) \Big) = 0 \tag{4.11}$$

4.2 Derivation as a CFIE from the MoM

4.2.1 EFIE and MFIE for PEC

In the previous section, we can express 4.11 more concisely by writing the equations in matrix form

$$[A][E^s] - [B][H^s] = 0 \tag{4.12}$$

This formulation can also be reached, perhaps more clearly, as a combined field integral equation (CFIE) from the EFIE and MFIE in the method of moments. For a PEC, we have the following boundary conditions in the change of –homogeneous– media:

$$0 = \vec{n} \times \vec{E} \tag{4.13}$$

$$\vec{J}^s = \vec{n} \times \vec{H} \tag{4.14}$$

and substituting in 2.2 we obtain

$$-\vec{n} \times \vec{E}^i = \vec{n} \times \mathcal{L}_{EJ}\vec{J}^s \tag{4.15}$$

$$\vec{J}^s - \vec{n} \times \vec{H}^i = \vec{n} \times \mathcal{L}_{HJ}\vec{J}^s \tag{4.16}$$

which are the EFIE and MFIE equations for the case of a PEC. By using the method of moments to discretize this equations with a set of testing functions $\vec{w}_i$, tangent to the boundary $\vec{w}_i \cdot \vec{n}$, and expanding the induced current with a set of basis functions

$$\vec{J}^s(\vec{r}) = \sum_{i=1}^{N} c_i \vec{u}_i(\vec{r}) \tag{4.17}$$

as seen in 2.5, the matrix equations take form as

$$-[E^i] = [Z^E][C] \tag{4.18}$$

$$-[H^i] = ([Z^H] - [D])[C] \tag{4.19}$$

where the matrices $[Z^E]$ and $[Z^H]$ are called *impedance matrices*, the elements of them, as well as of the matrices $[D]$, $[E^i]$ and $[H^i]$ are

$$d_{mi} = \langle \vec{w}_m, \vec{u}_i \rangle \tag{4.20}$$

$$z^E_{mi} = \langle \vec{w}_m, \mathcal{L}_{EJ}\vec{u}_i \rangle \tag{4.21}$$

$$z^H_{mi} = \langle \vec{w}_m \times \vec{n}, \mathcal{L}_{HJ}\vec{u}_i \rangle \tag{4.22}$$

$$e_m = \langle \vec{w}_m, \vec{E}^i \rangle \tag{4.23}$$

$$h_m = \langle \vec{w}_m \times \vec{n}, \vec{H}^i \rangle \tag{4.24}$$

4.2.2 Linear combination and equivalence

Multiplying 4.19 equations by some matrices $[A]$ and $[B]$ respectively, and adding the result, one obtains

$$-[A][E^i] - [B][H^i] = ([A][Z^E] + [B][Z^H] - [B][D])[C] \tag{4.25}$$

Since $[D]$ is sparse for subdomain functions $\vec{u}_i$ and $\vec{w}_i$, and $[B]$ can be arbitrarily chosen sparse, the matrix to be inverted in the solution for $[C]$ (the coefficients of the current) will be sparse if $[A][Z^E]+[B][Z^H]$ is sparse. This can be interpreted as a combined impedance matrix

$$[Z^C] = [A][Z^E] + [B][Z^H] \tag{4.26}$$

Moreover, by the PEC boundary conditions and equivalence theorem

$$[E^s] = -[E^i] \tag{4.27}$$

$$[H^s] = [D][C] - [H^i] \tag{4.28}$$

so substituting the incident fields into 4.12, we get

$$[A][E^i] + [B][H^i] = [B][D][C] \tag{4.29}$$

which is exactly equation 4.25 in the case $[Z^C][C] = ([A][Z^E] + [B][Z^H])[C] = 0$. For this reason, the IE-MEI is an approximate combined field integral equation in which the different weighting of the EFIE and the MFIE produces a vanishing product of the combined impedance matrix $[Z^C]$.

In practice it is not possible to find sparse matrices $[A]$ and $[B]$ such that the term $[Z^C][C]$ vanishes exactly: it suffices to see that the number of constraints outweighs the number of variables in these matrices.

4.3 Coefficient computation

To calculate the coefficients of $[A]$ and $[B]$, let z_n^{AB} the set of non-zeros in these matrices. Then, an element in row n of $[Z^C] = [A][Z^E] + [B][Z^H]$ may be expressed as

$$z_{ni}^C = \sum_{m \in z_n^{AB}} a_{nm} z_{mi}^E + \sum_{m \in z_n^{AB}} b_{nm} z_{mi}^H \tag{4.30}$$

where i is the column index in the impedance matrices. Setting $z_{ni}^C = 0$ for $i \in z_n^Z$ and setting $a_{nn} = 1$, 4.30 becomes a linear system of $N - N_n^Z$ equations with $2N_n^{AB} - 1$

unknowns, whereN_n^Z is the number of large elements in row n of $[Z^C]$ and N_n^{AB} is the number of non-zeros in row n of $[A]$ or $[B]$. In matrix form

$$\begin{pmatrix} A_{nm} & B_{nm} \end{pmatrix} \begin{pmatrix} Z^E_{mi} \\ Z^H_{mi} \end{pmatrix} = -[Z_n i^E] \qquad i \in z_n^Z, \quad m \in z_n^{AB}, \quad m \neq n \tag{4.31}$$

where the notation $M_i j$ denotes the submatrix of $[M]$ formed by the set of rows i and the set of columns j.

This system of equations is, in general, overdetermined, but can be solved in the least squares sense: for a given system $Ax = b$, the following equation

$$x = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}b \tag{4.32}$$

is the one that minimizes $\min_x \|Ax - b\|$, where the norm is the L^2 norm. If A is complex (as is the case with the IE-MEI), the Hermitian transpose is used instead of the classic transpose.

Therefore, with this procedure we can guarantee that the 2-norm of the vector of small elements (the ones that we would want to be zero) is minimum

$$\sum_{i \in z_n^Z} |z_{ni}^C|^2 = \min \tag{4.33}$$

and it will be possible to obtain matrices that, in theory, can make $[Z^C]$ very sparse by imposing a threshold to eliminate all the "small elements".

4.4 Metrons

4.4.1 Coefficient computation

In the solution of

$$-[A][E^i] - [B][H^i] = ([Z^C] - [B][D])[C] \tag{4.34}$$

where $[B]$ and $[D]$ are very sparse and $[C]$ is the vector of unknowns, it is not strictly necessary for $[Z^C]$ to be sparse or zero. If $[Z^C]$ can be decomposed in a sparse matrix $[Z^{CS}]$ that contains the large elements of $[Z^C]$ and zeros in place of the small elements, and a matrix $[Z^{CZ}]$ that contains the small elements of $[Z^C]$ and zeros in place of the large elements

$$[Z^C] = [Z^{CS}] + [Z^{CZ}] \tag{4.35}$$

then, to approximate $[Z^C]$ by $[Z^{CS}]$ in 4.34 it is enough that

$$[Z^{CZ}][C] \simeq 0 \tag{4.36}$$

which is a less restrictive condition that $[Z^{CZ}] \simeq 0$. When the coefficient matrices $[A]$ and $[B]$ are obtained by a least squares procedure, clearly the minimization of $\|[Z^{CZ}][C]\|$ produces an error smaller than the minimization of $\|[Z^{CZ}]\|$ when we approximate $[Z^C][C]$ by $[Z^{CS}][C]$. Unfortunately, this cannot be used to compute the IE-MEI matrices $[A]$, $[B]$, because $[C]$ is precisely the discretization coefficients of the induced current, which is unknown. Determining $[C]$ is the goal of solving the problem: once solved, there is no point in using it again to calculate $[A]$ and $[B]$, because $[A]$ and $[B]$ are part of the determination of $[C]$.

However, we can force for a set of P column vectors $[\sigma_p]$ that expand $[C]$ as closely as possible. As has been seen before, these are called the *metrons* of the method

$$[C] \simeq \sum_{p=1}^{P} [\sigma_p] \tag{4.37}$$

The system of equations to solve in order to obtain the coefficients a_{nm}, b_{nm} in row n of $[A]$ and $[B]$ is now

$$[Z_n i^E][\sigma_{ip}] = \begin{pmatrix} A_{nm} & B_{nm} \end{pmatrix} \begin{pmatrix} Z^E_{mi} \\ Z^H_{mi} \end{pmatrix} [\sigma_{ip}] = 0 \qquad i \in z_n^Z, \quad m \in z_n^{AB}, \quad p = 1, \ldots, P \tag{4.38}$$

4.4.2 Types of metrons

Different sets of metrons can be used in the system of equations 4.38. The more immediate are

- Harmonic metrons: the set of harmonic metrons for two dimensional scatterers is

$$\sigma_{ip} = e^{j2\pi ip/N} \qquad |p| < kR_{\max} \tag{4.39}$$

  where $R_{\max}$ is the radius of the minimum circumference that encloses the scatterer, p is the order of the metron and i increases along the scatterer boundary. This set approximately expands the induced current vector. However, for electrically large scatterers the number of metrons is much larger than the number of unknowns in the system of equations 4.38. If the system of equations is solved in the least squares

sense, the minimum $\|[Z^{CZ}][C]\|$ will be obtained and, since $[C] \simeq \sum_p \sigma_p$, the 2-norm of the term to be neglected in equation is approximately the minimum possible.

The main advantage of this set of metrons is that it allows the computation of the measures with the FFT with an operation count proportional to $N \log N$. The main problem is that it only works in 2D problems.

- Delta metrons: if the metrons are Kronecker delta functions

$$[\sigma_{ip}] = \begin{cases} 1 \text{ if } 1 = p \\ 0 \text{ if } 1 \neq p \end{cases} \tag{4.40}$$

the matrix of the metrons $[\sigma]$ is the identity matrix, and equation 4.38 becomes the same as before after setting $a_{nn} = 1$. Accordingly, the procedure to find the coefficients is an implementation of the MEI method using a set of delta metrons. If the equation is solved in the least squares sense, we have the minimum $\|[Z^{CZ}]\|$ but not the minimum $\|[Z^{CZ}][C]\|$ as before. The main advantage of this metron set is that it is easily applicable to 3D scatterers.

Chapter 5

Implementation of the code in the 2D case

In this chapter, the code specifically developed for the project is discussed. An adaptation of the IE-MEI method, which had only been used for PEC scatterers, is derived from the MoM equations from chapter 2 in the case of dielectric homogeneous media, in the manner seen in the last chapter.

As previously remarked, in this work there is a specific interest for complex relative permittivity: this allows the simulation of plasmonic and lossy dielectric media when the imaginary part of the permittivity is negative and, thus, the propagation of the field decreases exponentially. This fact is fundamental in the possibility of making the equations more local, as the decrease of the field as it advances in space has immediate consequences in the mutual dependence of nearby points (more dependent) and distant points (almost no relation at all). This notion will be made more precise in the following sections.

5.1 Development and routines

5.1.1 MoM for 2D dielectric scatterer

It was seen in the second chapter that for a homogeneous, oriented and closed dielectric scatterer, the EFIE and MFIE equations are, respectively

$$\vec{E}_i^{tan} = \Lambda_1^{tan}\vec{J} - \Omega_1^{tan}\vec{M} + \vec{n} \times \frac{\vec{M}}{2}$$
$$0 = \Lambda_2^{tan}\vec{J} - \Omega_2^{tan}\vec{M} - \vec{n} \times \frac{\vec{M}}{2}$$

and

$$\vec{H}_i^{tan} = \Omega_1^{tan}\vec{J} + \frac{1}{\eta_1^2}\Lambda_1^{tan}\vec{M} - \vec{n} \times \frac{\vec{J}}{2}$$
$$0 = \Omega_2^{tan}\vec{J} + \frac{1}{\eta_2^2}\Lambda_2^{tan}\vec{M} + \vec{n} \times \frac{\vec{J}}{2}$$

If we discretize all the currents and incident fields in terms of Kronecker deltas (i.e. rectangle functions), the linear operators Λ and Ω can be expressed in terms of matrices.

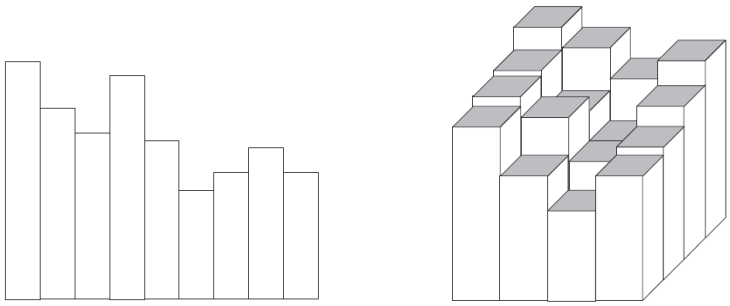

Figure 5.1: Pulse basis functions in 2D and 3D.

These operators had been developed in `C` for PEC back in 1996 by J.M. Rius. Now they have been updated to `C++` for homogeneous dielectric media with complex permittivity constants.

These routines are the only not developed in `MATLAB` language (as well as a complex Bessel's functions library, not written by the authors [1]) for performance reasons. They take the following form, with some apparently strange signs maintained for programming convenience and coherence:

[1] See the code section in Annexes

- EFIE:
  - Λ_i^{tan}, : referred as Z_{eji} in the code, implemented as

    `-oper_die_2d(...,TM,1,...)`
  - $-\Omega_i^{tan} M - n \times \frac{M}{2}$: referred as Z_{emi} in the code, implemented as

    `oper_die_2d(...,TE,0,...)`.

    so we have to add Id to get $-\Omega_i^{tan} M + n \times \frac{M}{2}$

  Thus, discretizing the equations using the method of moments with N elements, we obtain the system

$$\vec{E_i} = [Z_{ej1}]\vec{J} + ([Z_{em1}] + Id)\vec{M} \tag{5.1}$$
$$0 = [Z_{ej2}]\vec{J} + [Z_{em2}]\vec{M} \tag{5.2}$$

- MFIE:
  - Λ_i^{tan}, : referred as Z_{hmi} in the code, implemented as
    `-oper_die_2d(...,TE,1,...)`$/\eta_i^2$
  - $\Omega_i^{tan} J + n \times \frac{J}{2}$: referred as Z_{hji} in the code, implemented as

    `-oper_die_2d(...,TM,0,...)`

    so we have to substract Id to get $\Omega_i^{tan} M + n \times \frac{M}{2}$

  Again, discretizing the equations using the method of moments with N elements, we obtain the system

$$\vec{H_i} = [Z_{hm2}]\vec{M} + ([Z_{hj2}] + Id)\vec{J} \tag{5.3}$$
$$0 = [Z_{hj1}]\vec{J} + [Z_{hm1}]\vec{M} \tag{5.4}$$

5.1.2 MoM in 2Nx2N and NxN form

There are two ways to approach the solution of the obtained systems: the first is to treat it as a singular $2N \times 2N$ system, with matrices

$$\text{EFIE:} \quad \begin{pmatrix} E_i \\ 0 \end{pmatrix} = \begin{pmatrix} Z_1^{EJ} & Z_1^{EM} + Id \\ Z_2^{EJ} & Z_2^{EM} \end{pmatrix} \begin{pmatrix} J \\ M \end{pmatrix} \tag{5.5}$$

$$\text{MFIE:} \quad \begin{pmatrix} H_i \\ 0 \end{pmatrix} = \begin{pmatrix} Z_1^{HJ} & Z_1^{HM} + Id \\ Z_2^{HJ} & Z_2^{HM} \end{pmatrix} \begin{pmatrix} J \\ M \end{pmatrix} \tag{5.6}$$

This matrices can be inverted and solved, or a combination of the two systems can be used (CFIE) and later solved.

The other is to *see* the relations as different equations, and isolating and substituting we get

1. EFIE:

$$\vec{M} = -[Z_{em2}]^{-1}[Z_{ej2}]\vec{J} \tag{5.7}$$

$$\vec{E}_i = \left([Z_{ej1}] - ([Z_{em1}] + Id)[Z_{em2}]^{-1}[Z_{ej2}]\right)\vec{J} \tag{5.8}$$

2. MFIE:

$$\vec{J} = -[Z_{hj1}]^{-1}[Z_{hm1}]\vec{M} \tag{5.9}$$

$$\vec{H}_i = \left([Z_{hm2}] - ([Z_{hj2}] - Id)[Z_{hj1}]^{-1}[Z_{hm1}]\right)\vec{M} \tag{5.10}$$

This approach has the disadvantage that a matrix must be inverted to use it, and with that all the interest in the IE-MEI method vanishes: if the main objective is to reduce the memory requirements to a minimum, with a matrix inversion we will likely get a full matrix. However, if the relative permittivity has negative imaginary part, empiric results show that the inverted matrices relating the electric and magnetic currents will be almost diagonal. Hence, its inversion is trivial and the memory needs negligibles. Also, an $N \times N$ formulation uses $1/4$ of the memory used in a $2N \times 2N$ formulation.

Both formulations have been implemented and tested.

5.1.3 IE-MEI implementation

Circulant 2Nx2N matrix version

For the first formulation described, the IE-MEI is a linear combination of the two such that the matrix to be inverted is almost diagonal,

$$[A][EFIE] + [B][MFIE]$$

If we call $[Z_E]$ and $[Z_H]$ the two EFIE and MFIE matrices (relating the current to the incident fields) we obtain

(5.11)

where the condition $\simeq 0$ has been added, which is the main goal of multiplying by the IE-MEI and adding the result. From this condition the coefficients a_{nm} and b_{nm} of $[A]$ and $[B]$ will be derived. This last expression can be rewritten as

$$\begin{pmatrix} A & B \end{pmatrix} \begin{pmatrix} Z_E \\ Z_H \end{pmatrix} \simeq 0 \tag{5.12}$$

Matrices $[A]$ and $[B]$ will be of the form

$$[M] = \begin{bmatrix} m & m & m & 0 & \dots & 0 & m & m \\ m & m & m & m & 0 & \dots & 0 & m \\ m & m & m & m & m & 0 & \dots & 0 \\ 0 & m & m & m & m & \ddots & & 0 \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & & m & \ddots & \ddots & \ddots & \vdots \\ m & 0 & \dots & 0 & m & m & m & m \\ m & m & 0 & \dots & 0 & m & m & m \end{bmatrix} \tag{5.13}$$

where the subindices of the m elements have been avoided for the sake of simplicity and presentation, and the number of non null elements in each row can be set to any $< 2N$ number. Essentially, therefore, $[A]$ and $[B]$ are diagonal matrices where the diagonal has been expanded circularly to both sides to a number of BW elements, adding to a total of 2BW+1 elements per row.

Thus, for every row of $[A]$ and $[B]$, we'll have to solve the system

$$\begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} Z_E \\ Z_H \end{pmatrix} \simeq 0 \tag{5.14}$$

where (a, b) is just one concatenated row of the matrices. There is a trivial solution of this system,

$$\begin{pmatrix} a & b \end{pmatrix} = 0 \tag{5.15}$$

that we want to avoid. For this reason, the first elements of vector a (that is, the elements that are BW elements to left, counting from the diagonal) is set to 1, as any solution of the system will still be a solution after multiplication by a constant. Hence, the system will be equivalent to

$$\begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} Z_E \\ Z_H \end{pmatrix} = \begin{pmatrix} Z_E & Z_H \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \tag{5.16}$$

$$= \begin{pmatrix} Z'_E & Z_H \end{pmatrix} \begin{pmatrix} a' \\ b \end{pmatrix} = -z_E \tag{5.17}$$

where z_E is the first vector of the $[Z_E]$ matrix and Z'_E and a' are the modified matrix/vector after the extraction of its first element.

Defining

$$M \equiv \begin{pmatrix} Z'_E & Z_H \end{pmatrix} \tag{5.18}$$

we can solve 5.17 in the least squares sense, obtaining

$$\begin{pmatrix} a' \\ b \end{pmatrix} = (M^* M)^{-1} M^* \cdot (-z_E) \tag{5.19}$$

where * is the Hermitian transpose (conjugate transpose).

Without further modifications, this method would lead to

$$[Z] = [A][Z_E] + [B][Z_H] \simeq 0$$

However, $[Z]$ must be inverted to solve the problem and isolate the currents. Consequently a last refinement is necessary: the requirement of $[Z]$ to be one of the following:

- Circulant diagonal band matrix, as with $[A]$ and $[B]$.
- Circulant tridiagonal band matrix, of the form seen in figure This is in principle a more sensible choice, as in the first case the incident fields would only depend of one of the two currents, whether electric or magnetic. The performance of each is discussed in the Results section, as well as the use of circulant tridiagonal $[A]$ and $[B]$ matrices too.

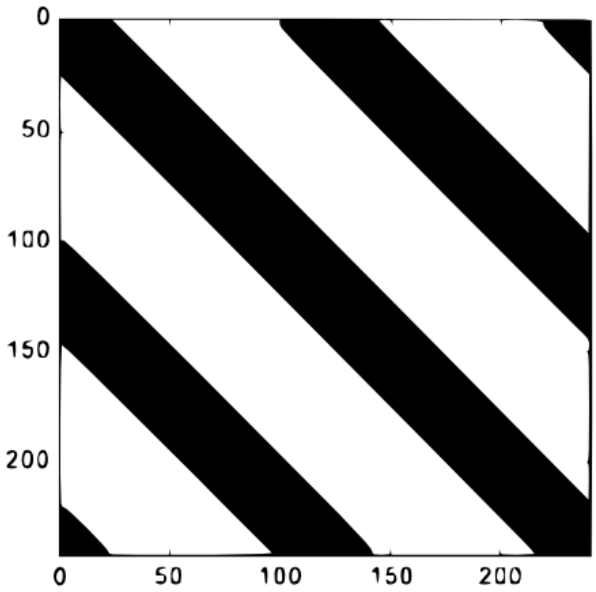

Figure 5.2: Example of a circulant tridiagonal band matrix. In black, non-zero elements

In whichever case, *the size of the diagonals doesn't necessarily have to be the same as in* $[A], [B]$. The number of constraints and number of variables available as coefficients in $[A]$ and $[B]$ must be taken into account.

These modifications can be easily achieved by only forcing zeros in the white sections (as in figure LIU FIGURA), outside of the circulant diagonals. This is accomplished by including in $[M]$ only the rows such that the corresponding column of $[Z]$ vanishes (this calculation will be performed for every row of $[Z]$).

Because of the minimisation of the 2-norm error in an overdetermined system, the elements forced to be zero won't be, in general, zero, but only small. As a last step, then, once the IE-MEI matrices have been applied, a truncation of the matrices to liberate memory will be performed, effectively transforming these matrices into sparse matrices. This can be done by making all elements smaller than a threshold ϵ zero.

NxN version

In the second formulation, by substituting and using $N \times N$ matrices we obtained

- EFIE:

$$\vec{M} = -[Z_{em2}]^{-1}[Z_{ej2}]\vec{J}$$
$$\vec{E}_i = \left([Z_{ej1}] - ([Z_{em1}] + Id)[Z_{em2}]^{-1}[Z_{ej2}]\right)\vec{J}$$

- MFIE:

$$\vec{J} = -[Z_{hj1}]^{-1}[Z_{hm1}]\vec{M}$$
$$\vec{H}_i = \left([Z_{hm2}] - ([Z_{hj2}] - Id)[Z_{hj1}]^{-1}[Z_{hm1}]\right)\vec{M}$$

If we substitute $\vec{M}$ by $\vec{J}$ in the MFIE equations, or viceversa in the EFIE, we get equations

$$\vec{E}_i = \left([Z_{ej1}] - ([Z_{em1}] + Id)[Z_{em2}]^{-1}[Z_{ej2}]\right)\vec{J} \tag{5.20}$$
$$\vec{H}_i = \left([Z_{hm2}][Z_{hm1}]^{-1}[Z_{hj1}] - ([Z_{hj2}] - Id)\right)\vec{J} \tag{5.21}$$

These two equations can be linearly combined by multipliying by two $[A]$, $[B]$ equations as before, following the same method to obtain circular band diagonal matrices after. Now, though, the matrices will be of size $N \times N$, and only electrical (r. magnetic) current will be involved. For this reason, only single diagonal bands are considered, instead of tridiagonal bands as before.

The IE-MEI resultant system will be

$$[A]\vec{E}_i + [B]\vec{H}_i = ([A][Z_E] + [B][Z_H])\vec{J} \tag{5.22}$$

where

$$[Z_E] \equiv \left([Z_{ej1}] - ([Z_{em1}] + Id)[Z_{em2}]^{-1}[Z_{ej2}]\right) \tag{5.23}$$
$$[Z_H] \equiv \left([Z_{hm2}][Z_{hm1}]^{-1}[Z_{hj1}] - ([Z_{hj2}] - Id)\right) \tag{5.24}$$

For the magnetic current, the development of the system is completely analogous.

From a memory point of view, it is in general a bad idea to invert matrices as in 5.21. Nonetheless, as indicated earlier, if the complex permittivity has negative imaginary part and the object is electrically big enough (usually not specially big for the tested values of plasmonic media), the matrix to be inverted is almost diagonal, hence very sparse and not memory intensive.

5.1.4 Reduced zero forcing for the IE-MEI

It has been seen that a least squares problem must be solved for every row of the IE-MEI matrices, to obtain its coefficients. This is easily seen to be too many operations to be worth it: it is usually best to solve the problem directly, instead of finding $[A]$ and $[B]$ matrices, multiplying, adding the two systems and solving. The IE-MEI, in this case, is not justified.

One way to overcome this problem is to force a smaller number of zeros, and hope that this is enough. This will reduce the dimensions of the least squares matrices and make the method much faster.

In the previous implementation of the method by J.M. Rius et al for PEC scatterers, they experimented by forcing only a small number of zeros at the sides of the diagonals, and verified empirically that the rest of elements outside the band diagonal and also after the forced zeros at the sides of these diagonals (thus, elements that ideally we would like to be zero) tended to be also very small, although no restriction was imposed upon them.

This same idea has been implemented for dielectric scatterers. However, as will be seen in the Results section, this "zero bands" are not enough to reduce the rest of elements by proximity once the number of elements inside the BW of the matrices becomes small.

For this reason, and also by aid of some heuristic experimentation, the forced zeros have been spread in different ways over the elements outside the diagonal and bandwidth. The different methods of distribution of the zeros are

1. *In bands near the diagonals*, the old distribution used.

2. *In a number of different bands.* Two after the band diagonal and the rest at constant distances between one another. This way, 3, 4, 5, 7, etc. bands have been used and compared. Some parameters to control the relative size of these bands have also been included.

3. *Equidistributed zeros*, or one zero and a number of non-zeros at constant distance, where distance is measured in number of columns apart, or equivalently as

$$|i_{\mathrm{col}_1} - i_{\mathrm{col}_2}|$$

4. *Normal distribution*, or zeros distributed according to a normal distribution with respect to the distance defined earlier.

5. *Inverse transform distribution.* Simulations show that with different geometries the magnitude of matrix elements in rows (dependency of elements between one another) is highly variable. This is clearly expressed in the EFIE and MFIE matrices, where some elements of the discretization (because of orientation, proximity,...) have high mutual impedance with other elements that are geometrically close together but are distant in matrix rows or columns. Thus, a distribution of the zeros that takes the mutual impedance into account is desirable. This mutual interaction has

been determined in two different ways: by taking into account only the geometry, and considering the elements as small radiating dipoles, and by directly taking the magnitude of EFIE and MFIE matrixs row elements as distribution.

Once a distribution is obtained, the absolute value of its elements (or a monotonous function of it, such as the logarithm or squaring) is the measure used as the future density of forced zeros in the region. This is done as in the inverse transform method in pseudo-random number sampling: if f is the (normalized) probability density function, we'll want a density of forced zeros according to f. Calling F its cumulative distribution, this will be accomplished by forcing the zeros according to $F^{-1}(U)$, where U is a vector of length the number of forced zeros and values ranging from 0 to 1 at constant spacing. Because $F^{-1}(U)$ won't be in general an integer number representing the position of the zero to be forced, rounding is applied after application of F^{-1}.

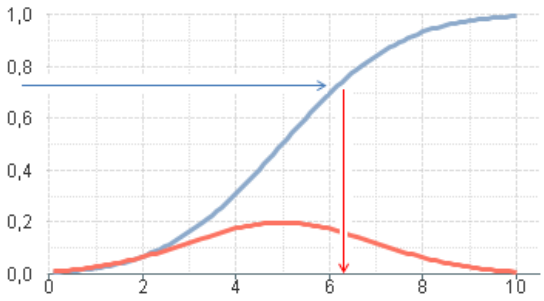

Figure 5.3: Visual representation of the inverse transform method

Here U has been generated with the `MATLAB` function linspace, but in more general cases (such as 3D scatterers), a random uniform generator could be preferred.

5.1.5 Green's function integral in field computation

Being a 2D problem, integration with the Green's function to find the fields generated by the equivalent currents is considerably simplified with respect to non canonical situations. This operation is performed in the routine `get_electric_field` by the rectangle method (where the rectangles are the discretization basis function), by exploiting `MATLAB`'s matrix calculus: first, a complex mesh is created with the real part of its elements representing the x coordinate of a point and the imaginary part the y coordinate. Thus, a matrix with

all the points where we want to integrate is obtained. Then, integration is performed at all points at once by making the proper operations with the matrix, rather than sequentially with a triple loop.

The convolution integral to be performed at every point $r \in \mathbb{R}^2$ takes a simplified form because of the 2D geometry:

$$E_s(r) = \int_{\partial V} \left(\vec{M}(r') \frac{\partial G(k|r-r'|)}{\partial n'} - jk\eta \vec{J}(r') G(k|r-r'|) \right) \mathrm{d}l' \tag{5.25}$$

where $r' \in \partial V$, k is the wavenumber of the corresponding medium (whether the wavenumber of the medium or that of the scatterer) and G is the Green's function of the problem

$$G = -\frac{j}{4} H_0^{(2)}(kR) \tag{5.26}$$

Here, $H_0^{(2)}$ is the Hankel function.

Detecting whether a point is inside or outside the scatterer can be an annoying problem in special geometries, specially in the presence of casuistics. This is not the case, however, with the geometries used, in which a sensible unique inequality can be used.

5.1.6 2D Geometry section

Three different geometries have been used, although most of the work was initially performed with a 2D circular section (cylinder in 3D).

Working with a circular geometry from the beginning has its pros and its cons. It simplifies enormously some parts of the work and is of big help in developing an intuition with the results. However, being a super symmetric special case as it is, some (usually not good) surprises can take place, as was the case.

After the circular geometry, the developed sections were

- *Elliptical*: this geometry already includes the circular case. The generated ellipses semiaxis are parallel to the x and y axis.
- *Rectangular*: rectangular geometry, specially interesting to see the effects of close elements in the discretization with very different subindex numbers. Also interesting for the study of the effects of boundary smoothness discontinuities at the vertices.

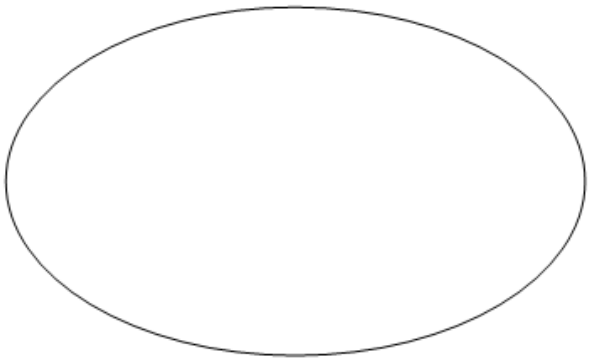

Figure 5.4: Boundary of an elliptical geometry

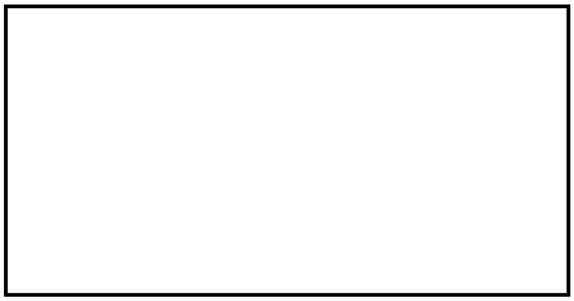

Figure 5.5: Boundary of a rectangular geometry

- *Superelliptical*: also called "Pill" geometry, for its resemblance. This is an intermediate case between a rectangle and an ellipse, as an ellipse of higher than 2 degree. It has the further interest of having equispaced elements in angles. The degree of the superellipse can be modulated to approach the ellipse or the rectangle more closely.

5.2 Outline of the code

The simulations follow, with slight differences depending on the input parameters, the same set and order of instructions.

An outline of the operations for the 2D IE-MEI method is here described:

1. Define and declare the physical constants (type of media, size, etc.).
2. Set axes handles.

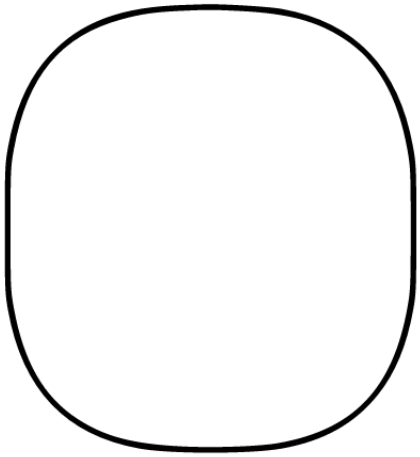

Figure 5.6: Boundary of a superelliptical geometry

3. Define which scatterer geometry will be used, its associated vectors and the incident field.

4. Calculate elements of the matrices (coefficients of the EFIE and MFIE).

5. With these equations, calculate the IE-MEI matrices $[A]$ and $[B]$, with the specified number of elements and zeros and apply the corresponding threshold.

6. Solve the problem by inverting the obtained sparse system.

7. Calculate the scattered field by integrating the found electric and magnetic currents with Green integrals.

8. Plot the results and print the data of the simulation.

Chapter 6

Graphical user interface

Initially, all the commands where executed from the `MATLAB` command line. As the complexity, number of variables and number of routines increased, though, more and more time was lost in human errors product of bad typing, swapping of input variables, increase of produced plots... that could be easily avoided by using a more intuitive and visual interface. This is why a graphical user interface (GUI) was developed.

This GUI was later divided in two different programs: one, for a clean MoM execution, used mainly as a 2D simulator and aimed at visual representation of the found solution; the other, for IE-MEI method testing, with all the parameters used in the simulation and method and a comprehensive log with the results, errors, etc.

6.1 2D Scattering simulator

6.1.1 Description of the GUI

The graphical user interface allows the user to work with the simulator without having to deal with the details; it can be called by writing "`gui_iemei_nn`" from `MATLAB`'s command line. The GUI is composed of three sections; on the upper left, the parameters of the problem can be specified:

1. N: the number of elements in which the scatterer is going to be discretized. The higher the number, the more precision in the solution, but also the longer it will take to simulate. As a rule of thumb, N should fulfill the following condition

$$N \geq 10k_{\text{scat}}r = 10\sqrt{\epsilon_r}r \simeq 30 \cdot k_0 r \tag{6.1}$$

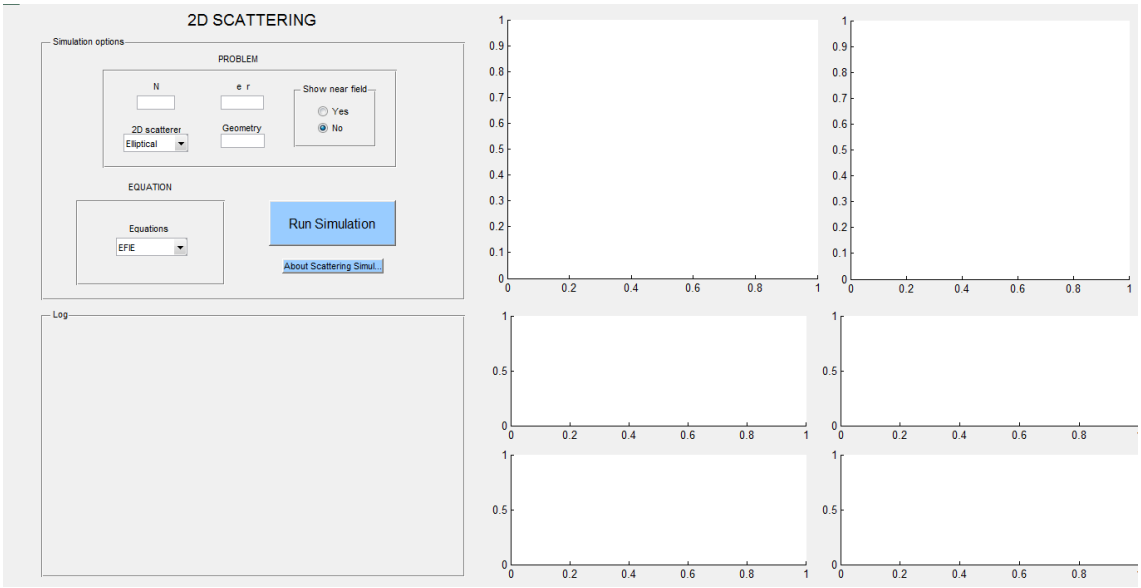

Figure 6.1: Clean GUI after executing from the command line

where k_{scat} is the wavenumber inside the scatterer and k_0 is the wavenumber in vacuum. Here, $|\sqrt{\epsilon_r}| \simeq 3$ for the values commonly encountered in plasmonic materials, but this could change to more strict conditions in materials with higher permittivity.

2. ϵ_r: the relative permittivity of the scatterer (ratio of permittivity to the permittivity of vacuum). This parameter controls "the material" of which the scatterer is made, and can be complex. In non magnetic materials such as this, $n = \sqrt{\epsilon_r}$, where n is the refractive index. A permittivity of 1 means the scatterer will be made of thin air and there will be no scattering at all (this can be tested to prove the code is correct), a permittivity of around 3 will simulate a glass and a permittivity of negative real and imaginary part (e.g. $-1000 - 400j$) will simulate a metal, where there is no penetration of the fields inside the scatterer.

3. 2D scatterer: the geometry of the scatterer can be chosen:

   - Elliptical (including circular case)
   - Rectangular
   - Pill: this geometry is strictly a superellipse (transition case between the other geometries).

4. Geometry parameters: a vector input of the form (a, b). The first parameter, a is

$k_0 r$ in the X axis: the value of $k \cdot r$. If the selected geometry is elliptical, it will be the semiaxis of the circle in this direction; if the selected geometry is rectangular, it will be half the side in the x axis. The second parameter, b, acts exactly the same but in the Y axis. The GUI also excepts only one parameter (the second will be assumed to be the same).

5. Show near field: the option to calculate the near field or only the currents in the surface. The convolution integrals involved in the calculation of the near field is the most time consuming of all the operations.

6. Method: EFIE or MFIE equations can be chosen. The solutions with either method should be almost the same except in extreme cases (like resonances).

7. Run: initiate the simulation once the rest of the parameters have been set.

8. About Scattering Simulator: shows help, sources and other diverse information in the log section.

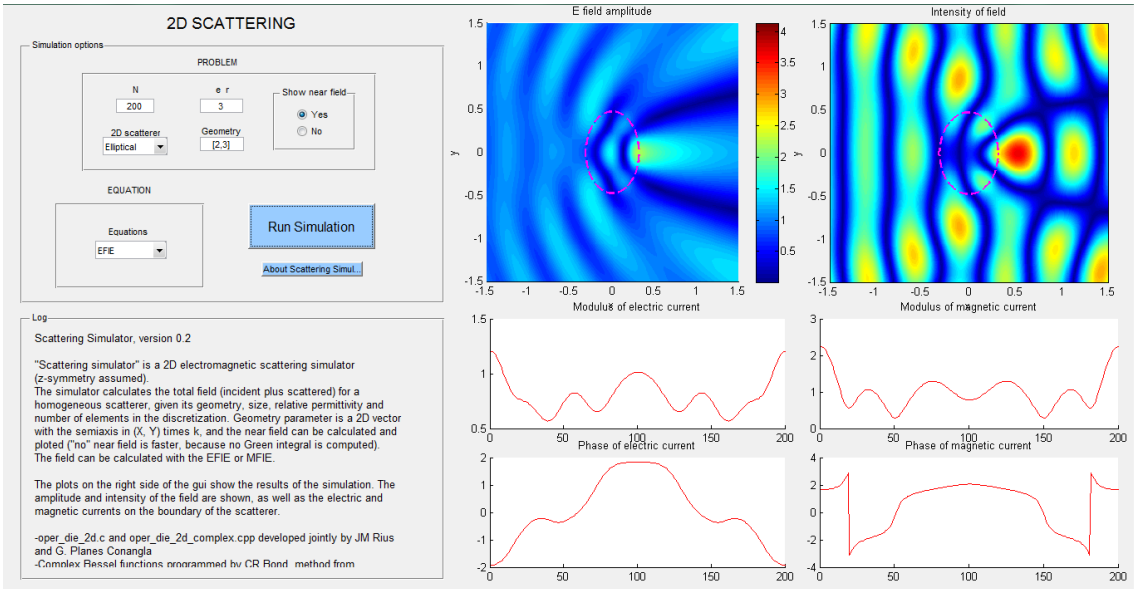

Figure 6.2: GUI after a simulation (elliptic metallic scatterer)

Tooltips have been included to give advice when passing by over a field with the mouse.

The other two sections are "Log" (in the lower left section of the gui), where information regarding the simulation is printed once the calculations are finished, and the right

section of the gui, where the plotting is performed and six figures are shown. From left to right and from top to bottom these are:

1. Amplitude of the E field (inside and outside the scatterer). The silhouette of the scatterer is shown in pink dashed lines.
2. Intensity of the field, which shows an animation of the permanent sinusoidal regime and the scattering for 5 complete periods.
3. Modulus of the electric current around the scatterer.
4. Modulus of the magnetic current around the scatterer.
5. Phase of the electric current around the scatterer.
6. Phase of the magnetic current around the scatterer.

A picture of the gui after a simulation is shown in figure 6.2

6.1.2 A guided example

Fill the parameter boxes with the following numbers:

- Put '300' in the number of elements box.
- Put '3' in the relative permittivity box (glass material).
- Write '[3,3]' in the $k_0 r$ section (radius of size $R \simeq \lambda$).
- Select 'Rectangular' geometry option.
- Mark 'yes' option in the Show near field rectangle.
- Select EFIE equations.
- Click on 'Run Simulation'

Once the simulation is done, the times spent in each step and the values used will appear in the 'Log' section. Graphs of the modulus of electric and magnetic current, its complex value and the total field (incident plus scattered) will be plotted at the right side of the GUI. The result should look like figure 6.3.

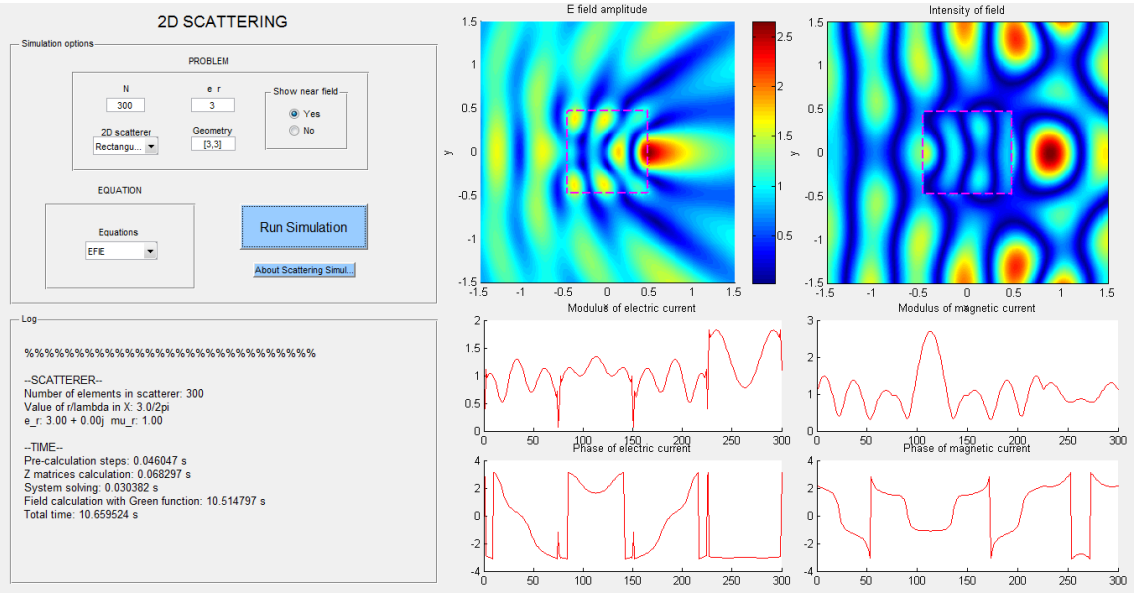

Figure 6.3: GUI after a simulation (square glass scatterer)

6.2 IE-MEI NxN GUI

6.2.1 Description of the GUI

As in the previous case, the GUI is composed of three sections; on the upper left, the parameters of the problem are specified. The ones for the 'PROBLEM' section are the same, and the IE-MEI parameters are set in the 'IEMEI' rectangle:

1. N: the number of elements in which the scatterer is going to be discretized.
2. ϵ_r: the relative permittivity of the scatterer (ratio of permittivity to the permittivity of vacuum).
3. 2D scatterer: geometry of the scatterer:
   - Elliptical
   - Rectangular
   - Pill
4. Geometry parameters: a vector input of the form (a, b). The first parameter, a is $k_0 r$ in the X axis; b is kr in the Y axis. In the rectangle case, both a and b are half the sides of the rectangle.

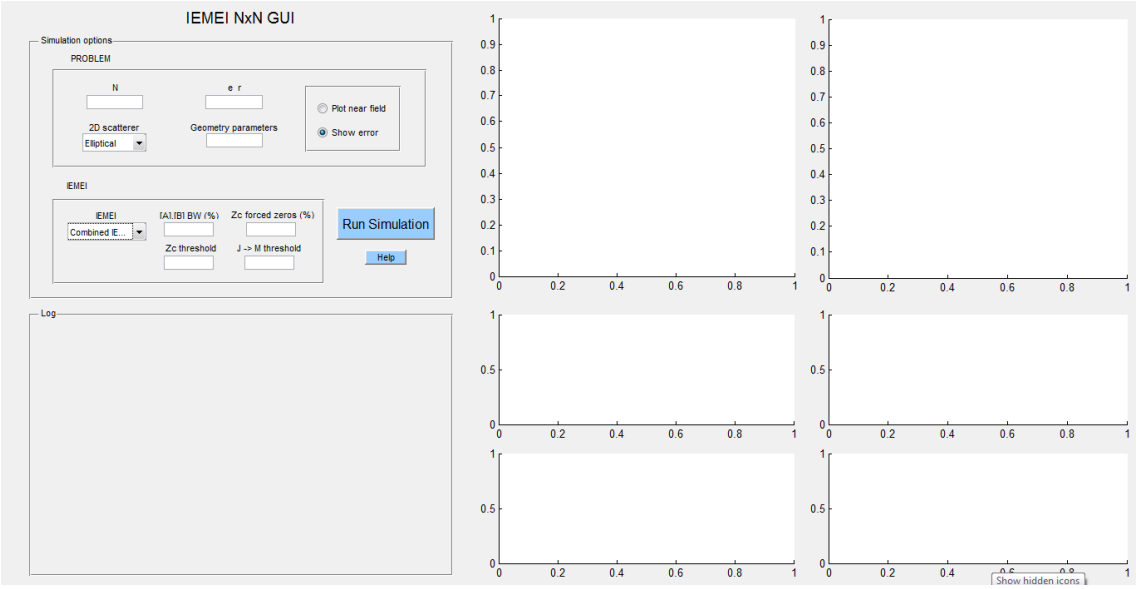

Figure 6.4: Clean GUI after executing from the command line

5. Plot near field: the option to calculate the near field or only the currents in the surface. Otherwise, it will plot the error in the elements.

6. IEMEI: single EFIE or MFIE IEMEI method or combined IEMEI method. In the first or second case, only $[A]$ or $[B]$ matrices, with the EFIE or the MFIE equations, are used. With the combined IEMEI both are used, and thus also both the EFIE and MFIE equations are calculated.

7. $[A][B]$ BW(%): percentage of the number of elements that will be used in the $[A]$, $[B]$ matrices. The rest will be set to zero.

8. Zc forced zeros (%): percentage of elements of the $[Z^C]$ matrix that will be set to zero, outside of the diagonal. The zeros will be distributed to best minimize the effect of the elements without constraints.

9. Zc threshold: threshold below which all elements will be forced to zero after multiplication by $[A]$ and $[B]$.

10. J $\rightarrow$ M threshold: threshold below which all elements will be forced to zero in the transformation matrix from $\vec{J}$ to $\vec{M}$. In lossy dielectrics, this transformation matrix is almost diagonal.

11. Run: initiate the simulation once the rest of the parameters have been set.

12. Help: shows help and diverse information in the log section.

Here, tooltips have also been included to give advice when passing by over a field with the mouse.

The other two sections are "Log" (lower left section of the gui), where information regarding the simulation is printed once the calculations are finished, and the right section of the gui, where the plotting is performed and six figures are shown. From left to right and from top to bottom these are:

1. Near field/Error per element in the geometry
2. Resultant $[Z^C]$ matrix. Useful to see the performance of the IE-MEI method.
3. Modulus of the electric current around the scatterer.
4. Modulus of the magnetic current around the scatterer.
5. Phase of the electric current around the scatterer.
6. Phase of the magnetic current around the scatterer.

6.2.2 A guided example

Fill the parameter boxes with the following numbers:

- Write '300' in the number of elements box.
- Put '-3-4j' in the relative permittivity box (plasmonic material).
- Write '[5,5]' in the Geometry parameters section.
- Select 'Elliptical' geometry option.
- Mark 'Show error' option in the Show near field rectangle.
- Select 'Combined IEMEI' equations.
- Put '4' in $[A][B]$ BW(%).
- Put '9' in Zc forced zeros.
- Write '1e-6' in the Zc threshold area.

- Write '5e-2' in the J→ M area.
- Click on 'Run Simulation'

Once the simulation is done, the times spent in each step and the values used will appear in the 'Log' section. Graphs of the modulus of electric and magnetic current, its complex value and the total field (incident plus scattered) will be plotted at the right side of the GUI. The result should look like figure 6.5.

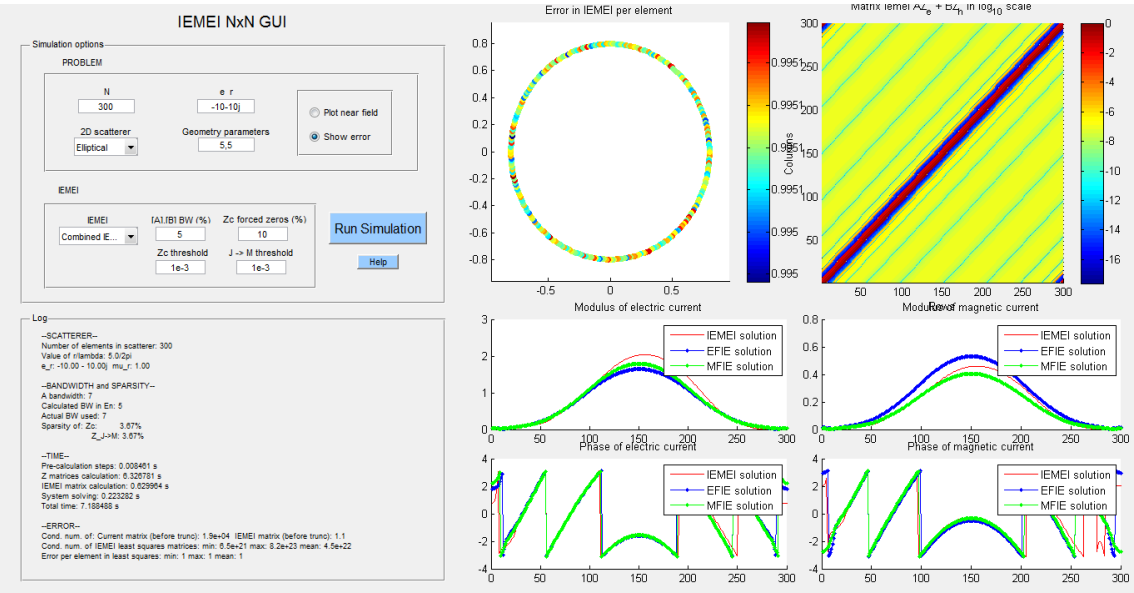

Figure 6.5: IE-MEI GUI after a simulation (square glass scatterer)

Chapter 7

Results

In this chapter the results of the studies and implemented methods are presented. The performance of the methods is compared with its equivalent pure MoM simulations and generally refer to results of the equivalent current in modulus and phase form, with some added figures and comments about the behaviour of the field.

All the incident fields have been set with TM polarisation, and this fact will be taken for granted in the whole section.

7.1 General considerations

The results of the MoM show that for dielectric simulations, the discretization step often used for boundary element methods for PEC scatterers of about $\lambda_0/10$ is not enough. Generally, a minimum of half this discretization size, $\lambda/20$ is required, and usually even more. In extreme cases, like with plasmonic resonances, a specially thin meshing is of maximum importance. Usual discretization sizes used throughout the work belong in the range of $\lambda_0/20$ to $\lambda_0/40$.

No IE-MEI method proved able for general real positive permittivities. In any of the formulations used, after setting the number of not forced to zero elements per row in the resulting $[A][Z_E] + [B][Z_H]$ matrix (i.e., the sparsity of the matrix), the results were reasonable only for sparsities in the range of 30% - 40% and higher, not justifying the application of the method. For smaller sparsities, spurious oscillations and noise appeared on the results. This explains the lack of publications on the sparsification of the MEI or IE-MEI methods to dielectric scatterers.

Hence, the study has focused on complex permittivites with negative imaginary part[1]; that is to say, media with exponential decay on the fields. If the real part of this permittivity is smaller than 1 (generally negative values), these materials can be classified as *plasmonic*. They can have a big negative imaginary part and therefore rapidly decaying interior fields; the limit case is a perfect conductor, in which no fields at all exists inside the scatterer. If the imaginary part is higher than 1, they can model lossy dielectrics, or real dielectrics, such as water or glass, and they tend to have small absolute values of imaginary part, usually negligible.

This can be seen in figures 7.1, 7.2 and 7.3.

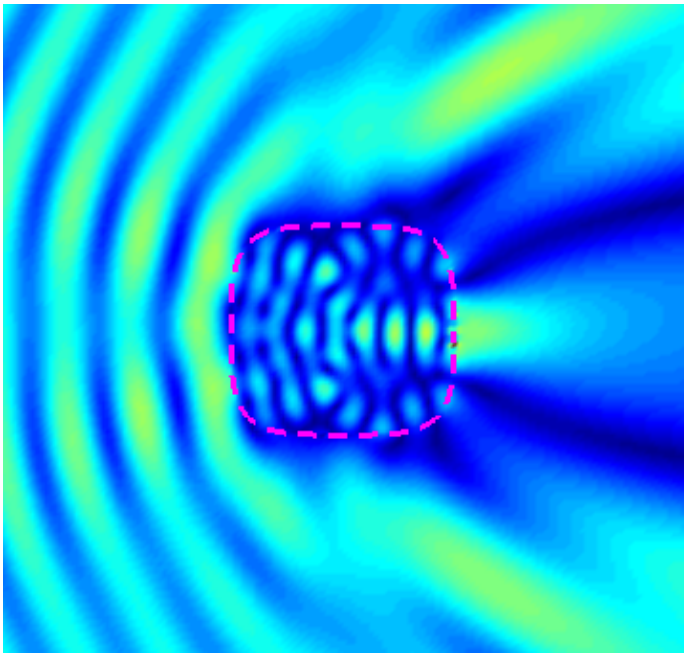

Figure 7.1: Complete penetration in a dielectric without decay, $\epsilon_r = 6$

In permittivites with negative imaginary part, the mutual dependence decays approximately as a negative exponential (this depends, of course, of the exact geometry of the scatterer), so a good sparsification, followed by a truncation appears plausible. The application of the IE-MEI in this case has been more succesful, with high sparsities obtained.

[1] See [9]

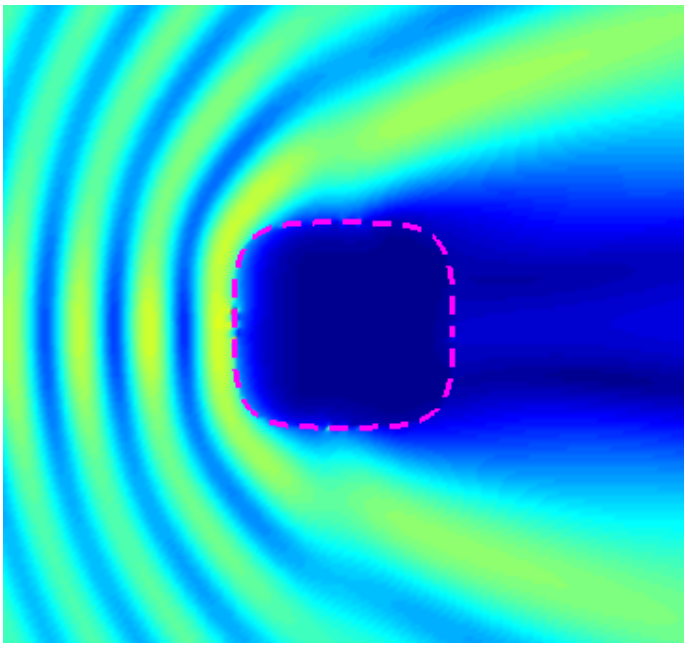

Figure 7.2: Simulation of a plasmonic material, $\epsilon_r = -3 - 5j$. There is some field penetration

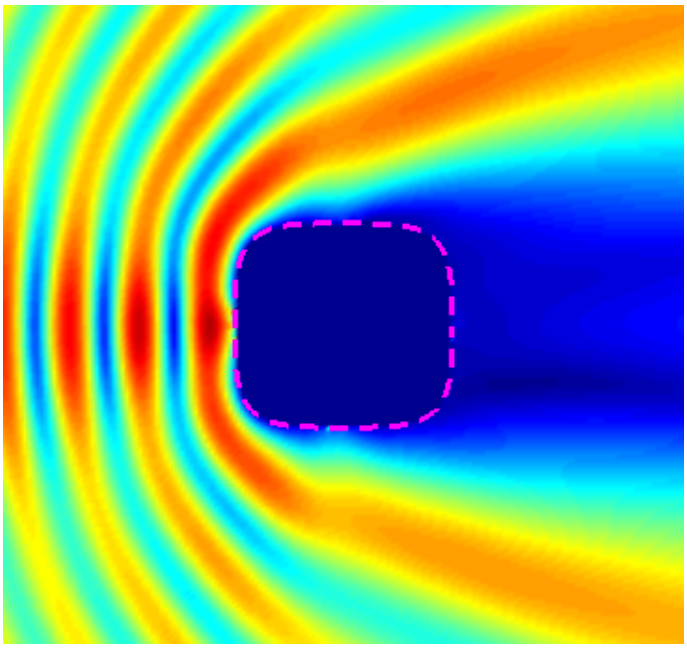

Figure 7.3: Simulation of a PEC scatterer. There is no field penetration

7.2 2Nx2N IE-MEI

Several options have been successively considered for the $2N \times 2N$ version of the IE-MEI method:

1. Single band circulant diagonal matrices $[A]$ and $[B]$ and the impedance matrix $[Z]$ of the same kind. This was rapidly discarded, as no good solutions could be obtained. The main reason for this was the inability to cancel the fields by using only one of the equivalent currents (electric or magnetic).

2. Single band circulant diagonal matrices $[A]$ and $[B]$ and triband impedance matrix $[Z]$, as the logical next step to the previous option. This notably improved the results, although not sufficiently. A similar problem still remained: with single band $[A]$ and $[B]$ there was no combination of $[Z_E]$ and $[Z_H]$ matrices, as only one of the boxes[2] was used.

3. Both the IE-MEI matrices $[A]$ and $[B]$ and the impedance matrix $[Z]$ of triband form. With this choice of matrix forms the results followed quite closely the MoM results for electrically small scatterers. Although the computaitonal cost of straightforward MoM is small and oes not require matrix sparsification, IE-MEI can be very useful to compress near field submatrices of MLFMM, that correspond to multilevel subdivision domains of small electrical size. This was the last version of the $2N \times 2N$, and was tested quite extensively with a circular scatterer.

Once the matrix form was set, different configurations for the simulations were tried, and the $2N \times 2N$ matrix (to be inverted) consistently presented very big condition numbers[3]. Thus, the solutions always had noisy/oscillating appearances.

The reasons were clarified soon after: the mix of 4 different matrices in the boxes resulted in very different matrix element relative magnitudes. This problem was attacked with some different but elementary preconditioners, and the technique of adding a small identity matrix multiplied by a Marquardt-Levenberg coefficient was also attempted. No solution was found for this. Instead, the $N \times N$ scheme was used.

[2]Reminder: in the $2N \times 2N$ system, the matrix to be inverted is the sum of 4 smaller matrix boxes.

[3]The condition number of an A matrix is $\kappa(A) = \left\|A^{-1}\right\| \cdot \|A\|$. The condition number measures how much the output value of the function can change for a small change in the input argument. This is used to measure how sensitive a function is to changes or errors in the input, and how much error in the output results from an error in the input.

7.3 NxN IE-MEI

The $N \times N$ scheme was a modification of the previous, with the aim of getting better condition numbers. In this case, the used matrices were clearly of the single band type, as only one type of current was used. Both $\vec{J}$ and $\vec{M}$ were used and compared between one another and no differences in the performance of the results was found. As previously indicated, to transform one current into the other, the equations

$$\vec{M} = -[Z_{em2}]^{-1}[Z_{ej2}]$$

in the EFIE case or

$$\vec{J} = -[Z_{hj1}]^{-1}[Z_{hm1}]\vec{M}$$

in the MFIE case could be used. In the studied permittivities, with negative imaginary part, the matrices $[Z_{em2}]$ and $[Z_{hj1}]$ were almost diagonal, even in elongated cases (like rectangles and ellipses), and truncation made the matrix very sparse. Usually, the threshold for truncation was set in the range $5 \cdot 10^{-3}$ - 10^{-4} (the number representing the relative size of the element to be truncated over the biggest element in the row). This scheme is not useful when the dielectrics are not lossy, because the matrix is not sparse and the extra inversion outweighs the motives for calculating the coefficients of the IE-MEI matrices (however is a fine application of the clean MoM method).

The results show that it's possible to make $[Z_C]$ sparse after multiplication with the IE-MEI matrices. However, the sparser the matrix, the more the variables needed and thus, more coefficients in the IE-MEI $[A]$ and $[B]$ matrices, making these matrices less sparse. Initial tests with no zero distributing showed an optimum performance at sparsities of $[A]$ and $[B]$ around 30% - 40%. For sparsities lower than these figures, the least squares matrices in the IE-MEI method gave bad results because of ill-conditioned matrices. Increasing the number of forced zeros and, thus, the dimensions of the matrices, started producing –paradoxically– worse results than with less forced zeros. The condition number, therefore, has been observed to increase to a point where forcing more zeros is counterproductive.

A solution to this problem was found in the reduced zero forcing strategies discussed in 5.1.4. By forcing a well distributed number of zeros among the rows of $[Z_C]$, the need for zeros decreased, because the positions where a zero was forced (in the least squares sense) had a limiting effect in the size of the adjacent positions: *less zeros are required if they are well distributed*, according to the heuristic principles described[4].

[4]Theoretical work would be needed to correctly explain this fact.

By testing with several zero distribution techniques, it was found that there was a direct relation between the height (row elements peak magnitude) of the EFIE and MFIE (any one of the two, although here the EFIE was mainly used) matrices (considering these matrices as two dimensional function defined on a grid) and the number of zeros required in that zone after multiplication with the IE-MEI matrices to make the height decrease sufficiently and be able to consider the peak negligible. The matrices used to decide the distribution do not necessarily have to be these ones; one taking into account the geometry (distance and orientation between elements) was also tested, but it gave poor results. However, more work could be done in this direction, because using the EFIE and MFIE *requires* calculating the EFIE and MFIE. In 2D, where the ordering of the elements also implies its position in the scatterer, the calculation of only a few elements and interpolation of the rest can be used to reduce the computer cost, but in 3D the elements can be anywhere.

For PEC scatterers, forcing a smaller number of zeros had only been done in bands immediately after the diagonal band (neighbourhood of the band). This has been shown to be improvable, because the zeros in the "middle" tended to form a peak that was often too big. Three different bands (one in the middle) was essayed later, with now two peaks at the zones where no zeros were forced. From here, the logical steps to follow were moving to a higher number of bands and finally to equidistributed zeros, with better and better outcomes. The results for the circular geometry in this case were very good, although a "concentrated" numbers of zeros close to the diagonal of about a 30% of the zeros gave a better performance.

Of course the circular case doesn't have much interest here. The described inverse method is interesting for more complicated geometries, where equidistributing zeros is not the best option. The ellipse, the rectangle and its intermediate step the superellipse, have also been tested. With the inverse function method for distributing zeros giving excellent results in the sparsification of matrices. Usually, only a few percent (2-5%) of the zeros had to be forced to obtain matrices with a big band diagonal and the rest of the elements (forced zeros and elements without constraints but close to the zeros) several orders below. The same effect as before persisted, though: when increasing the size of the scatterer and, therefore, using a higher number of elements, ill conditioned matrices appeared in the least squares method, sometimes to extreme points where the resulting matrices $[A]$ and $[B]$ were not of maximum rank. Specially prone to ill conditioned matrices were the very elongated objects, where elements in the center of the bar where

highly dependent on all the rest of elements (in particular to the ones at the other side of the bar). This elements contributed the most part of the error in the solution. In the future it would be interesting to implement a function able to force a variable number of zeros (now the number of zeros is constant in all the rows of the matrix); in this case, elements such as the ones described would "receive" more zeros than others.

All the work has been done with delta metrons (the most easy to export to 3D geometries). In PECs, harmonic metrons had been used successfully before in 2D scatterers, although these are not generalizable to 3D scatterers; they were also tested here with dielectrics. In contrast with delta metrons, they didn't produce sparse enough matrices. The "limited" zero forcing strategy gave very poor results, to the point where truncation was not applicable, and the use of delta metrons was resumed.

7.4 Zero-forcing schemes figures

A comparison of the different strategies for forcing zeros is shown in 7.1.

kR	N	ϵ_r	Zero forcing strategy	Sparsity (%)	δ_r
10	300	-3 -5j	only on neighbour bands	16	3,9558
			bands and center of zero region	16	0,044287
			5 equal bands	16	0,006641
			5 bands with bigger bands near diagonal	16	0,006472
			inverse method distributiong	16	0,006103

Table 7.1: Comparison of different zero forcing strategies, for a constant number of forced zeros, and its relative error in equation currents versus MoM

Figures 7.4 to 7.14 present the found distribution of forced zeros for a circle, ellipse, superellipse and a rectangle with the inverse function method. They are presented in order, to better perceive the effects of a geometrical deformation on the distribution. It can be seen, as stated, that the number of zeros per row is constant. A small band with 30% of the zeros was kept close to the diagonal, to better "force down" the zeros close to the diagonal peak.

Also, some examples of zero forcing matrices are included too. It can be seen how in elongated geometries, some rows have higher values than others: this happens because the element in the diagonal of that row has a higher impedance dependence with the other

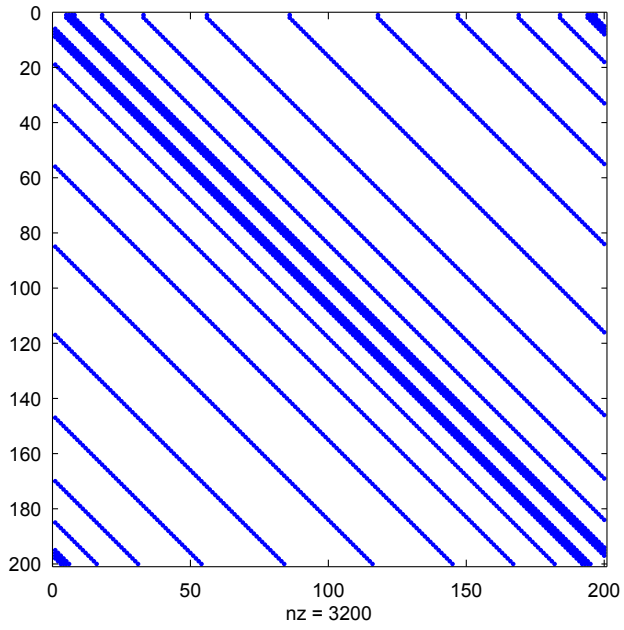

Figure 7.4: Position of the forced zeros in a circular scatterer geometry

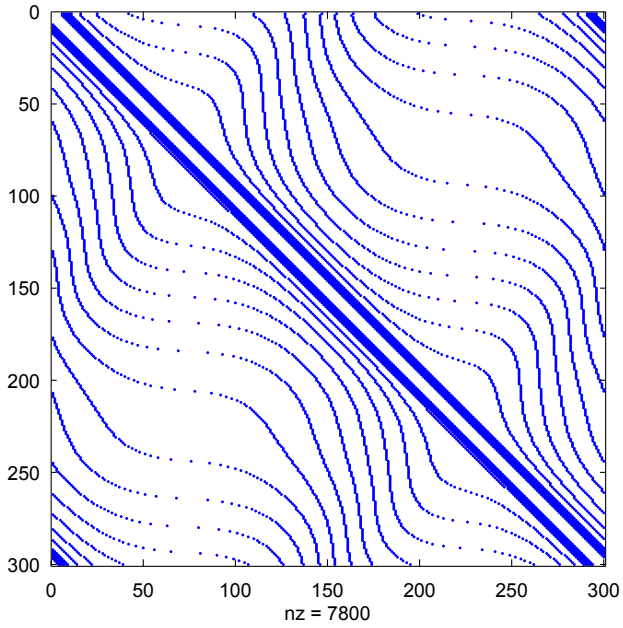

Figure 7.5: Position of the forced zeros in an elliptical scatterer geometry

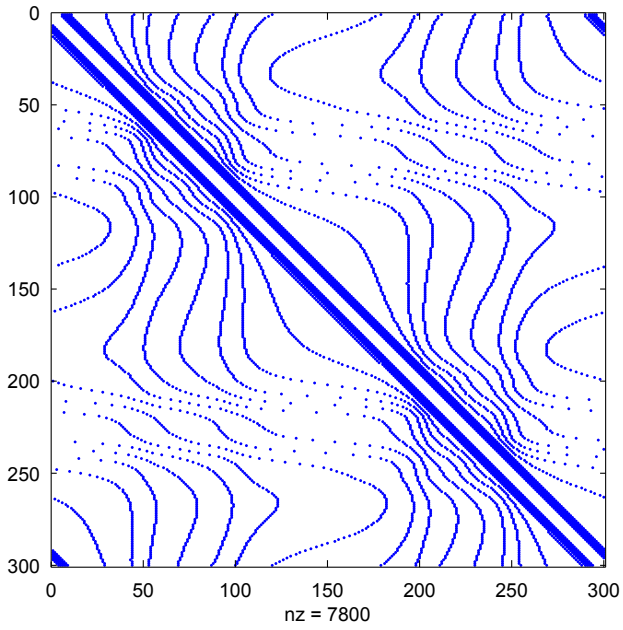

Figure 7.6: Position of the forced zeros in a superelliptical scatterer geometry

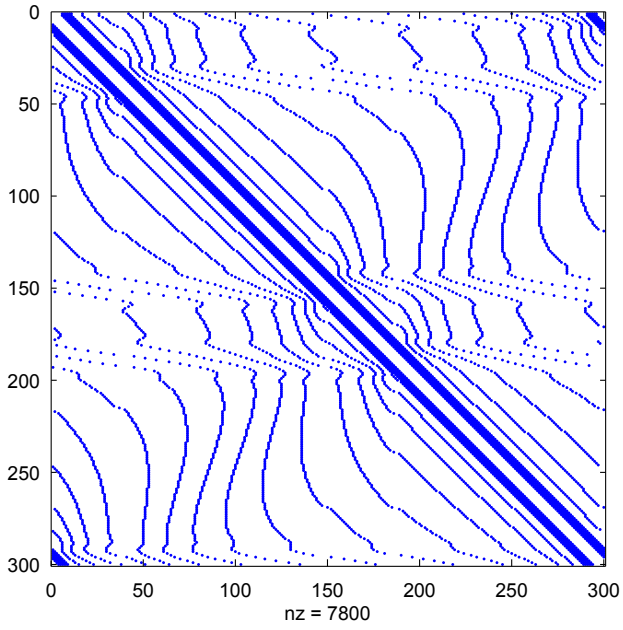

Figure 7.7: Position of the forced zeros in a rectangular scatterer geometry

elements. More zeros[5] could be used in this case.

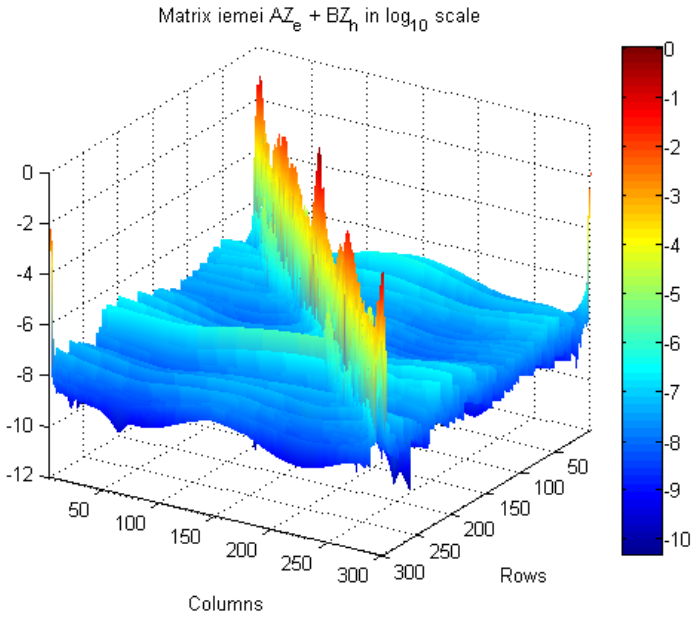

Figure 7.8: Forced zeros in an elliptical scatterer, $N = 300$, sparsity 5%, $\epsilon_r = -3 - 5j$

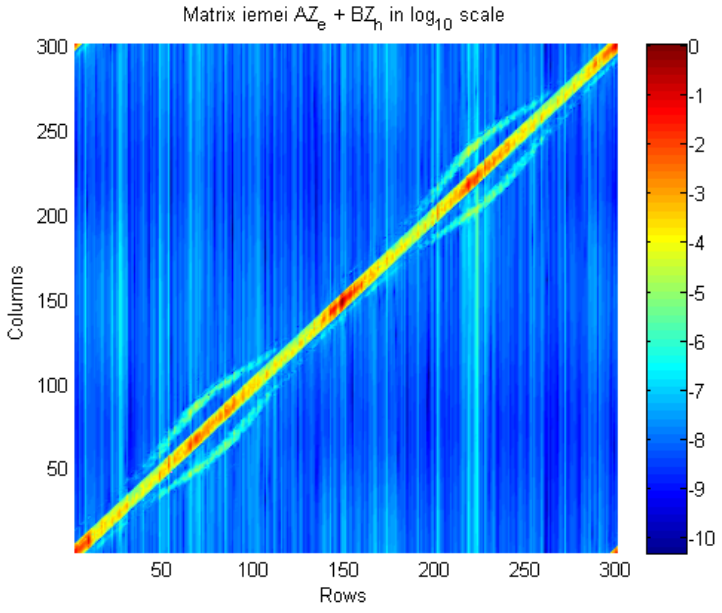

Figure 7.9: Forced zeros in an elliptical scatterer, $N = 300$, sparsity 5%, $\epsilon_r = -3 - 5j$. Superior view

[5]As in a variable number of zeros strategy

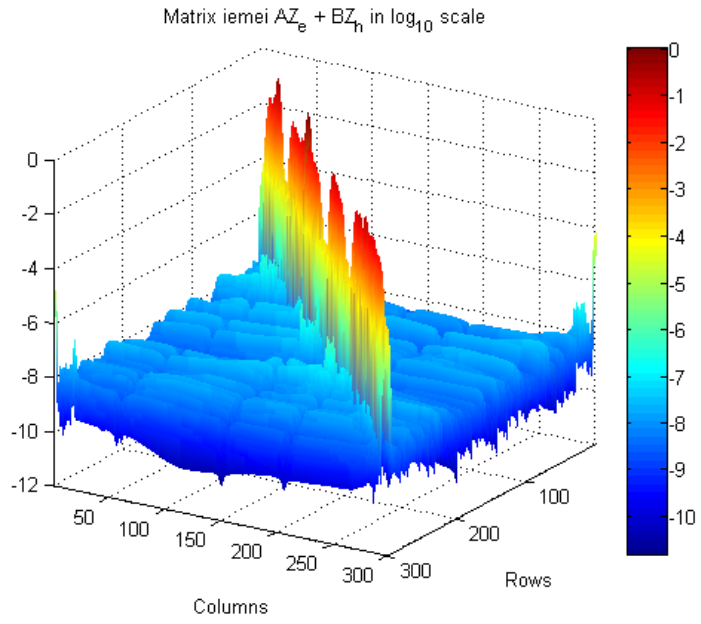

Figure 7.10: Forced zeros in a rectangular scatterer, $N = 300$, sparsity 5%, $\epsilon_r = -3 - 5j$

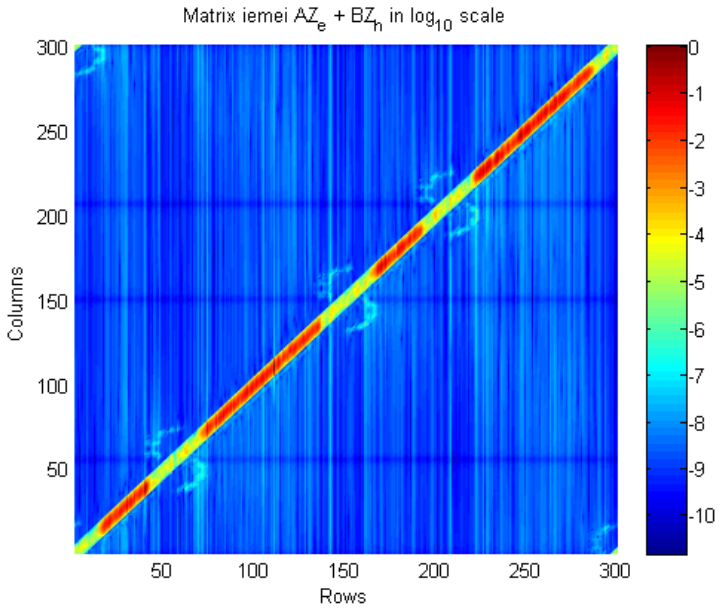

Figure 7.11: Forced zeros in a rectangular scatterer, $N = 300$, sparsity 5%, $\epsilon_r = -3 - 5j$. Superior view

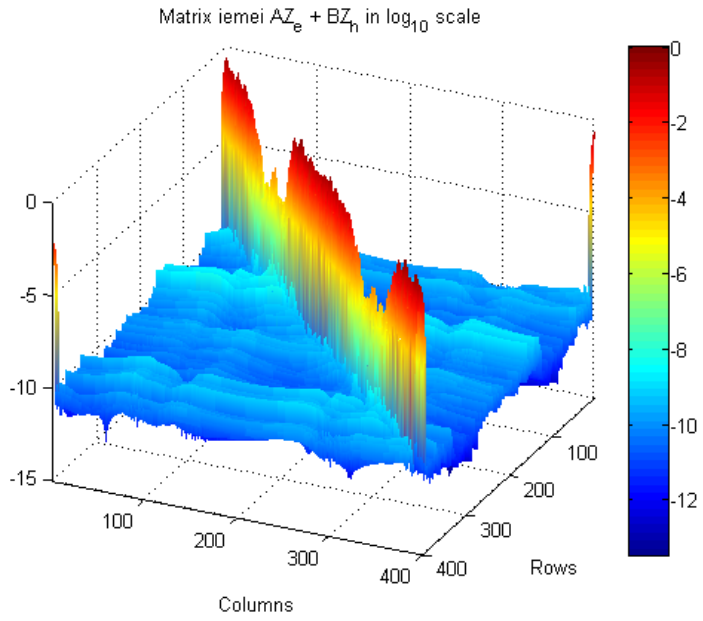

Figure 7.12: Forced zeros in a superelliptical scatterer, $N = 400$, sparsity 5%, $\epsilon_r = -3-3j$, kR semiaxis of $(3, 9)$

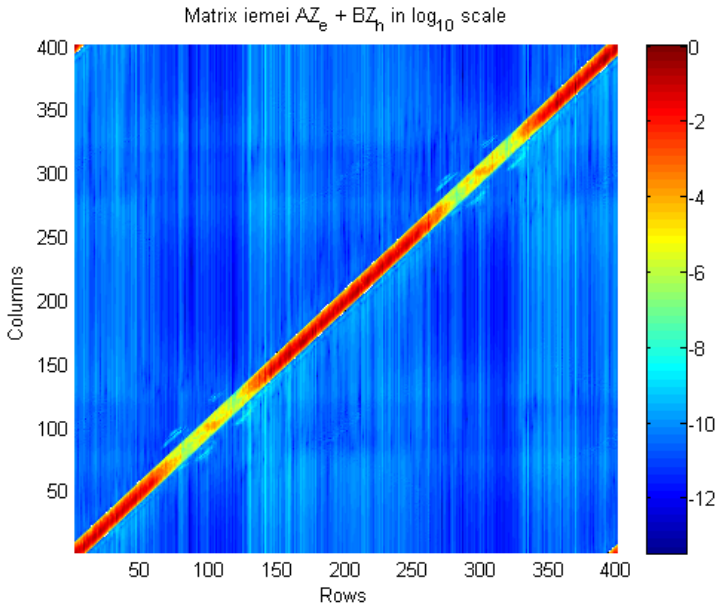

Figure 7.13: Forced zeros in a superelliptical scatterer, $N = 400$, sparsity 5%, $\epsilon_r = -3-3j$. Superior view

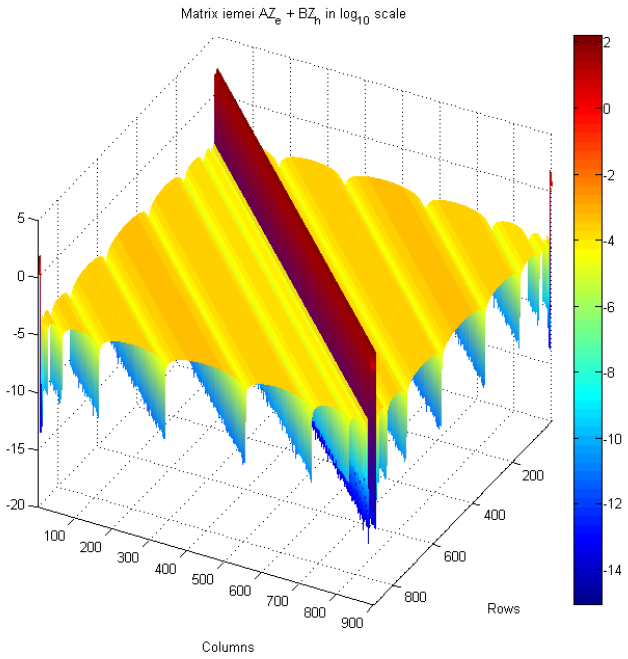

Figure 7.14: Forced zeros in a circular scatterer, $N = 900$, sparsity 2.33%, $\epsilon_r = -5.84 - 2.11j$ (gold $\lambda_0 = 546$ nm)

7.5 MoM vs IE-MEI figures

In table 7.2, a comparison of the obtained quadratic error in $\vec{J}$ and $\vec{M}$ between the MoM and IE-MEI for different thresholds is displayed. The scatterer is circular and a permittivity of $-3-3j$, corresponding to a plasmonic material, is used. The table shows that after a certain threshold is exceeded, the relative error stops decreasing. This is due to the comparison between the MoM results and the IE-MEI method: to calculate the error, one of the two equations (EFIE or MFIE) have to be chosen, whereas the IE-MEI uses both. Because the EFIE and MFIE are different equations, a certain distance between them always exist and can't be overcome. Once the IE-MEI produces errors of the same order, the solution can't be further improved. The forced and obtained sparsity values[6] are included.

The threshold in the currents transformation has been taken at 10^{-3} in the whole section, whereas truncation of the $[Z_C]$ matrix is performed at 10^{-4}. All the incident fields are 0° with TM polarization.

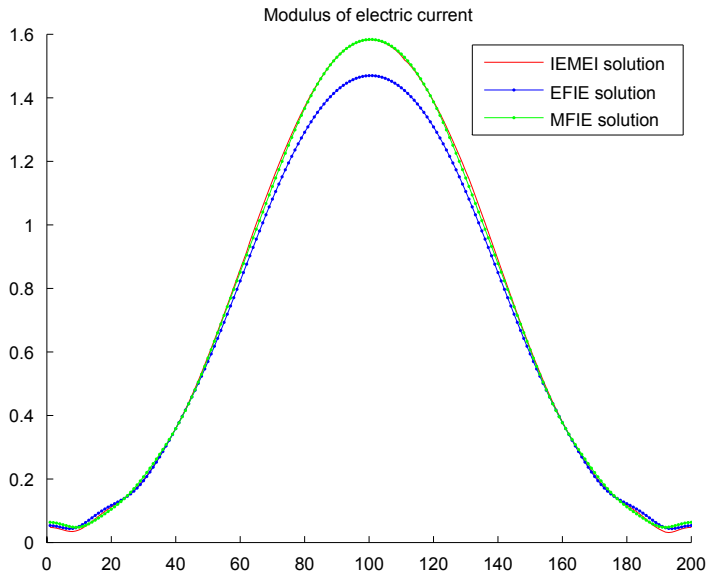

Figure 7.15: Modulus of $\vec{J}$ in circular scatterer, $N = 200$, $kR = 4$, sparsity 3%, $\epsilon_r = -3-5j$

[6]Once a *wanted* sparsity is set, least squares are applied, so the zeros are not exactly zeros. A truncation threshold is applied, which will give a slightly different sparsity.

k_0r	N	ϵ_r	Desired $[Z_C]$ sp. (%)	zeros (%)	Obt. $[Z_C]$ sp. (%)	e_{thr}	δ_r
10	300	-3 - 3j	35	52	27,43	10^{-4}	4,40513
					27,63	10^{-5}	0,261416
					27,67	10^{-6}	0,008002
					27,67	10^{-7}	0,008002
					27,67	10^{-8}	0,008002
10	300	-3 - 3j	35	36	27,63	10^{-4}	0,820586
					27,66	10^{-5}	0,127847
					27,67	10^{-6}	0,008711
					27,67	10^{-7}	0,008711
					39,05	10^{-8}	0,008153
10	300	-3 - 3j	25	37,5	19,42	10^{-4}	1,278478
					19,66	10^{-5}	0,054065
					19,67	10^{-6}	0,008006
					19,67	10^{-7}	0,008006
					19,67	10^{-8}	0,008006
10	300	-3 - 3j	25	18,75	19,5	10^{-4}	0,64752
					19,67	10^{-5}	0,020954
					19,67	10^{-6}	0,020954
					28,79	10^{-7}	0,016806
					56,88	10^{-8}	0,00808
10	300	-3 - 3j	15	12,75	11,55	10^{-4}	0,474551
					11,66	10^{-5}	0,008734
					11,67	10^{-6}	0,008251
					12,9	10^{-7}	0,008248
					46,17	10^{-8}	0,008066
10	300	-3 - 3j	15	12,75	11,64	10^{-4}	0,244069
					11,67	10^{-5}	0,020197
					11,7	10^{-6}	0,020153
					43,16	10^{-7}	0,008584

Table 7.2: Obtained sparsities (sp.) for a circular scatterer in three-band zero distribution. After the problem parameters are set, a different truncation threshold is applied, which can modify the sparsity of the matrix. In the last column, δ_r, relative error between MoM and IE-MEI formulation. The best results are obtained for a desired sparsity of 15% and thresholds of 10^{-5} or 10^{-6}

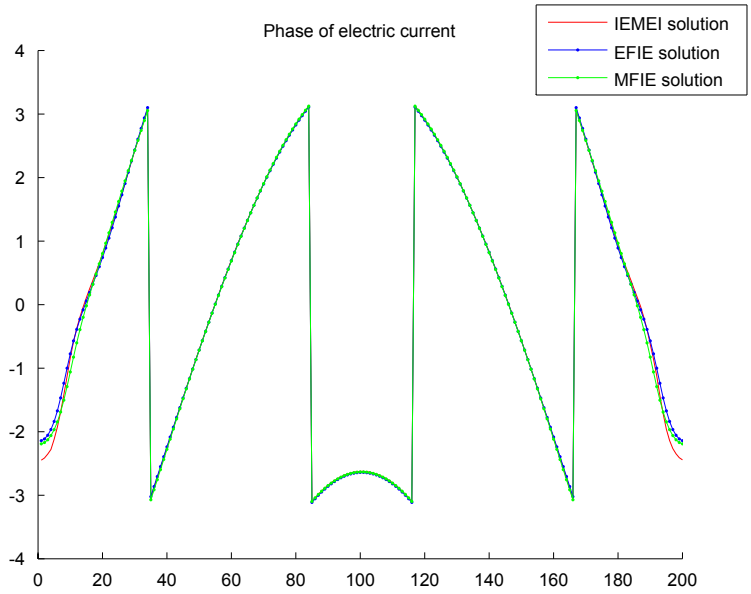

Figure 7.16: Phase of $\vec{J}$ in circular scatterer, $N = 200$, $kR = 4$, sparsity 3%, $\epsilon_r = -3 - 5j$

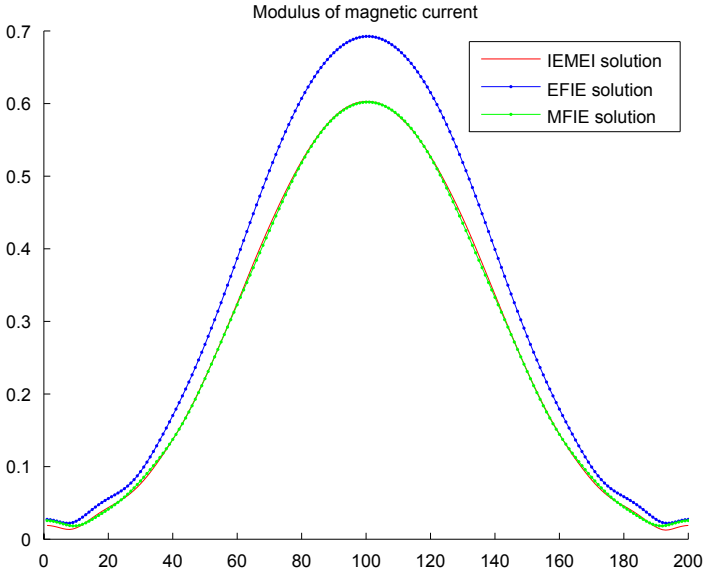

Figure 7.17: Modulus of $\vec{M}$ in circular scatterer, $N = 200$, $kR = 4$, sparsity 3%, $\epsilon_r = -3 - 5j$

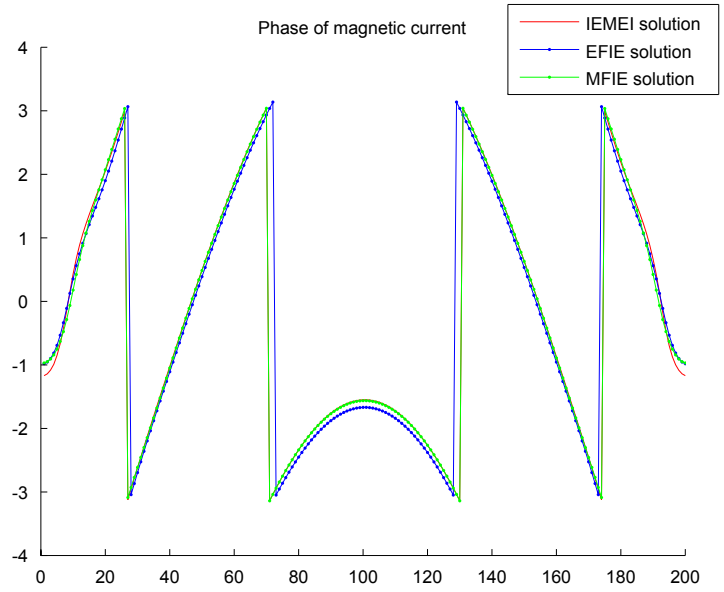

Figure 7.18: Phase of $\vec{M}$ in circular scatterer, $N = 200$, $kR = 4$, sparsity 3%, $\epsilon_r = -3-5j$

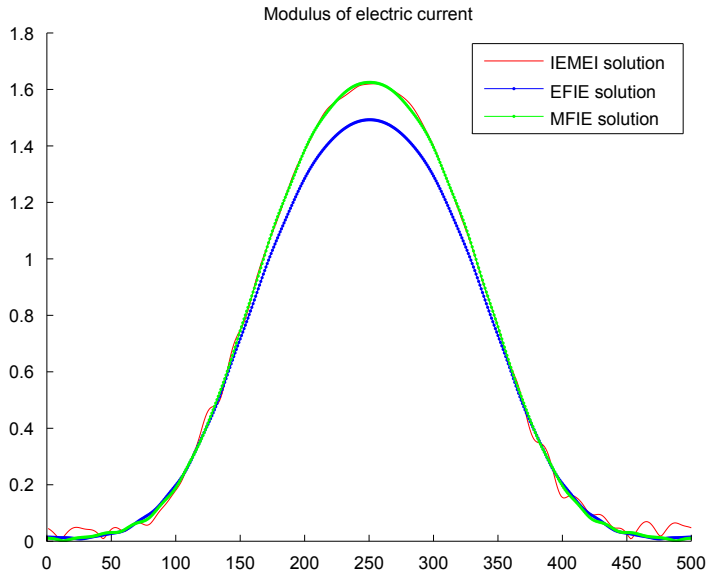

Figure 7.19: Modulus of $\vec{J}$ in circular scatterer, $N = 500$, $kR = 12$, sparsity 3%, $\epsilon_r = -3 - 5j$. Some slight oscillations appear at the sides.

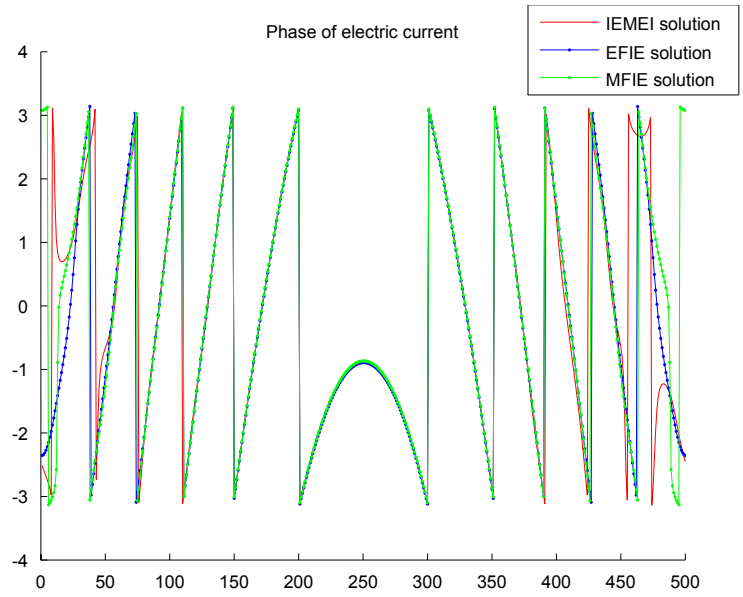

Figure 7.20: Phase of $\vec{J}$ in circular scatterer, $N = 500$, $kR = 12$, sparsity 3%, $\epsilon_r = -3-5j$

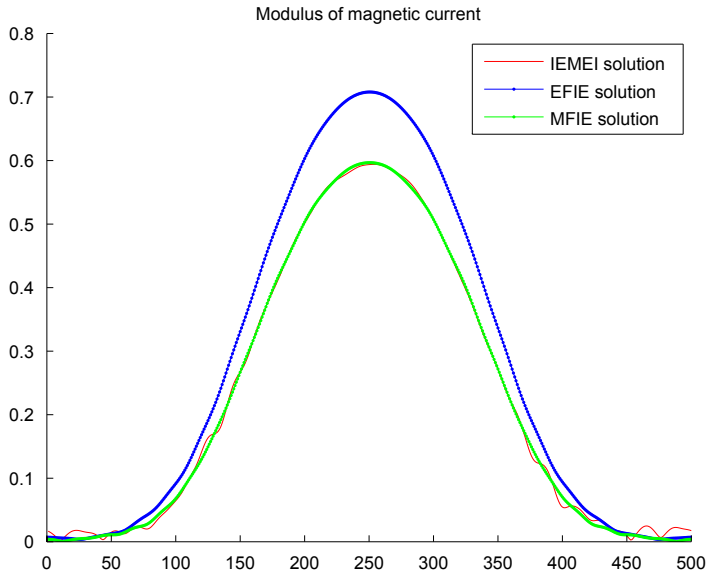

Figure 7.21: Modulus of $\vec{M}$ in circular scatterer, $N = 500$, $kR = 12$, sparsity 3%, $\epsilon_r = -3 - 5j$. Some slight oscillations appear at the sides, product of the increase in the condition number.

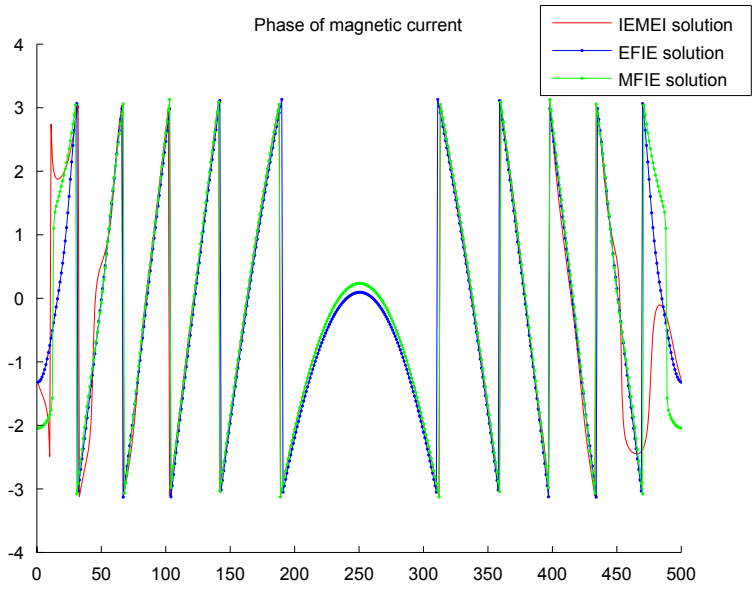

Figure 7.22: Phase of $\vec{M}$ in circular scatterer, $N = 500$, $kR = 12$, sparsity 3%, $\epsilon_r = -3-5j$

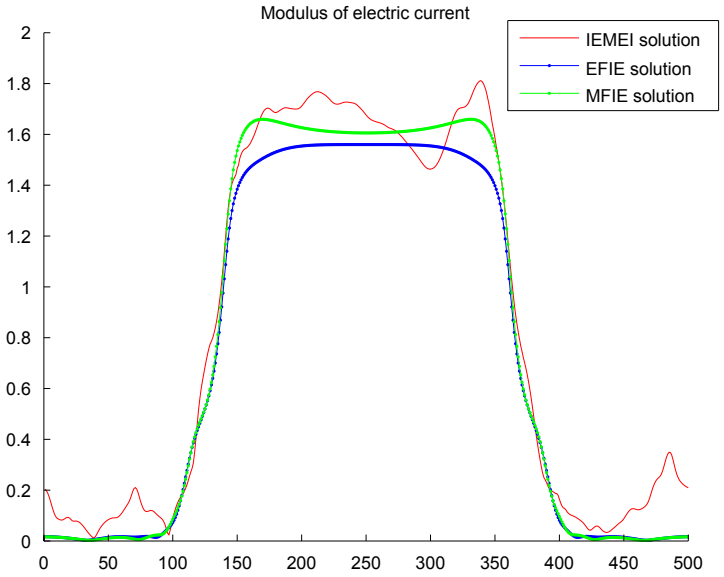

Figure 7.23: Modulus of $\vec{J}$ in superelliptical scatterer, $N = 500$, $kR = (4, 12)$, sparsity 5%, $\epsilon_r = -3-5j$. The results appear contaminated, product of too many variables in the least squares matrices and bad condition number. For smaller but big enough sparsity numbers, the results are good

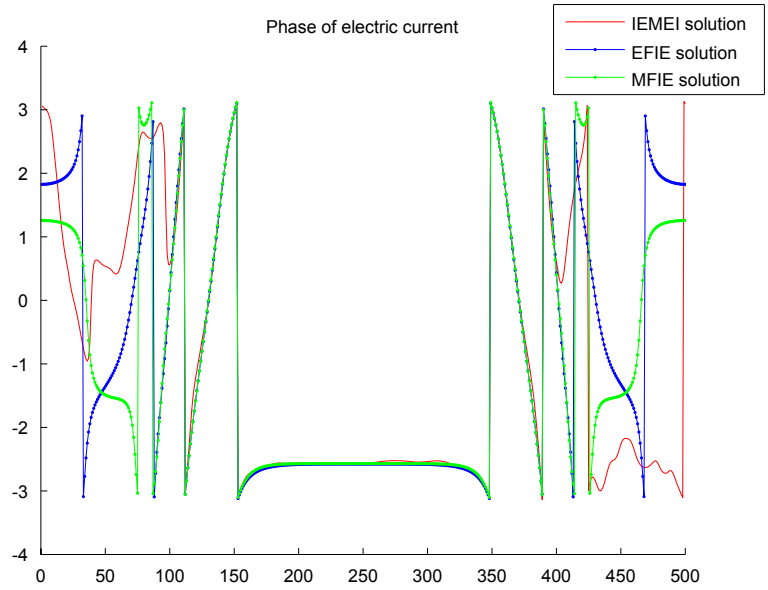

Figure 7.24: Phase of $\vec{J}$ in superelliptical scatterer, $N = 500$, $kR = (4, 12)$, sparsity 5%, $\epsilon_r = -3 - 5j$

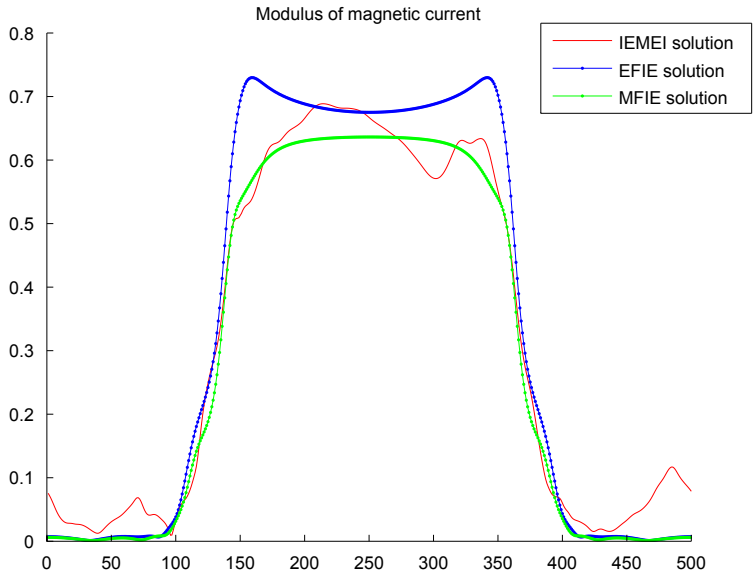

Figure 7.25: Modulus of $\vec{M}$ in superelliptical scatterer, $N = 500$, $kR = (4, 12)$, sparsity 5%, $\epsilon_r = -3 - 5j$.

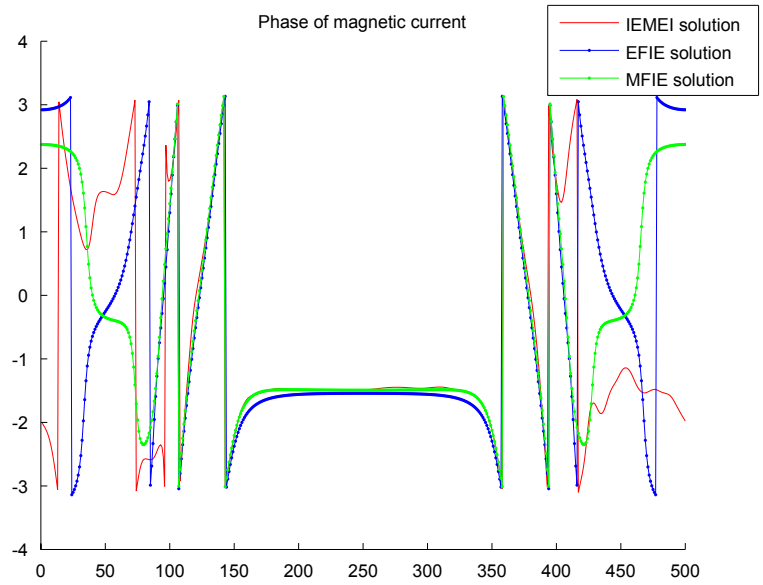

Figure 7.26: Phase of $\vec{M}$ in superelliptical scatterer, $N = 500$, $kR = (4, 12)$, sparsity 5%, $\epsilon_r = -3 - 5j$

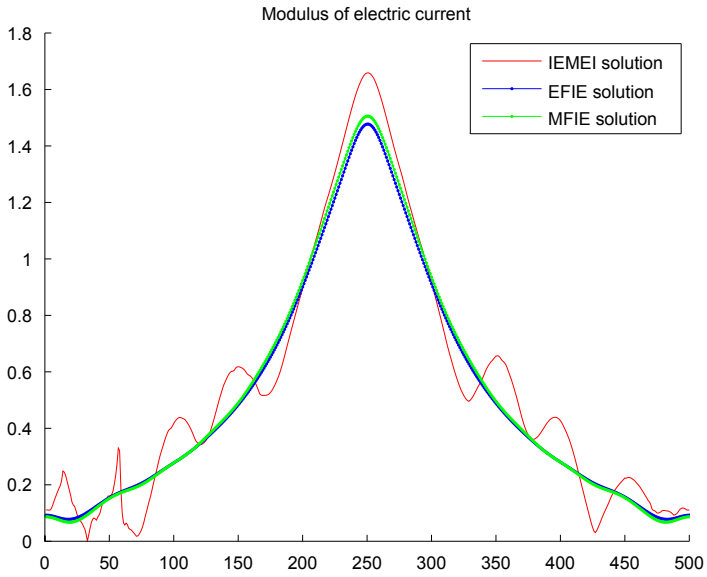

Figure 7.27: Modulus of $\vec{J}$ in elliptical scatterer, $N = 500$, $kR = (7, 2)$, sparsity 1.5%, $\epsilon_r = -3 - 5j$

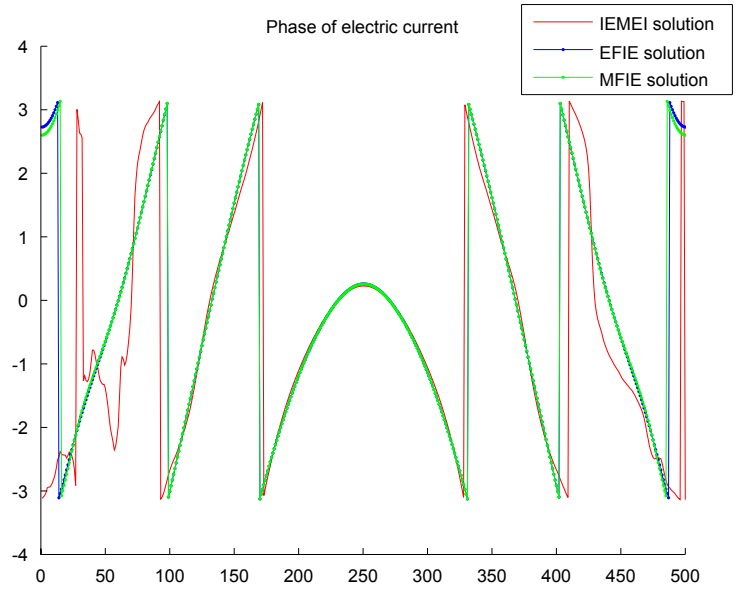

Figure 7.28: Phase of $\vec{J}$ in elliptical scatterer, $N = 500$, $kR = (7, 2)$, sparsity 1.5%, $\epsilon_r = -3 - 5j$

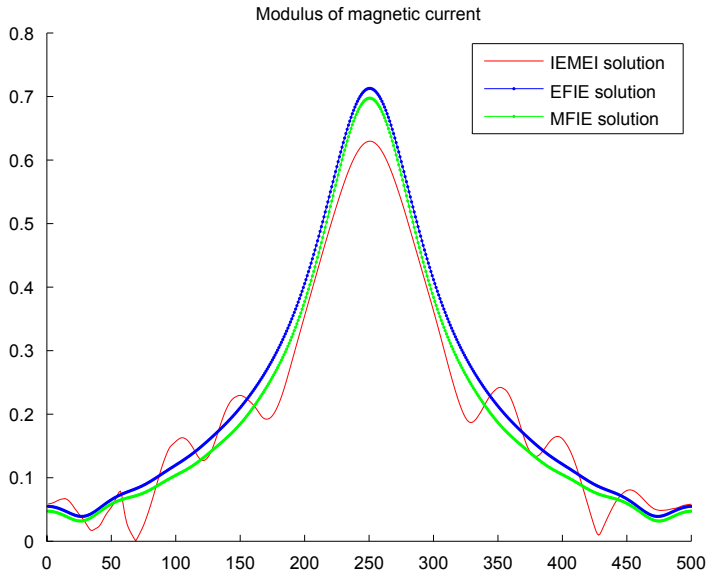

Figure 7.29: Modulus of $\vec{M}$ in elliptical scatterer, $N = 500$, $kR = (7, 2)$, sparsity 1.5%, $\epsilon_r = -3 - 5j$

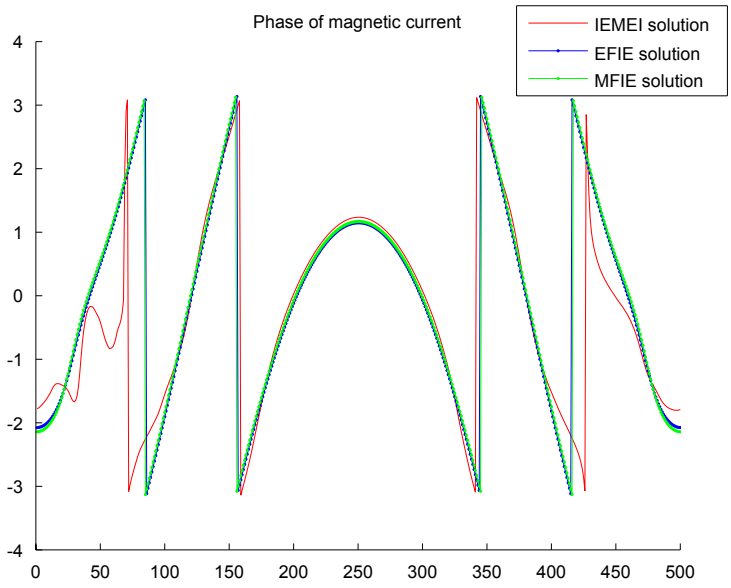

Figure 7.30: Phase of $\vec{M}$ in elliptical scatterer, $N = 500$, $kR = (7, 2)$, sparsity 1.5%, $\epsilon_r = -3 - 5j$

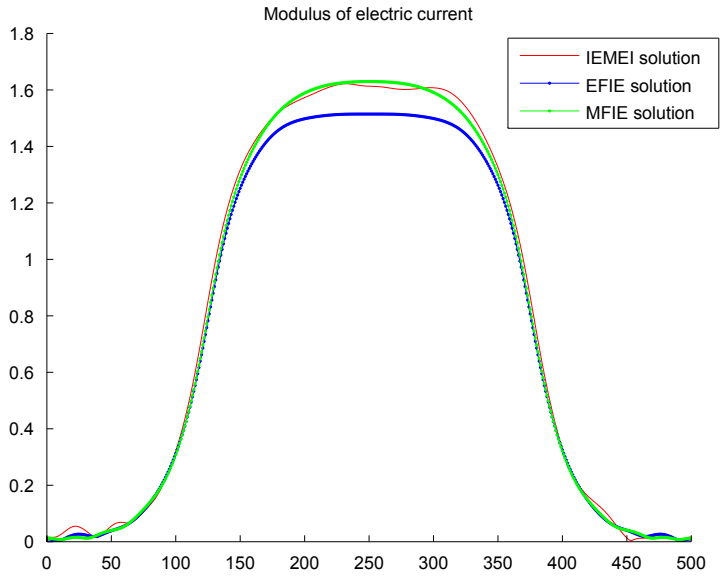

Figure 7.31: Modulus of $\vec{J}$ in elliptical scatterer, $N = 500$, $kR = (3, 10)$, sparsity 3%, $\epsilon_r = -3 - 5j$

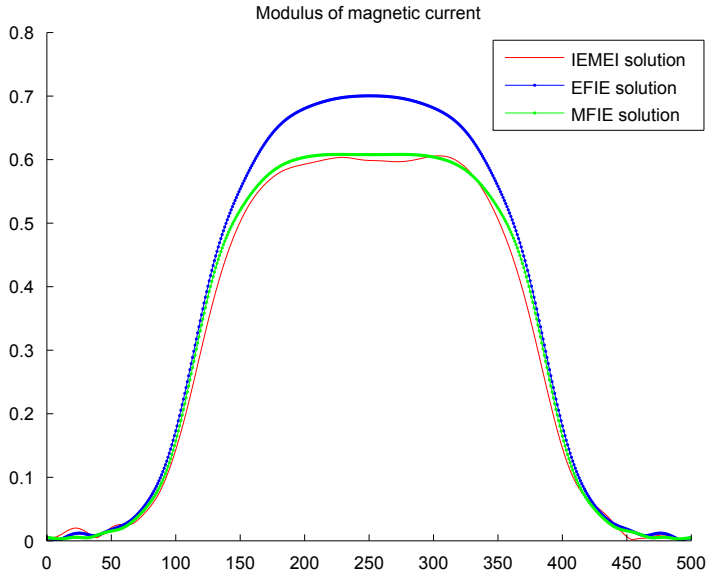

Figure 7.32: Modulus of $\vec{M}$ in elliptical scatterer, $N = 500$, $kR = (3, 10)$, sparsity 3%, $\epsilon_r = -3 - 5j$

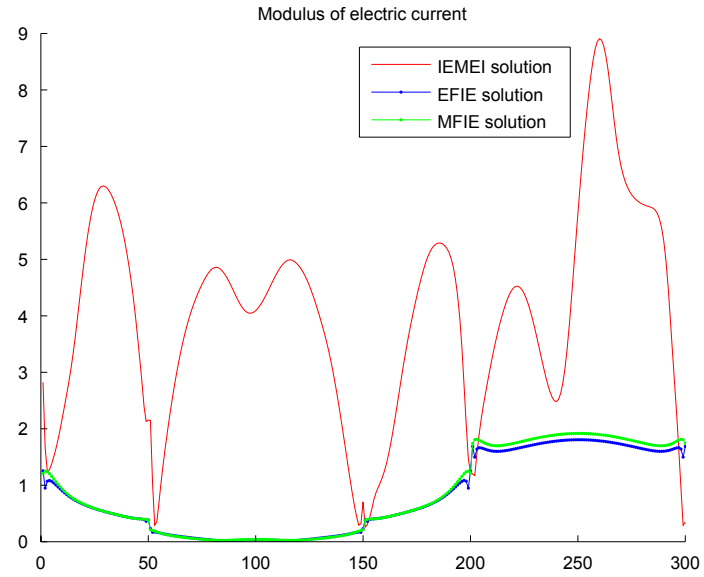

Figure 7.33: Modulus of $\vec{J}$ in rectangular scatterer, $N = 300$, $kR = (2, 4)$, sparsity 5%, $\epsilon_r = -12 - 9j$. The results are very bad, as in general with all the simulations of the rectangle.

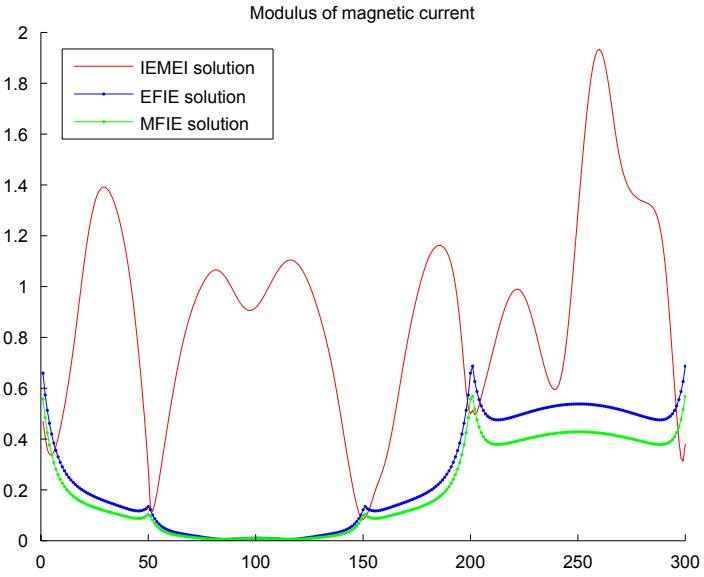

Figure 7.34: Modulus of $\vec{M}$ in rectangular scatterer, $N = 300$, $kR = (2, 4)$, sparsity 5%, $\epsilon_r = -12 - 9j$

Chapter 8

Conclusions

The IE-MEI method has been formulated and implemented for 2D dielectric objects. Preliminar results for high-loss complex dielectric constants having negative real part, like those of plasmonic materials, show good linear system matrix sparsification and excellent agreement with the Method of Moments reference results, and have been presented at the *IX Iberian Meeting on Computational Electromagnetics(EIEC 2015), Baeza*, and the *International Conference on Electromagnetics in Advanced Applications (ICEAA 2015), Torino* (the latter to take place at the beginning of September). In the $N \times N$ formulation used, the matrices relating $\vec{J}$ and $\vec{M}$ are almost diagonal, or at least extremely sparse, and this is independent of the IE-MEI method and can be used as a first small approximation of the MoM. The most important findings of this work are summarized below:

- In the calculation of the IE-MEI matrices $[A]$ and $[B]$, the increase of the number of unknowns, which is itself of interest in the simulation of plasmonic resonances (which require a very thin and smooth discretization), results in a steep growth of the condition number of the system and, particularly, of the least squares matrices, which is in itself considerably big. Specially prone to ill conditioned matrices are the very elongated objects, where elements in the center of the bar are highly dependent on all the rest of elements (in particular to the ones at the other side of the bar). This elements contributed the most part of the error in the solution.

- Although having big condition numbers, the problems in the $N \times N$ formulation provide better results than in the $2N \times 2N$ formulation, where the condition numbers were much worse. The latter was abandoned in favour of the former.

- It has been shown, too, that the distribution of imposed zeros has a considerable

effect on the accuracy, although no theoretical work has been done to support the claims. This is of particular interest in non canonical geometries, although discontinuities in the derivative of the boundary, such as in the rectangle case, present problems. Using a reduced numbers of zeros is also of tremendous help in reducing the computation time.

If well distributed, only a few percent (2-5%) of the zeros had to be forced to obtain matrices with a big band diagonal and the rest of the elements (forced zeros and elements without constraints but close to the zeros) several orders of magnitude below. As stated, though, when increasing the size of the scatterer and, therefore, using a higher number of elements, ill conditioned matrices appeared in the least squares method.

- Harmonic metrons (which had been used successfully before in 2D scatterers) were also tested here with dielectrics. In contrast with delta metrons, they didn't produce sparse enough matrices. The "limited" zero forcing strategy gave very poor results, to the point where truncation was not applicable, and the use of delta metrons was resumed.

Some suggestions are also given, if this or a similar line of research is to be followed in the future:

- Combination of the IE-MEI with multilevel decomposition interative fast solvers like MLFMM (in PEC, lossless or lossy dielectrics) to sparsify near field matrices.

- Work in the reduction of the condition number, by using preconditioners or even by using other different error minimisation techniques. Least squares was the option used in the work, but some shallow attempts (without much success) were made at using L^1 minimisation (of which there is plenty of material and bibliography to be found, in the domain of linear programming and simplex-like algorithms). Some form of Compressed Sensing or Orthogonal Matching Pursuit could be of use.

- Implementation of a variable number of zeros forced per row (instead of the constant number used in this work in all the rows of the matrix); in this case, some elements would "receive" more zeros than others.

- Exportation of the IE-MEI to 3D plasmonic objects, with the zero forcing scheme described. A possible way to do it (in geometries homeomorphic to a 2-sphere)

would be to order the elements as in the peeling of an apple. Another way would be to set a probability of an element (column index) to be a forced zero depending on the geometrical distance to the source (row index).

In any case, sparsification of electrically small geometries is working successfully in the current implementation, and is useful to reduce storage requirements of the near field matrix in multilevel decomposition fast solvers. In bigger geometries, further guarantees on the quality of the solution are required.

Chapter 9

Annexes

9.1 List of publications

The results of this work have been presented at the following conferences:

1. "Integral Equation MEI (IE-MEI) analysis of homogeneus dielectric and plasmonic media in 2D", Juan M. Rius, G. Planes, E. Ubeda and A. Heldring, IX Iberian Meeting on Computational Electromagnetics(EIEC 2015), Baeza, Spain, 6-8 May 2015.

2. "Efficient analysis of homogeneous dielectric and plasmonic media with integral equation MEI (IE-MEI)", Juan M. Rius, G. Planes, E. Ubeda and A. Heldring, International Conference on Electromagnetics in Advanced Applications (ICEAA 2015), Torino, Italy, September 7-11, 2015.

A paper is also in preparation:

1. "Sparsification of the MLFMA near field matrix with IE-MEI for high-loss dielectric objects", in preparation for IEEE Trans. on Antennas and Propag.

9.2 Review of electromagnetism

9.2.1 Maxwell's equations

Maxwell's equations are a set of partial differential equations that, together with the Lorentz force law, form the foundation of classical electrodynamics, classical optics, and

electric circuits. These fields in turn underlie modern electrical and communications technologies. Maxwell's equations describe how electric and magnetic fields are generated and altered by each other and by charges and currents. They are named after the physicist and mathematician James Clerk Maxwell, who published an early form of those equations between 1861 and 1862.

The powerful and most widely familiar form of Maxwell's equations, whose formulation is due to Oliver Heaviside, in the vector calculus formalism, is used throughout.

The equations introduce the electric field $\mathbf{E}$, a vector field, and the magnetic field $\mathbf{B}$, a pseudovector field, where each generally have time-dependence. The sources of these fields are electric charges and electric currents, which can be expressed as local densities namely charge density ρ and current density $\mathbf{J}$. A separate law of nature, the Lorentz force law, describes how the electric and magnetic field act on charged particles and currents. A version of this law was included in the original equations by Maxwell but, by convention, is no longer.

In differential terms, this equations take the form

$$\nabla \cdot \mathbf{D} = \rho \tag{9.1}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{9.2}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{9.3}$$

$$\nabla \times \mathbf{H} = \mathbf{J} + \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t} \tag{9.4}$$

Here, $\mathbf{H}$ and $\mathbf{D}$ are also used: in order to apply *Maxwell's macroscopic equations*, it is necessary to specify the relations between displacement field $\mathbf{D}$ and the electric field $\mathbf{E}$, as well as the magnetizing field $\mathbf{H}$ and the magnetic field $\mathbf{B}$. The equations specifying this response are called constitutive relations, and are usually taken in the form

$$\mathbf{D} = \varepsilon \mathbf{E}\,, \quad \mathbf{H} = \frac{1}{\mu}\mathbf{B} \tag{9.5}$$

Thus, Maxwell's equations are usually expressed in engineering in terms of E and H.

9.2.2 Wave equation

By solving in terms of $\mathbf{E}$ the Maxwell's equation (there are several ways to do it, see the Annexes of [4]), we obtain the classical wave equation

$$\left(c^2\nabla^2 - \frac{\partial^2}{\partial t^2}\right)\mathbf{E} = \mathbf{0} \tag{9.6}$$

Furthermore, if we can express the field as a product of the space depending field and a time varying harmonic function e^{jwt}, this equation simplifies to the Helmholtz equation

$$(\nabla^2 + k^2)\mathbf{E} = 0 \tag{9.7}$$

where $k = \frac{\omega}{c} = \frac{2\pi}{\lambda}$ is the wavenumber in vacuum. It is worth noting that in both equations 9.6 and 9.7, the differential linear operator is perfectly visible at the left side of the equation, as a "product" with $\mathbf{E}$.

9.2.3 Boundary conditions between media

Maxwell's equations describe the behaviour of electromagnetic fields; electric field, electric displacement field, magnetic field and magnetic field strength. The differential forms of these equations require that there is always an open neighbourhood around the point to which they are applied, otherwise the vector fields $\mathbf{E}$, $\mathbf{D}$, $\mathbf{B}$ and $\mathbf{H}$ are not differentiable. In other words, the medium must be continuous. On the interface of two different medium with different values for electrical permittivity and magnetic permeability that does not apply.

However the interface conditions for the electromagnetic field vectors can be derived from the integral forms of Maxwell's equations. These are, for the electric field

$$\mathbf{n}_{12} \times (\mathbf{E}_2 - \mathbf{E}_1) = \mathbf{M} \tag{9.8}$$

where $\mathbf{n}_{12}$ is normal vector from medium 1 to medium 2, and for the magnetic field

$$-\mathbf{n}_{12} \times (\mathbf{H}_2 - \mathbf{H}_1) = \mathbf{J} \tag{9.9}$$

Here, $\mathbf{J}$ and $\mathbf{M}$ are the *electric* and *magnetic* currents. Normally, $\mathbf{M}$ is zero because actual magnetic currents and charges never exist in a physical situation. However, for calculation purposes $\mathbf{M}$ can appear (and in this work, in fact, does) several times if the scattering medium is a dielectric. In PEC cases, $\mathbf{M} = 0$.

9.2.4 Green's functions

Definition and uses

A *Green's function*, $G(x, s)$, of a linear differential operator $L = L(f)$ acting on distributions over a subset of the Euclidean space $\mathbb{R}^n$ at a point s, is any solution of

$$LG(x, s) = \delta(x - s) \tag{9.10}$$

where δ is the Dirac delta function. This property of a Green's function can be exploited to solve differential equations of the form

$$Lu(x) = f(x) \tag{9.11}$$

If the kernel L is non-trivial, then the Green's function is not unique. However, in practice, some combination of symmetry, boundary conditions and/or other externally imposed criteria will give a unique Green's function. Also, Green's functions in general are distributions, not necessarily proper functions (as in the Dirac delta function case).

Green's functions are useful tools in solving wave equations and diffusion equations, among others. If the operator is translation invariant, that is, when L has constant coefficients with respect to x, then the Green's function can be taken to be a convolution operator, that is

$$G(x, s) = G(x - s) \tag{9.12}$$

In this case, the Green's function is the same as the impulse response of linear time-invariant system theory, and this is exactly the case in the wave equation in electromagnetism.

Motivation

Loosely speaking, if such a function G can be found for the operator L, then if we multiply the equation 9.10 for the Green's function by $f(s)$, and then perform an integration in the s variable, we obtain

$$\int LG(x, s)f(s)\,ds = \int \delta(x - s)f(s)\,\mathrm{d}s = f(x) \tag{9.13}$$

The right-hand side is now given by the equation 9.11 to be equal to $Lu(x)$, thus:

$$Lu(x) = \int LG(x, s)f(s)\,\mathrm{d}s \tag{9.14}$$

Because the operator $L = L(x)$ is linear and acts on the variable x alone (not on the variable of integration s), we can take the operator L outside of the integration on the right-hand side, obtaining

$$Lu(x) = L\left(\int G(x, s)f(s)\,\mathrm{d}s\right) \tag{9.15}$$

which suggests

$$u(x) = \int G(x,s)f(s)\,\mathrm{d}s = \int G(x-s)f(s)\,\mathrm{d}s \tag{9.16}$$

Thus, we can obtain the function $u(x)$ through knowledge of the Green's function in equation 9.10 and the source term on the right-hand side in equation 9.11. This process relies upon the linearity of the operator L.

In other words, the solution of equation 9.11, $u(x)$, can be determined by the integration given in equation 9.16. Although $f(x)$ is known, this integration cannot be performed unless G is also known. The problem now lies in finding the Green's function G that satisfies equation 9.10. For this reason, the Green's function is also sometimes called the fundamental solution associated to the operator L.

Not every operator L admits a Green's function. A Green's function can also be thought of as a right inverse of L. Aside from the difficulties of finding a Green's function for a particular operator, the integral in equation 9.16 may be quite difficult to evaluate. However the method gives a theoretically exact result.

9.3 Code

In this annex the realized programs for this project are displayed. Most of them are `MATLAB` routines, with the exception of a few `C` and `C++` routines in which a more low-level language was preferred: the routines `oper_die_2d` and `oper_die_2d_complex`, which have been developed jointly by J.M. Rius and the author. Complex Bessel functions in `C++` have also been used from a mathematical library developed by CR Bond. The rest of the code has been written by the author.

9.3.1 Main MATLAB routines

die_scat_nn.m

Main function. Performs all the operations described in 5.2 for a general 2D dielectric scatterer.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MoM for a dielectric scatterer, surrounded by dielectric medium
% Includes EFIE, MFIE equations and IEMEI method, with N x N matrices
%
```

```
% equations: 'efie', 'mfie' or 'iemei'
% N: number of elements in scatterer surface
% Z_tot_spars: wanted sparsity in final impedance matrix (the one reduced
%             by least squares)
% Z_hm_spars: wanted sparsity in inverted matrix
% BW: number of non zero elements in every of the bands of matrices A, B
% pc1: preconditioner. Best value around 200-300 usually
% e_r2, mu_r2: relative permittivity and permeability in dielectric scatterer
% kr: value of k*r (size of object to lambda)
%
% NOTES
% - The permittivity and permeability can be complex
% - There is one version of get_iemei_matrices.
%   get_iemei_matrices: A of the form: single circular band so that
%                       AZe is also single circular band
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% G. Planes Conangla, 3/2015
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function[str] = die_scat_nn(equations, N, Z_tot_spars, force_zero_frac,
    JM_threshold, trunc_bound, e_r2, mu_r2, k_geo_parameters, geometry, green,
    varargin)
%% Start timer
tic;
%% constants/static variables

% dielectric material constants
% outside dielectric
e_r1 = 1;
mu_r1 = 1;
% inside dielectric
% e_r2 and mu_r2 are called as input to the function
% impedance
eta_0 = 119.9169*pi; % in the void
eta_r1 = sqrt(mu_r1/e_r1);
eta_r2 = sqrt(mu_r2/e_r2);
```

```
eta_1 = eta_0*eta_r1;
eta_2 = eta_0*eta_r2;

% incident field;
lambda = 1;
k_0 = 2*pi/lambda;
k_1 = k_0*sqrt(mu_r1*e_r1);
k_2 = k_0*sqrt(mu_r2*e_r2);
ang_i = 0; % angle in degrees
k_i = k_1*exp(1i*ang_i*pi/180);

% SCATTERER
kr = k_geo_parameters(1);
geo_parameters = k_geo_parameters/k_1;

%% gui or command line
if ~isempty(varargin) % gui
    fig_1 = varargin{1};
    fig_2 = varargin{2};
    fig_3 = varargin{3};
    fig_4 = varargin{4};
    fig_5 = varargin{5};
    fig_6 = varargin{6};
    fig_7 = varargin{7};
else
    fig_1 = 1;
    fig_2 = 2;
    fig_3 = 3;
    fig_4 = 4;
    fig_5 = 5;
    fig_6 = 6;
    fig_7 = 7;
end

%% IEMEI: only for IEMEI method.
% BW: number of non zero elements in every of the bands of matrices A, B
%     Called as input of the function.
```

```
% BW_ext: forced BW in A*Z_e + B*Z_h (not necessarily the same as BW)
%        defined through factor k_bw
% En_bw: final BW used in A*Z_e + B*Z_h (slightly bigger or equal than the
%       forced BW_ext)
% Note: max(BW, BW_ext) <= En_bw
% The bandwidth is defined as: BW - diag - BW in matrices A & B
% so we have 2*BW + 1 nonzero elements for every band (A and B can be
% single or triband).
% B_force_zero: variable to force zeros only in the neighborhood of the
%              band (previous data showed that this could be enough to
%              force zeros everywhere). However with dielectrics this is
%              not actually the case (at least not with this
%              implementation), so B_force_zero is not used
BW = ceil((Z_tot_spars*N - 1)/2);
k_bw = 0.8;
BW_ext = floor(BW*k_bw);
En_bw = max(BW_ext, BW);
%force_zero_frac = 0.18; % from 1 to 0: 1 is force all numbers, 0.5 would force
                   % only 50% of matrix elements in the neighborhood of
                   % the band.
B_force_zero = floor((N - (2*BW+1))*(force_zero_frac/2));

% matrix conditioning
pc1 = 300; % factor for the magnetic current part of matrix

%% Field and geometry of the scatterer
% scatterer and boundary
deg = 0;
switch geometry
    case 'elliptical'
        zl = elliptical(N, geo_parameters);
        [un, dl, z] = normals(zl);
        rad = max(geo_parameters);
    case 'pill'
        [zl, deg] = pill(N, geo_parameters);
        [un, dl, z] = normals(zl);
        rad = max(geo_parameters);
```

```
    case 'rectangular'
        zl = rectang_geometry(N, geo_parameters);
        [un, dl, z] = normals(zl);
        rad = max(geo_parameters);
end
h = dl(1);

% distance matrix
dist1 = repmat(zl(1:end-1), 1, N);
dist_dual = dist1.';
dist = abs(dist1 - dist_dual); % distance matrix between discr. elements
ef_dist = exp(-1j*k_2*dist);
% draw_matrix(ef_dist,'ef_dist',2);

% incident field
Ei = exp(-1i*dot(k_i,z));
E_to_H = -dot(un,k_i/k_1)/eta_1;
Hi = E_to_H.*Ei;

%% MoM
% time before matrix calculation
t1 = toc;

% Calculate matrices (EFIE, MFIE or IEMEI case)
switch equations
    case 'efie'
        % EFIE field for dielectrics
        Inc_e = Ei;
        % impedance matrices
        [Z_ej1, Z_ej2, Z_em1, Z_em2] =
            efie_matrix(z,un,dl,k_0,e_r1,mu_r1,e_r2,mu_r2,N,pc1);

    case 'mfie'
        % MFIE field for dielectrics
        Inc_h = Hi;
        % impedance matrix
        [Z_hj1, Z_hj2, Z_hm1, Z_hm2] =
```

```
            mfie_matrix(z,un,dl,k_0,e_r1,mu_r1,e_r2,mu_r2,N,pc1);

    case 'iemei_efie' % A*EFIE, for the moment
        % EFIE field for dielectrics
        Inc_e = Ei;
        % MFIE field for dielectrics
        Inc_h = Hi;
        % impedance matrices EFIE
        [Z_ej1, Z_ej2, Z_em1, Z_em2] =
            efie_matrix(z,un,dl,k_0,e_r1,mu_r1,e_r2,mu_r2,N,pc1);
        % impedance matrices MFIE (only for solution comparison, not
        % needed)
        [Z_hj1, Z_hj2, Z_hm1, Z_hm2] =
            mfie_matrix(z,un,dl,k_0,e_r1,mu_r1,e_r2,mu_r2,N,pc1);
        % IEMEI matrix
        % truncate Z_em2
%         [Z_trunc, ~] = make_sparse(Z_em2, JM_threshold);
        Z_trunc = trunc_matrix(Z_em2, JM_threshold);
        sp_Z_trunc = sparsity(Z_trunc);
        % calc. aprox Z_efie with truncated Z_em2
        %[L, U] = ilu(sparse(-Z_trunc));
        Z_efie_aprox = (Z_ej1 + (Z_em1 + eye(N))*(-Z_trunc\Z_ej2));
        t_iemei = toc;
        [A, band_M, cond_vec, er_vec] = get_iemei_matrices(Z_efie_aprox, N,
            BW, BW_ext, B_force_zero);
    case 'iemei_mfie' % A*EFIE, for the moment
        % EFIE field for dielectrics
        Inc_e = Ei;
        % MFIE field for dielectrics
        Inc_h = Hi;
        % impedance matrices EFIE (only for solution comparison, not
        % needed)
        [Z_ej1, Z_ej2, Z_em1, Z_em2] =
            efie_matrix(z,un,dl,k_0,e_r1,mu_r1,e_r2,mu_r2,N,pc1);
        % impedance matrices MFIE
        [Z_hj1, Z_hj2, Z_hm1, Z_hm2] =
            mfie_matrix(z,un,dl,k_0,e_r1,mu_r1,e_r2,mu_r2,N,pc1);
```

```
        % IEMEI matrix
        % truncate Z_hj1
%         [Z_trunc, ~] = make_sparse(Z_hj1, JM_threshold);
        Z_trunc = trunc_matrix(Z_hj1, JM_threshold);
        sp_Z_trunc = sparsity(Z_trunc);
        % make sparse if sparsity low enough
%         if(sp_Z_trunc < 0.3)
%             Z_trunc = sparse(Z_trunc);
%         end
        % calc. aprox Z_efie with truncated Z_em2
        %[L, U] = ilu(sparse(-Z_trunc));
        Z_mfie_aprox = (Z_hm2 + (Z_hj2 + eye(N))*(-Z_trunc\Z_hm1));
        t_iemei = toc;
        [A, band_M, cond_vec, er_vec] = get_iemei_matrices(Z_mfie_aprox, N,
            BW, BW_ext, B_force_zero);
    case 'iemei' % A*EFIE, for the moment
        % EFIE field for dielectrics
        Inc_e = Ei;
        % MFIE field for dielectrics
        Inc_h = Hi;
        % impedance matrices EFIE
        [Z_ej1, Z_ej2, Z_em1, Z_em2] =
            efie_matrix(z,un,dl,k_0,e_r1,mu_r1,e_r2,mu_r2,N,pc1);
        % impedance matrices MFIE
        [Z_hj1, Z_hj2, Z_hm1, Z_hm2] =
            mfie_matrix(z,un,dl,k_0,e_r1,mu_r1,e_r2,mu_r2,N,pc1);
        % IEMEI matrix
        % truncate Z_em2 (EFIE)
%         [Z_trunc1, ~] = make_sparse(Z_em2, JM_threshold);
        Z_trunc1 = trunc_matrix(Z_em2, JM_threshold);
%         draw_matrix(Z_trunc1,'truncJM',1);
        sp_Z1_trunc = sparsity(Z_trunc1);
        % truncate Z_hj1 (MFIE)
%         [Z_trunc2, ~] = make_sparse(Z_hj1, JM_threshold);
        Z_trunc2 = trunc_matrix(Z_hj1, JM_threshold);
        sp_Z_trunc = sparsity(Z_trunc2);
        % calc. aprox Z_efie with truncated Z_em2 // amb M (millor)
```

```
%       Z_efie_aprox = (Z_ej1*(-Z_trunc2\Z_hm1) + (Z_em1 + eye(N)));
        Z_efie_aprox = (Z_ej1*(-Z_trunc2\Z_hm1) + (Z_em1)); % + eye(N) <-
            prova FFT
        Z_mfie_aprox = (Z_hm2 + (Z_hj2 + eye(N))*(-Z_trunc2\Z_hm1));
%       % calc. aprox Z_efie with truncated Z_em2 // amb J (pitjor)
%       Z_efie_aprox = (Z_ej1 + (Z_em1 + eye(N))*(-Z_trunc1\Z_ej2));
%       Z_mfie_aprox = (Z_hm2*(-Z_trunc1\Z_ej2) + (Z_hj2 + eye(N)));
        t_iemei = toc;
%       [A, ~] = get_iemei_matrices(Z_efie_aprox, N, BW, BW_ext,
    B_force_zero);
%       [B, band_M] = get_iemei_matrices(Z_mfie_aprox, N, BW, BW_ext,
    B_force_zero);
        [A, B, band_M, cond_vec, er_vec] = get_iemei_matrices2(Z_efie_aprox,
            Z_mfie_aprox, ef_dist, N, BW, BW_ext, B_force_zero);
    otherwise
        display('The equations should be "efie", "mfie", "iemei_efie",
            "iemei_mfie" or "iemei"');
        return;
end

%% Solution
% cpu time before matrix inversion
t2 = toc;
% solve equations for electric & magnetic current
switch equations
    case 'efie'
        X = (-Z_em2\Z_ej2);
        J = (Z_ej1 + (Z_em1 + eye(N))*X)\Inc_e;
        % get current vectors
        M = X*J; %magnetic current
        % time after matrix inversion
        t3 = toc;
    case 'mfie'
        X = (-Z_hj1\Z_hm1);
        M = (Z_hm2 + (Z_hj2 + eye(N))*X)\Inc_h;
        % get current vectors
        J = X*M; %electric current
```

```
        % time after matrix inversion
        t3 = toc;
    case 'iemei_efie'
        Z_iemei = (A*Z_efie_aprox);
%         draw_matrix(Z_iemei,'iemei mfie',fig_1);
        cond_prob = condest(Z_iemei);
        Z_iemei = trunc_matrix(Z_iemei,trunc_bound);
        % get current vectors
        J = Z_iemei\(A*Inc_e);
        M = (-Z_trunc\Z_ej2)*J;
        % time after matrix inversion
        t3 = toc;
        %% compare with exact system
        Z_efie = (Z_ej1 + (Z_em1 + eye(N))*(-Z_em2\Z_ej2));
        Z_mfie = (Z_hm2 + (Z_hj2 + eye(N))*(-Z_hj1\Z_hm1));

        J_e_efie = Z_efie\(Inc_e);
        M_e_efie = (-Z_em2\Z_ej2)*J_e_efie;
        M_e_mfie = Z_mfie\(Inc_h);
        J_e_mfie = (-Z_hj1\Z_hm1)*M_e_mfie;

        cond_current = condest(Z_em2);
        % L2 error
        % error between MFIE & EFIE
        er_J_fie = norm(J_e_efie - J_e_mfie)/norm(J_e_mfie);
        er_M_fie = norm(M_e_efie - M_e_mfie)/norm(M_e_mfie);
        % error between MFIE & IEMEI
        er_J = norm(J_e_mfie - J)/norm(J_e_mfie);
        er_M = norm(M_e_mfie - M)/norm(M_e_mfie);
        % calc. sparsity
        sp_iemei = sparsity(Z_iemei);
    case 'iemei_mfie'
        Z_iemei = A*Z_mfie_aprox;
        draw_matrix(Z_iemei,'iemei AZ_e + BZ_h',fig_1);
        cond_prob = condest(Z_iemei);
%         draw_matrix(Z_iemei,'iemei AZ_e + BZ_h',1);
        Z_iemei = trunc_matrix(Z_iemei,trunc_bound);
```

```
        % get current vectors
        M = Z_iemei\(A*Inc_h);
        J = (-Z_trunc\Z_hm1)*M;
        % time after matrix inversion
        t3 = toc;
        %% compare with exact system
        Z_efie = (Z_ej1 + (Z_em1 + eye(N))*(-Z_em2\Z_ej2));
        Z_mfie = (Z_hm2 + (Z_hj2 + eye(N))*(-Z_hj1\Z_hm1));

        J_e_efie = Z_efie\(Inc_e);
        M_e_efie = (-Z_em2\Z_ej2)*J_e_efie;
        M_e_mfie = Z_mfie\(Inc_h);
        J_e_mfie = (-Z_hj1\Z_hm1)*M_e_mfie;

        cond_current = condest(Z_hj1);
        % L2 error
        % error between MFIE & EFIE
        er_J_fie = norm(J_e_efie - J_e_mfie)/norm(J_e_mfie);
        er_M_fie = norm(M_e_efie - M_e_mfie)/norm(M_e_mfie);
        % error between MFIE & IEMEI
        er_J = norm(J_e_mfie - J)/norm(J_e_mfie);
        er_M = norm(M_e_mfie - M)/norm(M_e_mfie);
        % calc. sparsity
        sp_iemei = sparsity(Z_iemei);
    case 'iemei'
        Z_iemei = A*Z_efie_aprox + B*Z_mfie_aprox;
        draw_matrix(Z_iemei,'iemei AZ_e + BZ_h',fig_1);
        cond_prob = condest(Z_iemei + A);
%         draw_matrix(Z_iemei,'iemei AZ_e + BZ_h',1);
        Z_iemei = trunc_matrix(Z_iemei,trunc_bound);
        % get current vectors // amb M (millor)
        M = (Z_iemei + A)\(A*Inc_e + B*Inc_h);
        J = (-Z_trunc2\Z_hm1)*M;
        % time after matrix inversion
        t3 = toc;
        %% compare with exact system
        Z_efie = (Z_ej1 + (Z_em1 + eye(N))*(-Z_em2\Z_ej2));
```

```
        Z_mfie = (Z_hm2 + (Z_hj2 + eye(N))*(-Z_hj1\Z_hm1));

        J_e_efie = Z_efie\(Inc_e);
        M_e_efie = (-Z_em2\Z_ej2)*J_e_efie;
        M_e_mfie = Z_mfie\(Inc_h);
        J_e_mfie = (-Z_hj1\Z_hm1)*M_e_mfie;

        cond_current = condest(Z_em2);
        % L2 error
        % error between MFIE & EFIE
        er_J_fie = norm(J_e_efie - J_e_mfie)/norm(J_e_mfie);
        er_M_fie = norm(M_e_efie - M_e_mfie)/norm(M_e_mfie);
        % error between MFIE & IEMEI
        er_J = norm(J_e_mfie - J)/norm(J_e_mfie);
        er_M = norm(M_e_mfie - M)/norm(M_e_mfie);
        % calc. sparsity
        sp_iemei = sparsity(Z_iemei);
end

% Display calculation times/values used and plot results
if green
    [E, x, y] = get_electric_field(J, M, N, dl, geometry, geo_parameters, deg,
        zl, un, rad*k_1, k_1, k_2, eta_0, eta_2);
    plot_field(E, x, y, zl, fig_2);
else
    plot_error_geo(zl, er_vec, fig_2);
end
str =
    print_results(N,kr,e_r2,mu_r2,BW,BW_ext,En_bw,cond_vec,cond_prob,cond_current,er_vec,t1,t2,
plot_results_modphase(equations, eta_0, J, M, fig_4, fig_5, fig_6, fig_7,
    J_e_efie, M_e_efie, J_e_mfie, M_e_mfie);


end
```

pec.m

Method of moments for a PEC scatterer.

```
% MoM for PEC scatterer, surrounded by dielectric
% with variable e_r, mu_r.
%
% M_i: impedance matrix
%
% GP Conangla 2/2015

function[] = pec

%% constants
% incident field;
lambda = 100;
TM = 1;
% dielectric material constants
e_r = 1;
mu_r = 1;
eta_0 = 119.9169*pi;
eta = eta_0*sqrt(mu_r/e_r);
% wavevector
k_0 = 2*pi/lambda;
k = k_0*sqrt(mu_r*e_r);
ang_i = 0; %angle in degrees
k_i = k*exp(1i*ang_i*pi/180);
% scatterer
rad = 20/k; %radius
N = 1000; %number of elements in scatterer

%% MoM
% scatterer and boundary
zl = circular(N,rad);
[un,dl,z] = normals(zl);
h = dl(1);

% incident field
```

```
Ei = exp(-1i*dot(k_i,z));
Hi = -dot(un,k_i/k).*exp(1i*dot(-k_i,z))/eta;

% impedance matrix
M1 = oper_die_2d(z,un,z.',un,dl,k_0,TM,1,e_r,mu_r);
M2 = oper_die_2d(z,un,z.',un,dl,k_0,TM,0,e_r,mu_r);

%% Solution
% EFIE
J1 = -inv(M1)*Ei;
% MFIE
J2 = -inv(M2-eye(N,N))*Hi;

%% Field in space
% Number of points in x and y
Npun = 200;
% Small shift to avoid singularity of G(r-rn)
eps = 1e-3;
x = linspace(-4*6/k+eps,4*6/k+eps,Npun); % 6/k can be changed by rad, it's
    just some random number
y = x;
[rx,ry] = meshgrid(x,y);     % make the mesh
r = rx + 1j*ry;

% Scattered field
Es = zeros(Npun,Npun);
for n=1:N,
   R = abs(r-zl(n));       % Distance from points of boundary to r
   G = besselh(0,2,k*R)/(4*1j); % Green's function
   Es = Es - 1j*k*eta*h*J2(n)*G;
end

Ei = exp(-1j*k*real(r));
E = Ei + Es; % total field = scattered + incident

%% plot
% current
```

```
figure(1);
clf;
hold on;
plot(eta_0*abs(J1),'r');
plot(eta_0*abs(J2),'b');
legend('EFIE','MFIE');
title('Absolut value of Electric current');

% field
figure(2);
clf;
pcolor(x,y,abs(E));
colormap(jet);
shading('interp');
colorbar;
title('Total field');
xlabel('x'),
ylabel('y');
axis('equal');
axis('square');

end
```

oper_die_2d_complex.cpp

C++ function that implements the basic linear operators.

```
//
// JM Rius 1996
// GP Conangla 2015 (added e_r and mu_r, complex arguments)

#define _USE_MATH_DEFINES
#include <complex>
#include <cmath>
#include "complex_bessel.h"
#include "mex.h"
```

```
using namespace std;
typedef complex<double> dcomp; //complex double

#define _PI 3.141592653589

/* Arguments */

#define IN_ARG 12
#define p_r      prhs[0] /* r = field points (column vector) */
#define p_un     prhs[1] /* un = normal to boundary at FIELD points */
#define p_ri     prhs[2] /* ri = source points (row vector) */
#define p_uni    prhs[3] /* uni = normal to boundary at SOURCE points */
#define p_dl     prhs[4] /* dl = length of basis functions */
#define p_k      prhs[5] /* k = 2*pi/lam */
#define p_TM     prhs[6] /* Polarization, 1->TM, 0->TE */
#define p_Es     prhs[7] /* Field, 1-> Es, 0-> Hs */
#define p_e_r_re prhs[8] /* e_r = relative permittivity, real part */
#define p_e_r_im prhs[9] /* e_r = relative permittivity, imaginary part */
#define p_mu_r_re prhs[10] /* mu_r = relative permeability, real part */
#define p_mu_r_im prhs[11] /* mu_r = relative permeability, imaginary part */

#define OUT_ARG 1
#define p_field plhs[0] /* Field */

static complex<double> ij(0.0,1.0); // imag unit

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{ double *rx, *ry, *rix, *riy, *dl, k_0, *Er, *Ei, *Hr, *Hi, *unx, *uny,
    *unix, *uniy;
  double R, Rmry, Rmrx, Rmlx, Rmly, Rmr, Rml, lx2, ly2;
  dcomp e_real, e_im, e_r, mu_real, mu_im, mu_r, eta, k, kR, tmp, var, kRmr,
      kRml;
  int  N,M,Ni,Mi,i,j,p,TM,Es;

  if (nrhs != IN_ARG) mexErrMsgTxt("Input argument number");

  Mi = mxGetM(p_ri); Ni = mxGetN(p_ri); if(Mi != 1) mexErrMsgTxt("ri must be
```

```
    row vector");
M = mxGetM(p_r); N = mxGetN(p_r); if(N != 1) mexErrMsgTxt("r must be column
    vector");

p_field = mxCreateDoubleMatrix(M, Ni, mxCOMPLEX);

k_0  = *mxGetPr(p_k); TM = *mxGetPr(p_TM); Es = *mxGetPr(p_Es);

// complex permit., permeab.
e_real = *mxGetPr(p_e_r_re);
e_im   = *mxGetPr(p_e_r_im);
e_r    = e_real + e_im*ij;

mu_real = *mxGetPr(p_mu_r_re);
mu_im   = *mxGetPr(p_mu_r_im);
mu_r    = mu_real + mu_im*ij;

// Change k_0 to k
k = k_0*sqrt(e_r*mu_r);

// define eta
eta = 119.9169*_PI*sqrt(mu_r/e_r);

// distance
rx = mxGetPr(p_r);
if(mxIsComplex(p_r)) ry = mxGetPi(p_r); else ry = (double*)
    mxCalloc(M,sizeof(double));

unx = mxGetPr(p_un);
if(mxIsComplex(p_un)) uny = mxGetPi(p_un); else uny = (double*)
    mxCalloc(M,sizeof(double));

rix = mxGetPr(p_ri);
if(mxIsComplex(p_ri)) riy = mxGetPi(p_ri); else riy = (double*)
    mxCalloc(Ni,sizeof(double));

unix = mxGetPr(p_uni);
```

```
if(mxIsComplex(p_uni)) uniy = mxGetPi(p_uni); else uniy = (double*)
    mxCalloc(Ni,sizeof(double));

dl = mxGetPr(p_dl);
if(Es) { Er = mxGetPr(p_field); Ei = mxGetPi(p_field); }
else   { Hr = mxGetPr(p_field); Hi = mxGetPi(p_field); }

dcomp z, cj0, cj1, cy0, cy1, cj0p, cj1p, cy0p, cy1p, cj0_2, cj1_2, cy0_2,
    cy1_2, cj0p_2, cj1p_2, cy0p_2, cy1p_2;

// operators
if(TM && Es) for(j=0, p=0; j<Ni; j++) for(i=0; i<M; i++, p++)
{ R = hypot(rx[i]-rix[j],ry[i]-riy[j]); //distance
  if(R != 0)
  { tmp = eta/4.0*k*dl[j];
    z = k*R;
    cbessjy01(z, cj0, cj1, cy0, cy1, cj0p, cj1p, cy0p, cy1p);
    var = -tmp*cj0 + tmp*cy0*ij;
    Er[p] = var.real();
    Ei[p] = var.imag();
  }
  else
  { tmp = eta/4.0/_PI*k*dl[j];
    var = -_PI*tmp + 2.0*tmp*log(tmp*5.460166e-3)*ij;
    Er[p] = var.real();
    Ei[p] = var.imag();
  }
}

if(TM && !Es) for(j=0, p=0; j<Ni; j++) for(i=0; i<M; i++, p++)
{ R = hypot(rx[i]-rix[j],ry[i]-riy[j]);
  if(R != 0)
  { tmp = -0.25*k*dl[j]*(unx[i]*(rx[i]-rix[j]) + uny[i]*(ry[i]-riy[j]))/R;
    z = k*R;
    cbessjy01(z, cj0, cj1, cy0, cy1, cj0p, cj1p, cy0p, cy1p);
    var = tmp*cy1 + tmp*cj1*ij;
    Hr[p] = var.real();
```

```
    Hi[p] = var.imag();
  }
  else
  { Hr[p] = 0.5;
    Hi[p] = 0;
  }
}

if(!TM && Es) for(j=0, p=0; j<Ni; j++) for(i=0; i<M; i++, p++)
{ R = hypot(rx[i]-rix[j],ry[i]-riy[j]);
  if(R != 0)
  { tmp = -eta/4.0*k*dl[j]*(unx[i]*unix[j] + uny[i]*uniy[j]);
    z = k*R;
    cbessjy01(z, cj0, cj1, cy0, cy1, cj0p, cj1p, cy0p, cy1p);
    var = tmp*cj0 + -tmp*cy0*ij;
    Er[p] = var.real();
    Ei[p] = var.imag();
  }
  else
  { tmp = eta/4.0/_PI*k*dl[j];
    var = -_PI*tmp + 2.0*tmp*log(tmp*5.460166e-3)*ij;
    Er[p] = var.real();
    Ei[p] = var.imag();
  }

  lx2 = -uniy[j]*dl[j]/2;
  ly2 = unix[j]*dl[j]/2;

  Rmrx = rx[i]-rix[j]-lx2;
  Rmry = ry[i]-riy[j]-ly2;
  Rmlx = rx[i]-rix[j]+lx2;
  Rmly = ry[i]-riy[j]+ly2;

  Rmr = hypot(Rmrx,Rmry);
  Rml = hypot(Rmlx,Rmly);
  kRmr = k*Rmr;
  kRml = k*Rml;
```

```
    cbessjy01(kRmr, cj0, cj1, cy0, cy1, cj0p, cj1p, cy0p, cy1p);
    cbessjy01(kRml, cj0_2, cj1_2, cy0_2, cy1_2, cj0p_2, cj1p_2, cy0p_2,
        cy1p_2);

    var = -(eta/4.0*((-uny[i]*Rmrx + unx[i]*Rmry)*cj1/Rmr + (uny[i]*Rmlx -
        unx[i]*Rmly)*cj1_2/Rml)) + (eta/4.0*((-uny[i]*Rmrx +
        unx[i]*Rmry)*cy1/Rmr + (uny[i]*Rmlx - unx[i]*Rmly)*cy1_2/Rml))*ij;
    Er[p] = Er[p] + var.real();
    Ei[p] = Ei[p] + var.imag();
  }

  if(!TM && !Es) for(j=0, p=0; j<Ni; j++) for(i=0; i<M; i++, p++)
  { R = hypot(rx[i]-rix[j],ry[i]-riy[j]);
    if(R != 0)
    { tmp = 0.25*k*dl[j]*(unix[j]*(rx[i]-rix[j]) + uniy[j]*(ry[i]-riy[j]))/R;
      z = k*R;
      cbessjy01(z, cj0, cj1, cy0, cy1, cj0p, cj1p, cy0p, cy1p);
      var = tmp*cy1 + tmp*cj1*ij;
      Hr[p] = var.real();
      Hi[p] = var.imag();
    }
    else
    { Hr[p] = -0.5;
      Hi[p] = 0;
    }
  }
}
```

get_iemei_matrices.m

Calculate the IE-MEI matrices, version 1. This and the following pieces of code are different versions of the matrices calculation (not successive versions).

```
% Calculate A and B matrices for IEMEI method in dielectrics
%
% Z_h: magnetic impedance matrix
```

```
% Z_e: electric impedance matrix
% Band = Bandwidth. -> BW + 1 (diag) + BW
%
% G. P. Conangla 2/2015

function[A, band_M, cond_vec, er_vec] = get_iemei_matrices(Z_orig, N, Band,
    Band_ext, B_force_zero)

%% preliminary checks
% determine if the band where zeros are forced (considering forcing zeros
% in only a band around the non-zero coefficients of A and B is enough) is
% too big to be used
if B_force_zero ~= 0
    use_bfz = true;
else
    use_bfz = false;
end
if Band <= 0
    disp('Choose a positive "Band"');
    disp('Changed to Band = 1');
    Band = 1;
end
% return A, B identity matrices in case Band is too big
if Band >= (N-1)/2
    A = sparse(1:N, 1:N, 1);
    return
elseif Band_ext >= (N-1)/2
    A = sparse(1:N, 1:N, 1);
    return
end
%% A, B calculation
% adapt matrices Z_e, Z_h to multiply by circular matrix
Z_sup = Z_orig((N - Band +1):N, :);
Z_inf = Z_orig(1:Band, :);
Z = [Z_sup; Z_orig; Z_inf];

% vec points which elements from the first row A*Z_e+B*Z_h must be forced to
```

```
    be zero.
% shifting the vector we can determine the same for any row
vec = make_vec(B_force_zero, Band_ext, N, Z, 1); %first row of Z_e
calc_error = zeros(N,N);
calc_error(1,:) = vec;
% other vec2
vec2 = logical([zeros(1,Band_ext + 1) ,ones(1, N-(2*Band_ext + 1)),
    zeros(1,Band_ext)]);
% vec = sparse(vec);
band_M = zeros(N, N);
% A_B will be the matrix of coefficients. To be transformed later, after
% rearranging, to matrices A and B
A_tmp = zeros(N, (1+2*Band));
cond_vec = zeros(N,1);

% time before A, B
for i = 1:N % rows of matrix A, B
    force_zero = repmat(vec,(1+2*Band),1);
    % a_b will be the vector of unkowns. The first parameter, a_{-Band},
    % is forced to be 1 (to supress trivial solution)

    % temp_z is the matrix to construct the overdetermined system.
    temp_z = Z(i: i + 2*Band,:);

    % convert column to zero in case there is no constraint for this
    % variable (it doesn't have to be zero). Only for elements in band
    % diagonals.
    % Same effect as removing column, but faster.
    temp_z = force_zero.*temp_z;
    % calculate and solve system: a = (M^t*M)^{-1}*k
    k = -temp_z(1,:).';
    M = temp_z(2:end,:).';
    % calculate a with good condition number
    if issparse(M)
        R = qr(M);
    else
        R = triu(qr(M));
```

```
    end
    a = R\(R'\(M'*k));
    % cond vector
    cond_vec(i) = cond(M' * M);
    A_tmp(i,:) = [1, a.']; %first coef. forced to 1
    band_M(i,:) = vec2;
    vec2 = circshift(vec2, [0 1]);
    if i~= N
        vec = make_vec(B_force_zero, Band_ext, N, Z, i + 1);
        vec = circshift(vec, [0 i]);
        calc_error(i,:) = vec;
    end
end
%% construct A matrix from A_tmp
% number of non_zero elements in sparse matrix A
non_zero = (2*Band + 1)*N; % <- fer sparse
A_tmp = [A_tmp, sparse(N, N-(1+2*Band))];

% prealocate space for A and B
A = spalloc(N, N, non_zero);

% shift A_temp and B_temp
for i = 0 : N-1
    A(i + 1, :) = circshift(A_tmp(i + 1,:),[0 i]);
end
er_matrix = calc_error.*(A*Z_orig);
er_vec = sum(abs(er_matrix),2); % total error
total_zeros = B_force_zero*2;
er_vec = er_vec/total_zeros; % mean error

A = circshift(A,[0 -Band]);
band_M = ~band_M;
return
```

get_iemei_matrices2.m

Calculate the IE-MEI matrices, version 2.

```
% Calculate A and B matrices for IEMEI method in dielectrics
%
% Z_h: magnetic impedance matrix
% Z_e: electric impedance matrix
% Band = Bandwidth. -> BW + 1 (diag) + BW
%
% G. P. Conangla 2/2015

function[A, B, band_M, cond_vec, er_vec] = get_iemei_matrices2(Z_e_orig,
    Z_h_orig, ef_dist, N, Band, Band_ext, B_force_zero)
%% preliminary checks
% determine if the band where zeros are forced (considering forcing zeros
% in only a band around the non-zero coefficients of A and B is enough) is
% too big to be used
if B_force_zero ~= 0
    use_bfz = true;
else
    use_bfz = false;
end
if Band <= 0
    disp('Choose a positive "Band"');
    disp('Changed to Band = 1');
    Band = 1;
end
% return A, B identity matrices in case Band is too big
if Band >= (N-1)/2
    A = sparse(1:N, 1:N, 1);
    return
elseif Band_ext >= (N-1)/2
    A = sparse(1:N, 1:N, 1);
    return
end
%% A, B calculation
% adapt matrices Z_e, Z_h to multiply by circular matrix
```

```
Z_e_sup = Z_e_orig((N - Band +1):N, :);
Z_e_inf = Z_e_orig(1:Band, :);
Z_e = [Z_e_sup; Z_e_orig; Z_e_inf];

Z_h_sup = Z_h_orig((N - Band +1):N, :);
Z_h_inf = Z_h_orig(1:Band, :);
Z_h = [Z_h_sup; Z_h_orig; Z_h_inf];

% vec points which elements from the first row A*Z_e+B*Z_h must be forced to
    be zero.
% shifting the vector we can determine the same for any row
if use_bfz
%     vec = logical([zeros(1,Band_ext + 1), ones(1,B_force_zero), zeros(1,
    N-(2*Band_ext + 1 + 2*B_force_zero)), ones(1,B_force_zero),
    zeros(1,Band_ext)]);
%     % in case bforcezero is used in three bands
%     fzero_equi = floor(2*B_force_zero/3);
%     no_cond = floor((N-(2*Band_ext + 1 + 2*B_force_zero))/2);
%     missing = N - (2*Band_ext + 1) - 3*fzero_equi - 2*no_cond;
%     vec = logical([zeros(1,Band_ext + 1), ones(1,fzero_equi), zeros(1,
    no_cond), ones(1,fzero_equi), zeros(1, no_cond), zeros(1,missing),
    ones(1,fzero_equi), zeros(1,Band_ext)]);
%     % 5 bands
%     fzero_equi = floor(2*B_force_zero/5);
%     no_cond = floor((N-(2*Band_ext + 1 + 2*B_force_zero))/4);
%     missing = N - (2*Band_ext + 1) - 5*fzero_equi - 4*no_cond;
%     vec = logical([zeros(1,Band_ext + 1), ones(1,fzero_equi), zeros(1,
    no_cond), ones(1,fzero_equi), zeros(1, no_cond), ones(1,fzero_equi),
    zeros(1, no_cond), ones(1,fzero_equi), zeros(1, no_cond),
    zeros(1,missing), ones(1,fzero_equi), zeros(1,Band_ext)]);
%     % 5 bands, extended near diagonal
%     fzero_equi = floor(2*B_force_zero/7);
%     no_cond = floor((N-(2*Band_ext + 1 + 2*B_force_zero))/4);
%     missing = N - (2*Band_ext + 1) - 7*fzero_equi - 4*no_cond;
%     vec = logical([zeros(1,Band_ext + 1), ones(1,fzero_equi),
    ones(1,fzero_equi), zeros(1, no_cond), ones(1,fzero_equi), zeros(1,
    no_cond), ones(1,fzero_equi), zeros(1, no_cond), ones(1,fzero_equi),
```

```
    zeros(1, no_cond), zeros(1,missing), ones(1,fzero_equi),
    ones(1,fzero_equi), zeros(1,Band_ext)]);
%     % uniformly distributed
%     zero_one = zeros(1,N-(2*Band_ext + 1));
%     for i = 1:length(zero_one)
%         if mod(i,5) == 1
%             zero_one(i) = 1;
%         end
%     end
%     vec = logical([zeros(1,Band_ext + 1), zero_one, zeros(1,Band_ext)]);
% % uniformly distributed
%     total_zeros = B_force_zero*2;
%     zeros_here = linspace(1, N-(2*Band_ext + 1), total_zeros);
%     zero_one = zeros(1,N-(2*Band_ext + 1));
%     for i = 1:length(zeros_here)
%         zero_one(floor(zeros_here(i))) = 1;
%     end
%     vec = logical([zeros(1,Band_ext + 1), zero_one, zeros(1,Band_ext)]);
%     % uniformly distributed with x% of forced zeros near diagonal (x/2% in
%     % each side of diagonal)
%     x = 0.5;
%     total_zeros = B_force_zero*2;
%     zeros_in_band = floor(x/2*total_zeros);
%     zero_one_1 = ones(1, zeros_in_band);
%     uniform_zeros = floor((1-x)*total_zeros);
%     zeros_here = linspace(1, N-(2*Band_ext + 1) - 2*zeros_in_band,
    uniform_zeros);
%     zero_one_2 = zeros(1, N - (2*Band_ext + 1) - 2*zeros_in_band);
%     for i = 1:length(zeros_here)
%         zero_one_2(floor(zeros_here(i))) = 1;
%     end
%     zero_one = [zero_one_1, zero_one_2, zero_one_1];
%     vec = logical([zeros(1,Band_ext + 1), zero_one, zeros(1,Band_ext)]);
    % density of zeroes defined by function
    x = 0.3;
    sigma = 0.6; % the higher/lower sigma -> the less/more linear it will be
    total_zeros = B_force_zero*2;
```

```
    zeros_in_band = floor(x/2*total_zeros);
    zero_one_1 = ones(1, zeros_in_band);
    distributed_zeros = floor((1-x)*total_zeros);
    X = linspace(-3, 3, distributed_zeros);
    Y = 1/2*(1+erf(X/(sqrt(2)*sigma)));
    Y = (Y - Y(1))/(Y(end) - Y(1)); % make sure it starts at zero and ends at 1
    zeros_here = Y*(N - (2*Band_ext + 1) - 2*zeros_in_band - 1) + 1;
    zero_one_2 = zeros(1, N - (2*Band_ext + 1) - 2*zeros_in_band);
    for i = 1:length(zeros_here)
        zero_one_2(floor(zeros_here(i))) = 1;
    end
    zero_one = [zero_one_1, zero_one_2, zero_one_1];
    vec = logical([zeros(1,Band_ext + 1), zero_one, zeros(1,Band_ext)]);
    % other vec2
    vec2 = logical([zeros(1,Band_ext + 1) ,ones(1, N-(2*Band_ext + 1)),
        zeros(1,Band_ext)]);
else
    vec = logical([zeros(1,Band_ext + 1) ,ones(1, N-(2*Band_ext + 1)),
        zeros(1,Band_ext)]);
end
calc_error = zeros(N,N);
calc_error(1,:) = vec;
where_zeros = vec;
% vec = sparse(vec);
band_M = zeros(N, N);
% A_B will be the matrix of coefficients. To be transformed later, after
% rearranging, to matrices A and B
A_B = zeros(N, 2*(1+2*Band));
cond_vec = zeros(N,1);

% time before A, B
for i = 1:N % rows of matrix A, B
    force_zero = repmat(vec,2*(1+2*Band),1);
    % a_b will be the vector of unkowns. The first parameter, a_{-Band},
    % is forced to be 1 (to supress trivial solution)

    % temp_z is the matrix to construct the overdetermined system.
```

```
    temp_ze = Z_e(i: i + 2*Band,:);
    temp_zh = Z_h(i: i + 2*Band,:);
    ZeZh = [temp_ze; temp_zh];

    % convert column to zero in case there is no constraint for this
    % variable (it doesn't have to be zero). Only for elements in band
    % diagonals.
    % Same effect as removing column, but faster.
    ZeZh = force_zero.*ZeZh;
    % calculate and solve system: a = (M^t*M)^{-1}*k
    k = -ZeZh(1,:).';
    M = ZeZh(2:end,:).';
    % calculate a with good condition number
    if issparse(M)
        R = qr(M);
    else
        R = triu(qr(M));
    end
    a_b = R\(R'\(M'*k));
    % otherwise: normal solution
    cond_vec(i) = cond(((M' * M)));
%     a = (M' * M )\M'*k; % least squares for overdetermined system
    A_B(i,:) = [1, a_b.']; %first coef. forced to 1
    if use_bfz
        band_M(i,:) = vec2;
        vec2 = circshift(vec2, [0 1]);
    else
        band_M(i,:) = vec;
    end
    where_zeros = [where_zeros; circshift(vec', i-1)'];
    vec = circshift(vec, [0 1]);
    calc_error(i,:) = vec;
end
%% construct A,B matrix from A_B
% number of non_zero elements in sparse matrix A
A_temp = A_B(:,1:(1+2*Band));
A_temp = [A_temp, zeros(N, N-(2*Band + 1))];
```

```
B_temp = A_B(:,(1+2*Band)+1:end);
B_temp = [B_temp, zeros(N, N-(2*Band + 1))];

% shift A_temp and B_temp
A = zeros(N);
B = zeros(N);
for i = 0 : N-1
    A(i + 1, :) = circshift(A_temp(i + 1,:),[0 i]);
    B(i + 1, :) = circshift(B_temp(i + 1,:),[0 i]);
end

% calc. mean error of overdetermined system
% figure(1);
% spy(calc_error);
% return
er_matrix = calc_error.*(A*Z_e_orig + B*Z_h_orig);
er_vec = sum(abs(er_matrix),2); % total error
total_zeros = B_force_zero*2;
er_vec = er_vec/total_zeros; % mean error

A = circshift(A,[0 -Band]);
B = circshift(B,[0 -Band]);
band_M = ~band_M;
return
```

get_iemei_matrices3.m

Calculate the IE-MEI matrices, version 3.

```
% Calculate A and B matrices for IEMEI method in dielectrics
%
% Z_h: magnetic impedance matrix
% Z_e: electric impedance matrix
% Band = Bandwidth. -> BW + 1 (diag) + BW
%
% G. P. Conangla 2/2015
```

```
function[A, B, band_M, cond_vec, er_vec] = get_iemei_matrices3(Z_e_orig,
    Z_h_orig, ef_dist, N, Band, Band_ext, B_force_zero)

%% preliminary checks
% determine if the band where zeros are forced (considering forcing zeros
% in only a band around the non-zero coefficients of A and B is enough) is
% too big to be used

if Band <= 0
    disp('Choose a positive "Band"');
    disp('Changed to Band = 1');
    Band = 1;
end
% return A, B identity matrices in case Band is too big
if Band >= (N-1)/2
    A = sparse(1:N, 1:N, 1);
    return
elseif Band_ext >= (N-1)/2
    A = sparse(1:N, 1:N, 1);
    return
end
%% A, B calculation
% adapt matrices Z_e, Z_h to multiply by circular matrix
Z_e_sup = Z_e_orig((N - Band +1):N, :);
Z_e_inf = Z_e_orig(1:Band, :);
Z_e = [Z_e_sup; Z_e_orig; Z_e_inf];

Z_h_sup = Z_h_orig((N - Band +1):N, :);
Z_h_inf = Z_h_orig(1:Band, :);
Z_h = [Z_h_sup; Z_h_orig; Z_h_inf];

% vec points which elements from the first row A*Z_e+B*Z_h must be forced to
    be zero.
% shifting the vector we can determine the same for any row
vec = make_vec(B_force_zero, Band_ext, N, Z_e_orig, 1); %first row of Z_e
calc_error = zeros(N,N);
calc_error(1,:) = vec;
```

```
where_zeros = vec;

% other vec2
vec2 = logical([zeros(1,Band_ext + 1) ,ones(1, N-(2*Band_ext + 1)),
    zeros(1,Band_ext)]);
band_M = zeros(N, N);
% A_B will be the matrix of coefficients. To be transformed later, after
% rearranging, to matrices A and B
A_B = zeros(N, 2*(1+2*Band));
cond_vec = zeros(N,1);

% time before A, B
for i = 1:N % rows of matrix A, B
    force_zero = repmat(vec,2*(1+2*Band),1);
    % a_b will be the vector of unkowns. The first parameter, a_{-Band},
    % is forced to be 1 (to supress trivial solution)

    % temp_z is the matrix to construct the overdetermined system.
    temp_ze = Z_e(i: i + 2*Band,:);
    temp_zh = Z_h(i: i + 2*Band,:);
    ZeZh = [temp_ze; temp_zh];

    % convert column to zero in case there is no constraint for this
    % variable (it doesn't have to be zero). Only for elements in band
    % diagonals.
    % Same effect as removing column, but faster.
    ZeZh = force_zero.*ZeZh;
    ZeZh = ZeZh(:,any(ZeZh));

%     draw_matrix(force_zero,'force_zero',1);
    % calculate and solve system: M*a = k by minimising |Ma - k|
    k = -ZeZh(1,:).';
    M = ZeZh(2:end,:).';
%     % calculate L1 min
%       Tried but not working (it has good possibilities though)

    % preconditioner
```

```
    X = M' * M;
    b = max(X.').';
    B = diag(b);
    % solve
    cond_vec(i) = cond((B*(M' * M))); % vector of condition numbers for every
                                      % least squares matrix
    a_b = (B*(M' * M))\B*M'*k; % least squares for overdetermined system
    A_B(i,:) = [1, a_b.']; %first coef. forced to 1
    band_M(i,:) = vec2;
    vec2 = circshift(vec2, [0 1]);
    if i~= N
        vec = make_vec(B_force_zero, Band_ext, N, Z_e_orig, i + 1);
        where_zeros = [where_zeros; circshift(vec', i-1)'];
        vec = circshift(vec, [0 i]);
        calc_error(i,:) = vec;
    end
end

%% construct A,B matrix from A_B
% number of non_zero elements in sparse matrix A
A_temp = A_B(:,1:(1+2*Band));
A_temp = [A_temp, zeros(N, N-(2*Band + 1))];
B_temp = A_B(:,(1+2*Band)+1:end);
B_temp = [B_temp, zeros(N, N-(2*Band + 1))];

% shift A_temp and B_temp
A = zeros(N);
B = zeros(N);
for i = 0 : N-1
    A(i + 1, :) = circshift(A_temp(i + 1,:),[0 i]);
    B(i + 1, :) = circshift(B_temp(i + 1,:),[0 i]);
end

% calc. mean error of overdetermined system
% figure(1);
% spy(calc_error);
% return
```

```
er_matrix = calc_error.*(A*Z_e_orig + B*Z_h_orig);
er_vec = sum(abs(er_matrix),2); % total error
total_zeros = B_force_zero*2;
er_vec = er_vec/total_zeros; % mean error

% finish A, B matrices
A = circshift(A,[0 -Band]);
B = circshift(B,[0 -Band]);

band_M = ~band_M; % matrix with ones where A, B are different from zero

% figure(5);
% spy(where_zeros);
return
```

get_iemei_matrices4.m

Calculate the IE-MEI matrices, version 4. This version includes the inverse method for the forced zeros distribution.

```
% Calculate A and B matrices for IEMEI method in dielectrics
%
% Z_h: magnetic impedance matrix
% Z_e: electric impedance matrix
% Band = Bandwidth. -> BW + 1 (diag) + BW
%
% G. P. Conangla 2/2015

function[A, B, band_M] = get_iemei_matrices4(Z_e_orig, Z_h_orig, ef_dist, N,
    Band, Band_ext, B_force_zero)

%% preliminary checks
% determine if the band where zeros are forced (considering forcing zeros
% in only a band around the non-zero coefficients of A and B is enough) is
% too big to be used

if Band <= 0
```

```
    disp('Choose a positive "Band"');
    disp('Changed to Band = 1');
    Band = 1;
end
% return A, B identity matrices in case Band is too big
if Band >= (N-1)/2
    A = sparse(1:N, 1:N, 1);
    return
elseif Band_ext >= (N-1)/2
    A = sparse(1:N, 1:N, 1);
    return
end
%% A, B calculation
% adapt matrices Z_e, Z_h to multiply by circular matrix
Z_e_sup = Z_e_orig((N - Band +1):N, :);
Z_e_inf = Z_e_orig(1:Band, :);
Z_e = [Z_e_sup; Z_e_orig; Z_e_inf];

Z_h_sup = Z_h_orig((N - Band +1):N, :);
Z_h_inf = Z_h_orig(1:Band, :);
Z_h = [Z_h_sup; Z_h_orig; Z_h_inf];

% vec points which elements from the first row A*Z_e+B*Z_h must be forced to
    be zero.
% shifting the vector we can determine the same for any row

% vec = make_vec(B_force_zero, Band_ext, N, Z_e_orig, 1); %first row of Z_e
% other vec2
vec2 = logical([zeros(1,Band_ext + 1) ,ones(1, N-(2*Band_ext + 1)),
    zeros(1,Band_ext)]);
% vec = sparse(vec);
band_M = zeros(N, N);
% A_B will be the matrix of coefficients. To be transformed later, after
% rearranging, to matrices A and B
A_B = zeros(N, 2*(1+2*Band));

% time before A, B
```

```
for i = 1:N % rows of matrix A, B
%     force_zero = repmat(vec,2*(1+2*Band),1);
    % a_b will be the vector of unkowns. The first parameter, a_{-Band},
    % is forced to be 1 (to supress trivial solution)

    % temp_z is the matrix to construct the overdetermined system.
    temp_ze = Z_e(i: i + 2*Band,:);
    temp_zh = Z_h(i: i + 2*Band,:);
    ZeZh = [temp_ze; temp_zh];
    ZeZh = fft(ZeZh.').';
    % fftshift for columns
%     ZeZh = fftshift(ZeZh, 2);
%     draw_matrix(ZeZh, '',1);
%     return
%     ZeZh(:, floor(N/4):floor(3*N/4)) = 0;

    % convert column to zero in case there is no constraint for this
    % variable (it doesn't have to be zero). Only for elements in band
    % diagonals.
    % Same effect as removing column, but faster.
%     ZeZh = force_zero.*ZeZh;   TO BE UPDATED, FORCE ZERO MUST REMOVE
%     LIMITS
    % calculate and solve system: M*a = k by minimising |Ma - k|
    k = -ZeZh(1,:).';
    M = ZeZh(2:end,:).';
%     % calculate L1 min
%     [~, lg] = size(M);
%     x0 = ones(lg,1);
%     a_b = l1eq_pd(x0, M' * M, 1, M'*k, 1e-3, 50, 1e-8, 200);
%     % calculate a with good condition number
%
%     if issparse(M)
%         R = qr(M);
%     else
%         R = triu(qr(M));
%     end
%     a_b = R\(R'\(M'*k));
```

```
%     otherwise: normal solution
    % preconditioner
    X = M' * M;
    b = max(X.').';
    B = diag(b);
%     [l,~] = size(X);
%     B = dftmtx(l);
    % solve
    a_b = (B*(M' * M))\B*M'*k; % least squares for overdetermined system
    A_B(i,:) = [1, a_b.']; %first coef. forced to 1
    band_M(i,:) = vec2;
    vec2 = circshift(vec2, [0 1]);
%     if i~= N
%         vec = make_vec(B_force_zero, Band_ext, N, Z_e_orig, i + 1);
%         vec = circshift(vec, [0 i]);
%     end
end

%% construct A,B matrix from A_B
% number of non_zero elements in sparse matrix A
A_temp = A_B(:,1:(1+2*Band));
A_temp = [A_temp, zeros(N, N-(2*Band + 1))];
B_temp = A_B(:,(1+2*Band)+1:end);
B_temp = [B_temp, zeros(N, N-(2*Band + 1))];

% shift A_temp and B_temp
A = zeros(N);
B = zeros(N);
for i = 0 : N-1
    A(i + 1, :) = circshift(A_temp(i + 1,:),[0 i]);
    B(i + 1, :) = circshift(B_temp(i + 1,:),[0 i]);
end

A = circshift(A,[0 -Band]);
B = circshift(B,[0 -Band]);
X = A*fft(Z_e_orig,[],2) + B*fft(Z_h_orig,[],2);
draw_matrix(X,'AZe + BZh',3);
```

```
band_M = ~band_M;

figure(5);
spy(band_M);
return
```

make_vec.m

Integral part of `get_iemei_matrices4.m`. Calculates $F^{-1}(U)$.

```
% make_vec(B_force_zero, Band_ext, N, M_height, row)
%
% Calculate the vector with the zeros distributed according to geometry
%
% GP Conangla 2015

function[vec] = make_vec(B_force_zero, Band_ext, N, M_height, row)

% density of zeroes defined by Z_e
x = 0.4;
total_zeros = B_force_zero*2;
zeros_in_band = floor(x/2*total_zeros);

current_row = M_height(row,:);
current_row = circshift(current_row, [0 -row + 1]);
current_row = current_row(Band_ext + 2 : N-Band_ext);
current_row = current_row(zeros_in_band + 1: end - zeros_in_band);

num_zeros_distributed = total_zeros - 2*zeros_in_band;
zero_one_1 = ones(1, zeros_in_band);

density_zeros = (abs(current_row)).^2; %the squaring is just an option, any
    monotonous will do

cumul_zeros = cumsum(density_zeros);
cumul_zeros = (cumul_zeros - cumul_zeros(1))/(cumul_zeros(end) -
    cumul_zeros(1)); % normalize
```

```
zeros_here = cumul_zeros*(N - (2*Band_ext + 1) - 2*zeros_in_band) + 1;

zero_one_2 = zeros(1, N - (2*Band_ext + 1) - 2*zeros_in_band);

put_zeros = linspace(1, length(zero_one_2), num_zeros_distributed);

for i = 1:num_zeros_distributed
    [~, m] = min(abs(put_zeros(i) - zeros_here)); % find closer position to
       put_zeros(i)
    zero_one_2(m) = 1;
end

zero_one = [zero_one_1, zero_one_2, zero_one_1];
vec = logical([zeros(1,Band_ext + 1), zero_one, zeros(1,Band_ext)]);
end
```

operator.m

Adapter for the previous C++ function, to make its access easier.

```
% transform the programmed oper_die_2d into Omega and Lambda operators
% for readability

function[M] =
    operator(z,un,z_2,un_2,dl,operator_type,current_type,k_0,e_r,mu_r,N)

TM = 1;
e_r_re = real(e_r);
e_r_im = imag(e_r);
mu_r_re = real(mu_r);
mu_r_im = imag(mu_r);

% case of real relative permittivity and permeability -> faster
if e_r_im == 0 && mu_r_im == 0 && e_r_re > 0 && mu_r_re > 0
    switch current_type
        case 'J'
            switch operator_type
```

```
            case 'Omega'
                M = -(oper_die_2d(z,un,z.',un,dl,k_0,TM,0,e_r_re,mu_r_re));
                    % mfie
            case 'Lambda'
                M = -oper_die_2d(z,un,z_2,un_2,dl,k_0,TM,1,e_r_re,mu_r_re);
                    % efie
        end
    case 'M'
        switch operator_type
            case 'Omega'
                M = oper_die_2d(z,un,z_2,un_2,dl,k_0,~TM,0,e_r_re,mu_r_re);
                    %efie
            case 'Lambda'
                M = -oper_die_2d(z,un,z_2,un_2,dl,k_0,~TM,1,e_r_re,mu_r_re);
                    % mfie
        end
end
else % complex relative permittivity/permeability
    switch current_type
    case 'J'
        switch operator_type
            case 'Omega'
                M =
                    -(oper_die_2d_complex(z,un,z.',un,dl,k_0,TM,0,e_r_re,e_r_im,mu_r_re,mu_r_im)
                    %mfie
            case 'Lambda'
                M =
                    -oper_die_2d_complex(z,un,z_2,un_2,dl,k_0,TM,1,e_r_re,e_r_im,mu_r_re,mu_r_im
                    % efie
        end
    case 'M'
        switch operator_type
            case 'Omega'
                M =
                    oper_die_2d_complex(z,un,z_2,un_2,dl,k_0,~TM,0,e_r_re,e_r_im,mu_r_re,mu_r_im
                    % efie
            case 'Lambda'
```

```
                M =
                    -oper_die_2d_complex(z,un,z_2,un_2,dl,k_0,~TM,1,e_r_re,e_r_im,mu_r_re,mu_r_ir
                    % mfie
        end
    end
end
% % case of real relative permittivity and permeability -> faster
% if e_r_im == 0 && mu_r_im == 0 && e_r_re > 0 && mu_r_re > 0
%     switch current_type
%         case 'J'
%             switch operator_type
%                 case 'Omega'
%                     M = -(oper_die_2d(z,un,z.',un,dl,k_0,TM,0,e_r_re,mu_r_re)
    - 0.5*eye(N,N)); % mfie
%                 case 'Lambda'
%                     M = oper_die_2d(z,un,z_2,un_2,dl,k_0,TM,1,e_r_re,mu_r_re);
    % efie
%             end
%         case 'M'
%             switch operator_type
%                 case 'Omega'
%                     M = oper_die_2d(z,un,z_2,un_2,dl,k_0,~TM,0,e_r_re,mu_r_re)
    + 0.5*eye(N,N); % efie
%                 case 'Lambda'
%                     M =
    -oper_die_2d(z,un,z_2,un_2,dl,k_0,~TM,1,e_r_re,mu_r_re); % mfie
%             end
%     end
% else % complex relative permittivity/permeability
%     switch current_type
%     case 'J'
%         switch operator_type
%             case 'Omega'
%                 M =
    -(oper_die_2d_complex(z,un,z.',un,dl,k_0,TM,0,e_r_re,e_r_im,mu_r_re,mu_r_im)
    - 0.5*eye(N,N)); % mfie
%             case 'Lambda'
```

```
%               M =
    oper_die_2d_complex(z,un,z_2,un_2,dl,k_0,TM,1,e_r_re,e_r_im,mu_r_re,mu_r_im);
    % efie
%       end
%     case 'M'
%       switch operator_type
%           case 'Omega'
%               M =
    oper_die_2d_complex(z,un,z_2,un_2,dl,k_0,~TM,0,e_r_re,e_r_im,mu_r_re,mu_r_im)
    + 0.5*eye(N,N); % efie
%           case 'Lambda'
%               M =
    -oper_die_2d_complex(z,un,z_2,un_2,dl,k_0,~TM,1,e_r_re,e_r_im,mu_r_re,mu_r_im);
    % mfie
%       end
%     end
% end

end
```

efie_matrix.m

Generate the EFIE matrix.

```
% EFIE matrix for dielectrics

function[Z_ej_1, Z_ej_2, Z_em_1, Z_em_2] =
    efie_matrix(z,un,dl,k_0,e_r1,mu_r1,e_r2,mu_r2,N,pc1)

% constants
eta_0 = 119.9169*pi; % in the void
eta_r1 = sqrt(mu_r1/e_r1);
eta_r2 = sqrt(mu_r2/e_r2);
eta_1 = eta_0*eta_r1;
eta_2 = eta_0*eta_r2;

% impedance matrix
```

```
% EFIE dielectric equations
% 1st box
Z_ej_1 = operator(z,un,z.',un,dl,'Lambda','J',k_0,e_r1,mu_r1,N);

% 2nd box
Z_em_1 = operator(z,un,z.',un,dl,'Omega','M',k_0,e_r1,mu_r1,N);

% 3rd box
Z_ej_2 = operator(z,un,z.',un,dl,'Lambda','J',k_0,e_r2,mu_r2,N);

% 4th box
Z_em_2 = operator(z,un,z.',un,dl,'Omega','M',k_0,e_r2,mu_r2,N);

end
```

mfie_matrix.m

Generate the MFIE matrix.

```
% MFIE matrix for dielectrics
%
%
%

function[Z_hj_1, Z_hj_2, Z_hm_1, Z_hm_2] =
    mfie_matrix(z,un,dl,k_0,e_r1,mu_r1,e_r2,mu_r2,N,pc1)

% constants
eta_0 = 119.9169*pi; % in the void
eta_r1 = sqrt(mu_r1/e_r1);
eta_r2 = sqrt(mu_r2/e_r2);
eta_1 = eta_0*eta_r1;
eta_2 = eta_0*eta_r2;

% impedance matrix
```

```
% MFIE dielectric equations
% 1st box
Z_hj_1 = operator(z,un,z.',un,dl,'Omega','J',k_0,e_r2,mu_r2,N);

% 2nd box
Z_hm_1 = operator(z,un,z.',un,dl,'Lambda','M',k_0,e_r2,mu_r2,N)/eta_2^2;

% 3rd box
Z_hj_2 = operator(z,un,z.',un,dl,'Omega','J',k_0,e_r1,mu_r1,N);

% 4th box
Z_hm_2 = operator(z,un,z.',un,dl,'Lambda','M',k_0,e_r1,mu_r1,N)/eta_1^2;

end
```

get_electric_field.m

Calculate the convolution integral with the currents and Green's function to find the near field.

```
% Calculate Green integrals: get E field from electric & magnetic current
% ref: Integral Equation Methods for Electromagnetics, Morita et al (pg 156,
    3.115, 3.116);
%
% Inputs:
% J: electric current
% M: magnetic current
% N: discretization number
% dl: distance between disc. points -> h: step size
% geometry: geometry of object
% geo_parameters: semiaxis in x, y
% deg: degree of superellipse (only in pill geometry case)
% zl: disc. points
% normal: normals in disc. points
% kr: k* bigger semiaxis
% k_1: k of medium (usually free space)
% k_2: k in object
```

```
% eta_0: impedance in free space
% eta_2: impedance of object
%
% Outputs:
% E: electric field
% x: x coordinates mesh
% y: y coordinates mesh
%
% G. Planes Conangla 5/2015

function[E, x, y] = get_electric_field(J, M, N, dl, geometry, geo_parameters,
    deg, zl, normal, kr, k_1, k_2, eta_0, eta_2)

% Number of points in x and y
Npun = 160; % must be multiple of 4

eps = 1e-3; % Small shift to avoid singularity of G(r-rn)
x = linspace(-kr/2 - eps, kr/2 + eps, Npun);
y = x;
[rx,ry] = meshgrid(x,y); % make the mesh
r = rx + 1j*ry;

% Scattered field outside
Es = zeros(Npun, Npun);

for n = 1:N
    h = dl(n);
   R = abs(r - zl(n));         % Distance from points of boundary to r
   G = besselh(0, 2, k_1*R)/(4*1j); % Green's function
    % now calculate diff in normal direction of G (for M current part)
    delta = normal(n)/(abs(normal(n))*1000); % increment for differentiation
    R_up_delta = abs(r - (zl(n) + delta));
    G_up_delta = besselh(0, 2, k_1*R_up_delta)/(4*1j);
    R_down_delta = abs(r - (zl(n) - delta));
    G_down_delta = besselh(0, 2, k_1*R_down_delta)/(4*1j);
    dif_n_G = (G_up_delta - G_down_delta)/(2*abs(delta)); % central finite
        diff.
```

```
   Es = Es + (M(n)*dif_n_G)*h - (1j*k_1*eta_0*J(n)*G)*h; % integrate
end

Ei = exp(-1j*k_1*real(r));
E = Es + Ei; % total field outside = scattered + incident

% Scattered field inside scatterer
r_small = r(Npun/4 + 1: 3/4*Npun, Npun/4 + 1: 3/4*Npun); % extract submatrix
    from r
Es = zeros(Npun/2, Npun/2);

for n = 1:N
    h = dl(n);
   R = abs(r_small - zl(n));   % Distance from points of boundary to r
   G = besselh(0, 2, k_2*R)/(4*1j); % Green's function
    % now calculate diff in normal direction of G (for M current part)
    delta = normal(n)/(abs(normal(n))*1000); % increment for differentiation
    R_up_delta = abs(r_small - (zl(n) + delta));
    G_up_delta = besselh(0, 2, k_2*R_up_delta)/(4*1j);
    R_down_delta = abs(r_small - (zl(n) - delta));
    G_down_delta = besselh(0, 2, k_2*R_down_delta)/(4*1j);
    dif_n_G = (G_up_delta - G_down_delta)/(2*abs(delta)); % central finite
        diff.
   Es = Es + (M(n)*dif_n_G)*h - (1j*k_1*eta_2*J(n)*G)*h; % integrate
end

Es = -Es; % inside: opposite sign
% insert inside of scatterer in E matrix
big_Es = zeros(Npun, Npun);
big_Es(Npun/4 + 1: 3/4*Npun, Npun/4 + 1: 3/4*Npun) = Es;

% detect interior of scatterer and put right field inside
switch geometry
    case 'elliptical'
        ind = find((abs(real(r))/geo_parameters(1)).^2 +
            (abs(imag(r))/geo_parameters(2)).^2 < 1);
    case 'pill'
```

```
        ind = find((abs(real(r))/geo_parameters(1)).^deg +
            (abs(imag(r))/geo_parameters(2)).^deg < 1);
    case 'rectangular'
        ind = find(abs(real(r)) < geo_parameters(1) & abs(imag(r)) <
            geo_parameters(2));
end

E(ind) = big_Es(ind); % change field inside of scatterer

end
```

9.3.2 Geometry routines

circular.m

Circular geometry boundary generator.

```
% z = circular(N,param)
% Circular boundary
%
% N   : number of basis functions
% param : radius
%
% z   : sampling points, left end of basis functions
%    z(N+1) = z(1)
% 2-D vectors are represented by complex numbers z = x + j*y
%
% MEI v3.0, Juan M. Rius, Sept. 1996

function z=circular(N,param)

R = param(1);

t = linspace(0,2*pi,N+1)';
z = R * exp(1i*t);
```

elliptical.m

Elliptical geometry boundary generator

```
% z = elliptical(N, param)
% Elliptical boundary
%
% N   : number of basis functions
% param : semiaxis in x and in y
%
% z   : sampling points, left end of basis functions
%   z(N+1) = z(1)
% 2-D vectors are represented by complex numbers z = x + j*y
%
% Juan M. Rius, Sept. 1996
% G Planes Conangla 5/2015

function z = elliptical(N, param)

a = param(1);
b = param(2);

t = linspace(0,2*pi,N+1)';
z = a*cos(t) + 1j*b*sin(t);
```

rectang_geometry.m

Rectangular geometry boundary generator.

```
% z = rectang_geometry(N,param)
% Rectangular boundary
%
% N     : number of basis functions
% geo_parameters: half the sides in x, y
% z     : sampling points, left end of basis functions
%
% Returns column vector with N+1 elements: z(N+1)=z(1)
% 2-D vectors are represented by complex numbers z = x + 1j*y
```

```
%
% Gerard Planes Conangla, May 2015

function z = rectang_geometry(N, geo_parameters)

lx = 2*geo_parameters(1); % width of rectangle in x
ly = 2*geo_parameters(2); % width of rectangle in y
L2  = lx + ly;

N12 = floor(N/2);
N34 = N-N12;
p1 = round(N12*lx/L2); % Points in x direction
p2 = N12 - p1;     % Points in y direction
p3 = round(N34*lx/L2);
p4 = N34 - p3;

i1 = lx/p1; i3 = lx/p3;  % Step in x direction
i2 = ly/p2; i4 = ly/p4;  % Step in y direction

z1 = linspace(-lx/2 - 1j*ly/2, lx/2 - i1 - 1j*ly/2, p1); % bottom
z2 = linspace( lx/2 - 1j*ly/2, lx/2 + 1j*(ly/2-i2), p2); % rigth
z3 = linspace( lx/2 + 1j*ly/2, -lx/2 + i3 + 1j*ly/2, p3); % upper
z4 = linspace(-lx/2 + 1j*ly/2, -lx/2 - 1j*(ly/2-i4), p4); % left
z=[z1 z2 z3 z4 z1(1)].';
```

pill.m

Superelliptical geometry boundary generator.

```
% z = pill(N, param)
% "Pill" boundary (rectangular boundary with smooth vertices (specifically
%                  a superellipse))
%
% N   : number of basis functions
% param : semiaxis in x, y
%
% z   : sampling points, left end of basis functions
```

```
%    z(N+1) = z(1)
% deg: degree of superellipse used
% 2-D vectors are represented by complex numbers z = x + j*y
%
% G Planes Conangla 5/2015

function [z, deg] = pill(N, param)

a = param(1);
b = param(2);

deg = 4; % this number can be changed: the bigger the number, the closer
         % to a rectangle. deg = 2 gives an ellipse as result

t = linspace(0,2*pi,N+1)'; % equispaced angles for circle
k_equi = zeros(N + 1, 1); % equispaced in superellipse

for i = 1:length(t)
    cos_x = cos(t(i));
    sin_x = sin(t(i));
    angle_fun = @(k) (k*cos_x/a).^deg + (k*sin_x/b).^deg - 1;
    k_equi(i) = fzero(angle_fun, 1);
end
z_circ = exp(1i*t);
z = z_circ.*k_equi;
```

dot.m

Dot product in the 2D complex sense used in the project.

```
% Dot product of 2-D vectors represented by complex numbers
%
% p = dot(a,b)
%
% If a and b are vectors, p is a vector p(n) = dot(a(n),b(n))

function p=dot(a,b)
```

```
p = real(a.*conj(b));
```

normals.m

Return normals to the surface from a given geometry.

```
% Obtain unit normals and length of basis functions
%
% [un,dl,zc] = normals(zl)
%
% zl  = sampling points, left of pulse basis functions.
%  For closed objects, z(N+1) = z(1)
%  For open objects z(N+1) is right end of last basis function
% un = exterior unit normals
% dl = length of basis functions
% 2-D vectors are represented by complex numbers z = x + j*y
%
% IE-MEI, version 3.0, Juan M. Rius, sept. 1996

function [un,dl,zc]=normals(zl)

N  = length(zl)-1;
t  = zl(2:N+1) - zl(1:N);
un = -1i*sign(t);
dl = abs(t);
zc = (zl(1:N)+zl(2:N+1))/2;
```

9.3.3 Others

band_matrix.m

Generate a circulant band matrix.

```
% Obtain matrix with ones in the three diagonal bands
%
% G. P. Conangla 2/2015
```

```
function[tri_band] = band_matrix(N, Band)

if Band >= (N - 1)/2
    tri_band = ones(2*N,2*N);
    return
end

tri_band = zeros(2*N,2*N);

vec = [zeros(1,Band + 1), ones(1, N-(2*Band + 1)), zeros(1,2*Band + 1),
    ones(1, N-(2*Band + 1)), zeros(1,Band)];
for i = 1:2*N % rows of matrix A, B
    for j = 1:2*N % columns of Z_e, Z_h
        if j >= i % col >= row
            include = vec(1 + j - i);
        else
            include = vec(2*N + 1 - (i - j));
        end
        if include == false
            tri_band(i,j) = 1;
        end
    end
end

return
```

draw_matrix.m

Plot adequately a matrix.

```
% draw matrix with good orientation, axis and title
%
function[] = draw_matrix(M,name,h_1)

if issparse(M)
    M = full(M);
```

```
end
if h_1 == 1 || h_1 == 2 || h_1 == 3
   figure(h_1);
else
    axes(h_1);
end
cla reset;
A = log10(abs(M.'));
A = A - max(max(A));
% fprintf('Biggest element of figure %f is %f\n',h_1,max(max(abs(M))));
surf(A);
shading interp;
text = strcat({'Matrix '}, name, {' in log_{10} scale'});
title(text);
dim1 = size(M,1);
dim2 = size(M,2);
axis([1 dim1 1 dim2]);
xlabel('Rows');
ylabel('Columns');
colorbar;
end
```

make_sparse.m

```
% make M sparse by truncating M
% the statistics of sparsity are taken from just one row to increase speed
% spars is the wanted sparsity

function[M, k] = make_sparse(M, spars)

% return if operation is redundant
if sparsity(M(1,:)) < spars
    k = 0;
    return
end
top = abs(max(max(M)));
```

```
M_1 = M(1,:);
% implementation of bisection method to find best k to truncate
% (Newton - Raphson is problematic in this case, already tried)
k_0 = 0;
k_1 = 1;
% function
f = @(k) sparsity(trunc_matrix(M_1, top, k)) - spars;
k = (k_0 + k_1)/2;
f_k = f(k);
% control time (if it takes too much, exit)
t_ini = toc;
limit = 0.1;
while abs(f_k)/spars > 0.01
    if f_k < 0
        k_1 = k;
    else
        k_0 = k;
    end
    k = (k_0 + k_1)/2;
    f_k = f(k);
    exec_time = toc - t_ini;
    % leave if execution time is increasing too much
    if exec_time > limit
        k = 0.01;
        fprintf('\nGiven value of sparsity of Z_em2 not possible, used
            %f\n\n', k);
        break
    end
end
M = (trunc_matrix(M, top, k));
end
```

plot_error_geo.m

Plot the error of the method in each element of the discretization.

```
% plot error in least squares minimisation of IEMEI per element
```

```
% over the element itself.
%
function[] = plot_error_geo(zl, er_vec, fig_2)

axes(fig_2);
cla reset;
scat_x = real(zl(1:end-1));
scat_y = imag(zl(1:end-1));
scatter(fig_2, scat_x, scat_y, 25, er_vec,'filled');
colorbar;
lim_axis = max(max(scat_x, scat_y));
axis([-1.2*lim_axis 1.2*lim_axis -1.2*lim_axis 1.2*lim_axis])
title('Error in IEMEI per element');
end
```

plot_field.m

Plot the near field.

```
% plot fielld

function[] = plot_field(E, x, y, zl, h_1)

% plot field
axes(h_1);
cla reset;
hold on;
pcolor(x,y,abs(E));
colormap(jet);
shading('interp');
colorbar;
title('E field amplitude');
xlabel('x'),
ylabel('y');
axis([x(1) x(end) x(1) x(end)]);
% add scatterer boundary
scat_x = real(zl);
```

```
scat_y = imag(zl);
plot(scat_x, scat_y, 'm--', 'LineWidth', 2);


% % plot field with time evolution
% axes( );
% cla reset;
% colormap(jet);
% h = colorbar;
% top = max(max(abs(E)));
% low = -top;
% set(h, 'ylim', [low top]);
% axis([x(1) x(end) x(1) x(end)]);
% % add scatterer boundary
% scat_x = real(zl);
% scat_y = imag(zl);
% % plot, first with animation
% wt = linspace(0,2*pi,40);
% i = 1;
% Z1 = abs(real(exp(1j*wt(i))*E));
% h_surf = pcolor(x, y, Z1);
% hold on;
% plot(scat_x, scat_y, 'm--', 'LineWidth', 2);
% title('Intensity of field');
% xlabel('x'),
% ylabel('y');
% while i < 5*length(wt)
%     j = mod(i, length(wt));
%     if j == 0;
%         j = j + 1;
%     end
%     Z = abs(real(exp(1j*wt(j))*E));
%     set(h_surf, 'Cdata',Z);
%     shading('interp');
%     % MATLAB pauses for 0.05 sec before moving on to execute the next plot
%     % the effect will be that of an animation
%     pause(0.05);
```

```
%     i = i + 1;
% end

end
```

plot_results.m

Plot the found electric and magnetic currents.

```
% Plot graphs
%
%
%
function[] = plot_results(equations, eta_0, J, M, h_1, h_2, varargin)

% Electric current
axes(h_1);
cla reset;
hold on;
plot(eta_0*real(J),'r');
plot(eta_0*imag(J),'b');
title([upper(equations) ' ' 'Electric current']);
legend('real','imag');

% Magnetic current
axes(h_2);
cla reset;
hold on;
plot(real(M),'r');
plot(imag(M),'b');
title([upper(equations) ' ' 'Magnetic current']);
legend('real','imag');
% iemei comparison with efie
switch equations
    case {'iemei_efie','iemei_mfie','iemei'}
        J_e = varargin{1};
        M_e = varargin{2};
```

```
        % Electric current
        axes(h_1);
        title('IEMEI: Electric current');
        hold on;
        plot(eta_0*real(J_e),'r.-');
        plot(eta_0*imag(J_e),'b.-');

        % Magnetic current
        axes(h_2);
        title('IEMEI: Magnetic current');
        hold on;
        plot(real(M_e),'r.-');
        plot(imag(M_e),'b.-');
end
end
```

plot_results_modphase.m

Plot the modulus and phase of the currents.

```
% Plot graphs
%
%
%
function[] = plot_results_modphase(equations, eta_0, J, M, h_3, h_4, h_5, h_6,
    varargin)

% Electric current
axes(h_3);
% figure(3);
cla reset;
hold on;
plot(eta_0*abs(J),'r');
title('Modulus of electric current');
axes(h_4);
% figure(4);
cla reset;
```

```
hold on;
plot(angle(J),'r');
title('Phase of electric current');

% Magnetic current
axes(h_5);
% figure(5);
cla reset;
hold on;
plot(abs(M),'r');
title('Modulus of magnetic current');
axes(h_6);
% figure(6);
cla reset;
hold on;
plot(angle(M),'r');
title('Phase of magnetic current');
% iemei comparison with efie
switch equations
    case {'iemei_efie','iemei_mfie','iemei'}
        J_e_efie = varargin{1};
        M_e_efie = varargin{2};
        J_e_mfie = varargin{3};
        M_e_mfie = varargin{4};
        % Electric current
        axes(h_3);
% figure(3);
        hold on;
        plot(eta_0*abs(J_e_efie),'b.-');
        plot(eta_0*abs(J_e_mfie),'g.-');
        legend('IEMEI solution','EFIE solution','MFIE solution');
        axes(h_4);
% figure(4);
        hold on;
        plot(angle(J_e_efie),'b.-');
        plot(angle(J_e_mfie),'g.-');
        legend('IEMEI solution','EFIE solution','MFIE solution');
```

```
        % Magnetic current
        axes(h_5);
% figure(5);
        hold on;
        plot(abs(M_e_efie),'b.-');
        plot(abs(M_e_mfie),'g.-');
        legend('IEMEI solution','EFIE solution','MFIE solution');
        axes(h_6);
% figure(6);
        hold on;
        plot(angle(M_e_efie),'b.-');
        plot(angle(M_e_mfie),'g.-');
        legend('IEMEI solution','EFIE solution','MFIE solution');
end
end
```

print_results.m

Print the results of the simulation in txt format.

```
% Display calculation times, values used and results
% (on command line or gui)
%
%

function[str] =
    print_results(N,kr,e_r2,mu_r2,BW,BW_ext,En_bw,cond_vec,cond_prob,cond_current,er_vec,t1,t2,
    order_erm, sp_AZ,sp_Z_trunc,equations)
%     str =
    sprintf('\n%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%\n');
    str = sprintf('--SCATTERER--\n');
    str = sprintf('%sNumber of elements in scatterer: %d\n',str, N);
    str = sprintf('%sValue of r/lambda: %.1f/2pi\n',str, kr);
    if imag(e_r2) < 0
        str = sprintf('%se_r: %.2f - %.2fj mu_r: %.2f\n\n',str, e_r2,
            abs(imag(e_r2)/1j), mu_r2);
    else
```

```
    str = sprintf('%se_r: %.2f + %.2fj mu_r: %.2f\n\n',str, e_r2, e_r2/1j,
        mu_r2);
end
switch equations
    case {'iemei_efie','iemei_mfie','iemei'}
        str = sprintf('%s--BANDWIDTH and SPARSITY--\n',str);
        str = sprintf('%sA bandwidth: %d\n',str, BW);
        str = sprintf('%sCalculated BW in En: %d\n',str, BW_ext);
        str = sprintf('%sActual BW used: %d\n',str,En_bw);
        str = sprintf('%sSparsity of: Zc:     %.2f%%\n',str,sp_AZ*100);
        str = sprintf('%s                Z_J->M:
            %.2f%%\n\n',str,sp_Z_trunc*100);
        str = sprintf('%s--TIME--\n',str);
        str = sprintf('%sPre-calculation steps: %f s\n',str, t1);
        str = sprintf('%sZ matrices calculation: %f s\n',str, t_iemei - t1);
        str = sprintf('%sIEMEI matrix calculation: %f s\n',str, t2 -
            t_iemei);
        str = sprintf('%sSystem solving: %f s\n',str, t3 - t2);
        str = sprintf('%sTotal time: %f s\n\n',str, t3);
        str = sprintf('%s--ERROR--\n',str);
        str = sprintf('%sCond. num. of: Current matrix (before trunc): %.2g
            IEMEI matrix (before trunc):
            %.2g\n',str,cond_prob,cond_current);
        str = sprintf('%sCond. num. of IEMEI least squares matrices: min:
            %.2g max: %.2g mean: %.2g\n',str,min(cond_vec), max(cond_vec),
            mean(cond_vec));
        str = sprintf('%sError per element in least squares: min: %.2g max:
            %.2g mean: %.2g\n\n',str,min(er_vec), max(er_vec),
            mean(er_vec));
        str = sprintf('%sRelative L2 error in J between MFIE/EFIE:
            %f\n',str, order_erj);
        str = sprintf('%sRelative L2 error in M between MFIE/EFIE:
            %f\n',str, order_erm);
        str = sprintf('%sRelative L2 error in J (with MFIE): %f\n',str,
            er1);
        str = sprintf('%sRelative L2 error in M (with MFIE): %f\n\n',str,
            er2);
```

```
            str = sprintf('%sMean L2 error: %f\n\n',str, (er1 + er2)/2);

        otherwise % EFIE or MFIE
            str = sprintf('%s--TIME--\n',str);
            str = sprintf('%sPre-calculation steps: %f s\n',str, t1);
            str = sprintf('%sZ matrices calculation: %f s\n',str, t2 - t1);
            str = sprintf('%sSystem solving: %f s\n',str, t3 - t2);
            str = sprintf('%sTotal time: %f s\n\n',str, t3);
    end
end
```

sparsity.m

Determine the sparsity of a given matrix.

```
function[sp] = sparsity(M)

sp = nnz(M)/(size(M,1)*size(M,2));

end
```

trunc_matrix.m

Truncate a matrix for a given threshold.

```
% truncate matrix M. All elements < k*max_element
% set to zero

function[M] = trunc_matrix(M, k, varargin)

if nargin > 2
    M(abs(M) < k*varargin{1}) = 0;
else
%     M(abs(M) < k*abs(max(max(M)))) = 0;
    for i = 1:length(M(1,:))
        E = M(i,:);
        E(abs(E) < k*abs(max(E))) = 0;
```

```
        M(i,:) = E;
    end
end

end
```

Bibliography

[1] N. Morita, N. Kumagai, J.R. Mautz, INTEGRAL EQUATION METHODS FOR ELECTROMAGNETICS, Artech House, 1990

[2] J.M. Rius, R. Pous, A. Cardama, INTEGRAL FORMULATION OF THE MEASURED EQUATION OF INVARIANCE: A NOVEL SPARSE MATRIX BOUNDARY ELEMENT METHOD. IEEE Transactions on Antennas and Propagation, vol. 32 No. 3, May 1996

[3] Juan M. Rius, NUMERICAL ADAPTIVE TESTING FUNCTIONS FOR APPROXIMATE SPARSE MATRIX METHOD OF MOMENTS. PART 1: PERFECTLY CONDUCTING SCATTERER. October 1996

[4] A. Cardama, L. Jofre, J.M. Rius et al., ANTENAS (2nd edition). Edicions UPC, Barcelona 2002

[5] Korada Umashankar, Allen Taflove, COMPUTATIONAL ELECTROMAGNETICS, Artech House, 1993

[6] R.F. Harrington. FIELD COMPUTATION BY MOMENT METHOD, McGraw-Hill, New York 1968

[7] C.A. Pérez Carpintero, J.M Rius Casals, DESARROLLO Y APLICACIONES DEL MEI EN OBJETOS CONDUCTORES DE DOS DIMENSIONES (PFC). Departament de teoria de senyal i comunicacions, ETSETB, UPC, Barcelona 1996

[8] K.K. Mei, R.Pous et al., MEASURED EQUATION OF INVARIANCE: A NEW CONCEPT IN FIELD COMPUTATIONS. IEEE Transactions on Antennas and Propagation, vol. 42, no. 3, March 1994

[9] Stefan A. Maier, PLASMONICS: FUNDAMENTALS AND APPLICATIONS. Springer Science+Business Media LLC 2007.

[10] J.M. Rius, C.P Carpintero, A. Cardama, J.R. Mosig, THE THEORETICAL ERROR IN THE INTEGRAL EQUATION MEI. IEEE Electronic Letters 1996.

[11] Jovan O. Jevtic, Robert Lee, A THEORETICAL AND NUMERICAL ANALYSIS OF THE MEASURED EQUATION OF INVARIANCE. IEEE Transactions on Antennas and Propagation, vol 42. No. 8, August 1994.

[12] Joel A. Tropp, Anna C. Gilbert, SIGNAL RECOVERY FROM RANDOM MEASUREMENTS VIA ORTHOGONAL MATCHING PURSUIT. IEEE Transactions on Information Theory, vol. 53, no. 12, December 2007

[13] D.M. Solís, M.G. Araújo, L. Landesa et al, MLFMA-MOM FOR SOLVING THE SCATTERING OF DENSELY PACKED PLASMONIC NANOPARTICLE ASSEMBLIES. IEEE Photonics Journal, vol. 7, n. 3, June 2015.

[14] C.A. Balanis, ADVANCED ENGINEERING ELECTROMAGNETICS. Ed. Wiley, May 1989

[15] J.M. Rius, E. Úbeda, J. Parrón, THE INTEGRAL EQUATION MEI APPLIED TO THREE-DIMENSIONAL ARBITRARY SURFACES. Electronic Letters, vol. 33, Issue 24, November 1997.

[16] J.M Rius, C.P. Carpintero, A. Cardama, K.A. Michalski. ANALYSIS OF ELECTRICALLY LARGE CONCAVE SCATTERERS WITH THE INTEGRAL EQUATION MEI. Microwave and Optical Technology Letters, 14(5), pp. 287-289, 5th April 1997

[17] J. Song, C. C. Lu, and W. C. Chew, MULTILEVEL FAST MULTIPOLE ALGORITHM FOR ELECTROMAGNETIC SCATTERING BY LARGE COMPLEX OBJECTS. Antennas and Propagation, IEEE Transactions vol. 45, no. 10, pp. 1488–1493, Oct 1997.