2020-06

MODELO COMPUTACIONAL PARA SIMULAR LA DISPERSION DE ONDASELECTROMAGNETICAS EN SENSORES DE TENSION EN POLIMEROS, UTILIZANDO APROXIMACIONES POTENCIALES

VELASQUEZ CARCAMO, RENE EDUARDO

Anexo A

Fundamentos teóricos matemáticos.

A.1 Cálculo vectorial.

La palabra vector significa 'que conduce', fisicamente hablando es un elemento que posee magnitud, dirección y sentido. Siendo la magnitud la longitud del vector, la dirección la oriendación de la flecha, el sentido indica hacia el lado donde se dirige el vector. Su expresión geométrica es la de una recta.

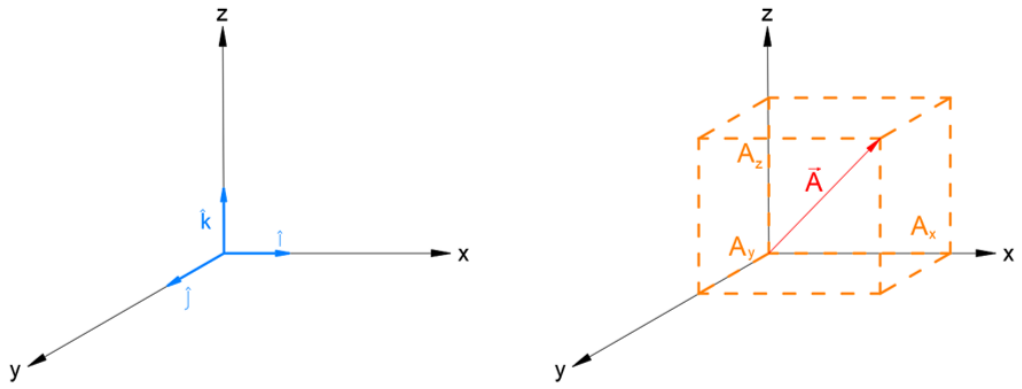


Figura A.1: Representación y suma de vectores

Además de los vectores también existen los escalares para asignar magnitudes, estos solo poseen magnitud. Los vectores pueden sumarse y restarse, es importante notar que se debe tomar en cuenta la dirección y sentido.
El sistema de referencia para ubicar el vector es usualmente el sistema de coordenadas cartesianas, con ejes denotados como x, y y z. Para mayor facilidad de escritura se asignan vectores unitarios a cada uno de los ejes, siendo $\hat{\imath}$, $\hat{\jmath}$ y $\hat{k}$
Ahora denotemos un vetor $\vec{r} \in \mathbb{R}^3$, usando coordenadas cartesianas lo podemos denotar como $\vec{r} = (x, y, z) = x\hat{\imath} + y\hat{\jmath} + z\hat{k}$.
Existen operaciones algebraicas posibles con los vectores, estás son:
Sea el vector $\vec{A} = (A_x, A_y, A_z)$ y el vector $\vec{B} = (B_x, B_y, B_z)$:

- Suma de vectores $\vec{A}$ y $\vec{B}$:

$$\vec{A} + \vec{B} = (A_x + B_x)\hat{\imath} + (A_y + B_y)\hat{\jmath} + (A_z + B_z)\hat{k}$$

- Resta de vectores $\vec{A}$ y $\vec{B}$:

$$\vec{A} - \vec{B} = (A_x - B_x)\hat{\imath} + (A_y - B_y)\hat{\jmath} + (A_z - B_z)\hat{k}$$

- Mulplicación de un escalar k por un vector $\vec{A}$:

$$k\vec{A} = kA_x\hat{\imath} + kA_y\hat{\jmath} + kA_z\hat{k}$$

- Producto punto entre vectores $\vec{A}$ y $\vec{B}$:

$$\vec{A} \cdot \vec{B} = A_xB_x + A_yB_y + A_zB_z$$

- Producto cruz entre vectores $\vec{A}$ y $\vec{B}$:

$$\vec{A} \times \vec{B} = \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{vmatrix}$$

Sea Ω un espacio abierto en $\mathbb{R}^3$ Llamaremos campo vectorial sobre Ω a toda función

$$F : \Omega \subseteq \mathbb{R}^3 \to \mathbb{R}^3$$

Escribiendo en coordenadas cartesianas esto nos queda:

$$\vec{F}(x, y, z) = \vec{F}_1(x, y, z)\hat{\imath} + \vec{F}_2(x, y, z)\hat{\jmath} + \vec{F}_3(x, y, z)\hat{k}$$

Entonces podríamos definir el campo vectorial como un campo que asocia un vector a cada punto del espacio.

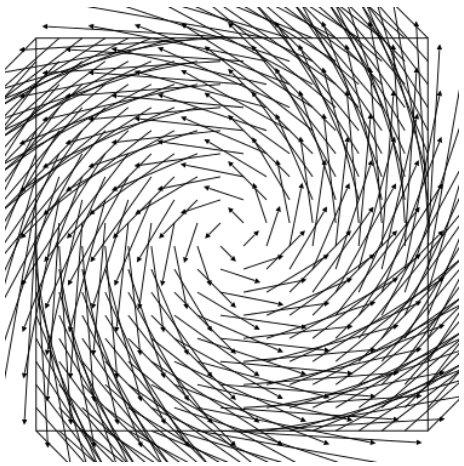


Figura A.2: Campo vectorial. (Fuente: [4])

Para graficar estos campos usualmente se escalan los vectores en cada punto de manera que al mirar el campo completo el dibujo sea entendible, es decir escalamos los vectores para que quepan todos en el dibujo. Otra buena manera de graficar estos campos de forma que sean más fácil de interpretar por parte del lector es coloreando las líneas y establecer una escala de colores para poder interpretar de mejor manera.

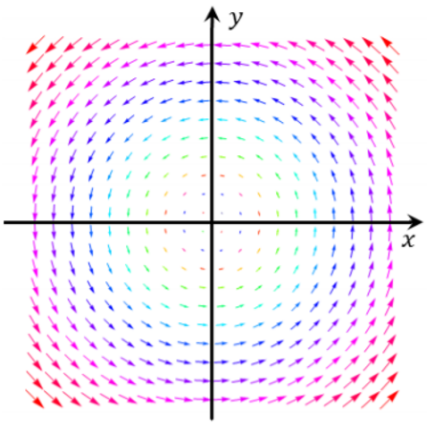


Figura A.3: Campo vectorial. (Fuente: [1])

Es necesario también saber las operaciones posibles entre campos vectoriales, las fundamentales son 2. El gradiente y el rotacional del campo. Según las definiciones de citecamposvectoriales2: Sea $f : A \subset \mathbb{R}^3 \to \mathbb{R}^3$ un campo vectorial. Suponiendo las condiciones necesarias de derivación definimos divergencia y rotor respectivamente como:

$$div\ F(x,y,z) = \frac{\partial F_1}{\partial x}(x,y,z) + \frac{\partial F_2}{\partial y}(x,y,z) + \frac{\partial F_3}{\partial z}(x,y,z)$$

$$rot\ F(x,y,z) = \left(\frac{\partial F_3}{\partial y} - \frac{\partial F_2}{\partial z}, \frac{\partial F_1}{\partial z} - \frac{\partial F_3}{\partial x}, \frac{\partial F_2}{\partial x} - \frac{\partial F_1}{\partial y}\right) = \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ F_1 & F_2 & F_3 \end{vmatrix}$$

Otro tipo de notación también es utilizada, ocupando el simbolo de gradiente como un operador:

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

Entonces, reescribimos la divergencia y el rotor utilizando el producto punto y cruz con nuestro operador de la forma:

$$\nabla \cdot F = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \cdot (F_1, F_2, F_3) = \frac{\partial F_1}{\partial x} + \frac{\partial F_2}{\partial y} + \frac{\partial F_3}{\partial z} = div\ F$$

$$\nabla \times F = \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ F_1 & F_2 & F_3 \end{vmatrix} = \left(\frac{\partial F_3}{\partial y} - \frac{\partial F_2}{\partial z}, \frac{\partial F_1}{\partial z} - \frac{\partial F_3}{\partial x}, \frac{\partial F_2}{\partial x} - \frac{\partial F_1}{\partial y}\right) = rot\ F$$

Este operador se puede aplicar sobre si mismo, este 'nuevo' operador se conoce como el 'operador laplaciano' y corresponde a la divergencia del gradiente, para representarlo se utiliza Δ o ∇^2:

$$\nabla \cdot (\nabla \cdot F) = \nabla^2 \cdot F = \frac{\partial^2 F_1}{\partial x^2} + \frac{\partial^2 F_2}{\partial y^2} + \frac{\partial^2 F_3}{\partial z^2}$$

Existen dos propiedades importantes que debemos tener en cuenta:

$$\nabla \times (\nabla \cdot F) = 0 \tag{A.1}$$

$$\nabla \cdot (fV) = f \cdot (\nabla V) - V \cdot \nabla f \tag{A.2}$$

Otro concepto de cálculo vectorial que debemos tener claro son los campos vectoriales conservativos los cuales son de vital importancia en el campo físico. Los campos vectoriales que pueden ser definidos bajo el gradiente de una función escalar f son llamados conservativos. Esto es $\vec{F} = \vec{\nabla} f$, además la función escalar f es conocida como la función potencial del campo $\vec{F}$.
En otras palabras, esto es, siendo el campo vectorial $\vec{F}(x,y,z) = \vec{F}_1(x,y,z)\hat{\imath} + \vec{F}_2(x,y,z)\hat{\jmath} + \vec{F}_3(x,y,z)\hat{k}$ la función potencial $f(x,y,z)$ del campo $\vec{F}$ está definida como:

$$\frac{\partial f}{\partial x}(x,y,z) = F_1(x,y,z), \qquad \frac{\partial f}{\partial y}(x,y,z) = F_2(x,y,z), \qquad \frac{\partial f}{\partial z}(x,y,z) = F_3(x,y,z)$$

A.2 Cálculo integral.

En BEM resolveremos ecuaciones diferenciales, en donde a las funciones diferenciales se les aplica muchas veces el concepto de integral. Si bien en este documento no nos adentraremos en demasía en el tema, los teoremas que se presentan a continuación serán de gran utilidad para resolver la problematica planteada:

- Teorema fundamental del cálculo: Sea f una función escalar.

$$\int_a^b \frac{df(x)}{dx} dx = f(b) - f(a) \tag{A.1}$$

- Teorema fundamental del gradiente: Sea F una función vectorial.

$$\int_a^b \nabla F ds = F(b) - F(a) \tag{A.2}$$

- Teorema de Gauss:

$$\int_\Omega (\nabla \cdot F) d\Omega = \oint_\Gamma n \cdot F d\Gamma \tag{A.3}$$

- Integración por partes:

$$\int udv = uv - \int vdu \tag{A.4}$$

A.3 Ecuaciónes Diferenciales.

En este documento investigaremos el comportamiento de dos ecuaciones de gran importancia en el modelamiento físico de varios fenómenos, por lo que es necesario tener un poco de entendimiento sobre su comportamiento matemático.

A.3.1 Ecuación de Laplace.

Esta es una ecuación de derivadas parciales de tipo elíptico. Es un caso particular de la ecuación de Helmholtz, sin embargo, la ecuación de Laplace junto con la ecuación de Poisson son los dos modelos más simples de las ecuaciones en derivadas parciales (EDP) de tipo elípticas.
La ecuación de Laplace en se puede escribir de distintas formas: Escrita con el operador nabla ∇

$$\nabla^2 \phi = 0 \tag{A.5}$$

Escrita en derivadas, para el caso tridimensional, en coordenadas cartesianas:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} = 0 \tag{A.6}$$

Esta ecuación es la encargada de modelar el comportamiento de, por ejemplo, el potencial eléctrico en una región sin cargas.

A.3.2 Ecuación de Helmholtz.

Esta ecuación, por otro lado, está compuesta por:

$$(\nabla^2 + k^2)\phi = 0 \tag{A.7}$$

Anexo B

Código Python Caso Verificación.

We are going to emulate the behavior of a magnetic field in microwires. Using the boundary element method and using Laplace as Green function, we're going to solve:

$$\begin{bmatrix} \frac{1}{2}I + K_L^\Gamma & -V_L^\Gamma \\ \frac{1}{2}I - K_L^\Gamma & \frac{\varepsilon_1}{\varepsilon_2}V_L^\Gamma \end{bmatrix} \begin{bmatrix} \varphi_{1d} \\ \frac{\partial}{\partial n}\varphi_{1d} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{\varepsilon_2-\varepsilon_1}{\varepsilon_2}\frac{\partial \varphi_{inc}}{\partial n} \end{bmatrix} \tag{B.1}$$

Being K the double layer operator, V the single layer operator and I the identity. And using the subindex 1 for the interior of the studied item and the subindex 2 for the exterior of it. We begin by calling all the libraries that we'll use.

```
#Import libraries
import bempp.api
import numpy as np
import timeit
import scipy as sp
import math
bempp.api.set_ipython_notebook_viewer()
```

Firstly, it is necessary to load the working meshes. An analysis of an incident EM wave will be made to a silver sphere immersed in water.

```
grid = bempp.api.import_grid("Sphere_1.msh")
grid2 = bempp.api.import_grid("Sphere_2.msh")
grid3 = bempp.api.import_grid("Sphere_3.msh")
grid4 = bempp.api.import_grid("Sphere_4.msh")
grid5 = bempp.api.import_grid("Sphere_5.5.msh")
mesh = (grid,grid2,grid3,grid4)
n_elem = []
for grid_elem in mesh:
number_of_elements = grid_elem.leaf_view.entity_count(0)
n_elem.append(number_of_elements)
```

The unit of the mesh is of the order of μm, so it is necessary to change the units if necessary

```
#Reference units
um = 1
mm = 1.e3
m = 1.e6
```

The electromagnetic properties of the materials present in the analysis, in addition to the wave properties are entered below:

```
#Data

#Frecuencia
omega = 1 / 3800

#Permitividad eléctrica
e0 = 1.7972083599999999+8.504766399999999e-09j #Permitividad Plata
e1 = -3.3876520488233184+0.19220746083441781j #Permitividad Agua

#Permeabilidad magnética
mu0 = 4. * np.pi * 1e-7 * 1.e6 * um  #micrometros

#Propiedades del material
mu = (-2.9214 + 0.5895j) * mu0
e = (82629.2677-200138.2211*1.j) * e0
cc = 6.5e4 * 1.e-18  #micrometros #Conductividad eléctrica

#Permitividad eléctrica relativa del material c/r al medio
er = e0 / e1
#Amplitud de la onda en micrometros
Amp = -1. * um #micrometros

#Longitud de onda
Diff_coef = np.sqrt(e0)
lamb = 3800
k = 2 * np.pi *Diff_coef/ lamb

#Antena
antena = np.array([[1.e5, 5.e5, 10.e5],[0., 0., 0.],[0., 0., 0.]])
```

The necessary spaces functions will be created to generate the operators:

```
#Set the Dirichlet and Neumann orders
order_neumann = 0
order_dirichlet = 0
def function_spaces(grid, order_neumann, order_dirichlet):
print("Setting the Dirichlet and Neumann spaces...")
n = grid.leaf_view.entity_count(0)
NS = bempp.api.function_space(grid, "DP", order_neumann)
DS = bempp.api.function_space(grid, "DP", order_dirichlet)
#Print the degrees of freedom
print("BEM dofs: {0}".format(NS.global_dof_count))
print("The grid has {0} elements.".format(n))
print("Spaces defined.")
print("-------------------------------------------------")
return NS, DS, n
```

Next, the single layer and double layer operator will be created, as well as an identity operator to generate the left side of the equation.

```
def operators(NS,DS):
print("Defining operators...")
slp = bempp.api.operators.boundary.laplace.single_layer(NS,DS,DS)
dlp = bempp.api.operators.boundary.laplace.double_layer(DS,DS,DS)
id = bempp.api.operators.boundary.sparse.identity(DS,DS,DS)
print("The operators are correctly defined.")
print("-------------------------------------------------")
return slp, dlp, id
```

The left side of the equation is generated, which has the following form:

$$\begin{bmatrix} \frac{1}{2}I + K & -V \\ \frac{1}{2}I - K & \frac{\varepsilon_1}{\varepsilon_2}V \end{bmatrix} \tag{B.2}$$

```
def blocked_operator(slp, dlp, id):
print("Defininig the blocked operator...")
blocks = [[None,None],[None,None]]
blocks[0][0] = (0.5 * id + dlp).weak_form()
blocks[0][1] = - slp.weak_form()
blocks[1][0] = (0.5 * id - dlp).weak_form()
blocks[1][1] = (er * slp).weak_form()
blocked = bempp.api.BlockedDiscreteOperator(np.array(blocks))
print("The blocked operator is correctly defined.")
print("-------------------------------------------------")
return blocked
```

The right hand side of the equation has the following composition:

$$\begin{bmatrix} 0 \\ \frac{\varepsilon_2 - \varepsilon_1}{\varepsilon_2} V_L^{\Gamma} \frac{\partial \varphi_{inc}}{\partial n} \end{bmatrix} \tag{B.3}$$

```
#Definition of functions
def rhs_equation(DS, slp, n):
print("Defining the right hand side of the equation...")
def funcion1(x, n, domain_index, result):
result[:] = ((1 - er) / 1) * (Amp * n[0])
funcion_fun = bempp.api.GridFunction(DS, fun=funcion1)
funcion_fun = slp * funcion_fun
rhs_fem = np.zeros(n)
rhs_bem = funcion_fun.projections(DS)
rhs = np.concatenate([rhs_fem, rhs_bem])
print("The right hand side of the equation is ready to be used.")
print("-------------------------------------------------")
return rhs
```

An iteration counter is generated

```
class gmres_counter(object):
def __init__(self, disp=True):
self._disp = disp
self.niter = 0
def __call__(self, rk=None):
self.niter += 1
if self.niter %10 == 0:
if self._disp:
print('iteration %3i\trest = %s' % (self.niter, str(rk)))
```

The equation is solved through gmres and the number of iterations is printed every 10 iterations.

```
def solving(blocked,rhs):
print("Solving the equation...")
import inspect
from scipy.sparse.linalg import gmres
counter = gmres_counter()
print("Shape of matrix: {0}".format(blocked.shape))
aa = blocked.shape
a = aa[0]/2
soln,info = gmres(blocked, rhs, tol=1e-6, callback = counter,
        maxiter = 5000, restart = 5000)
print("El sistema fue resuelto en {0} iteraciones".format(counter.niter))
it = counter.niter
np.savetxt("Solucion.out", soln, delimiter=",")
print("Equation Solve.")
print("-------------------------------------------------")
return soln, a, it
```

It is necessary to divide the solution obtained. A matrix is obtained whose structure is:

$$\begin{bmatrix} \varphi_{1d} \\ \frac{\partial}{\partial n}\varphi_{1d} \end{bmatrix}$$

So, in this case, you should only divide the solution by 2.

```
def divide(soln, DS, NS, n):
print("Is necessary to separate the solution...")
solution_dirichl=soln[-n:]
solution_neumann=soln[n:]
c1 = solution_dirichl  # These are the coefficients
solution_dirichl_s = bempp.api.GridFunction(DS, coefficients=c1)
c2 = solution_neumann  # These are the coefficients
solution_neumann_s = bempp.api.GridFunction(DS, coefficients=c2)
print("Solution divided.")
print("-------------------------------------------------")
return(solution_dirichl, solution_neumann,
                solution_dirichl_s, solution_neumann_s)
```

Now, the extinguished cross section will be calculated, for that we must make calculations in each element of the mesh. From the mesh file, we will calculate the center and normal of each element and record them in a list.

```
def calc_centers(grid):
print("Calculate element's center...")
elements = list(grid.leaf_view.entity_iterator(0))
n = (len(elements))
area = []
xc = []
yc = []
zc = []
for i in range (n):
area1 = elements[i].geometry.volume
area.append(area1/10**3)
corners = elements[i].geometry.corners

p1 = corners[0]
p2 = corners[1]
p3 = corners[2]

xi,yi,zi = ((p1 + p2 + p3) / 3)
xc.append(xi)
yc.append(yi)
zc.append(zi)
center = np.array([xc,yc,zc])
return center, area
```

```
def calc_normal(grid,center):
print("Calculate normals...")
elements = list(grid.leaf_view.entity_iterator(0))
n = (len(elements))
normal = []
nx = []
ny = []
nz = []
for i in range (n):
corners = elements[i].geometry.corners
p2 = corners[1]
p3 = corners[2]

v1 = p2 - center[:,i]
v2 = p3 - center[:,i]

normalv = np.cross(v2, v1)
a, b, c = normalv
norm_normalv = np.linalg.norm(normalv)
ax, ay, az = normalv/norm_normalv
nx.append(ax)
ny.append(ay)
nz.append(az)
normal = np.array([nx,ny,nz])
return normal
```

It is necessary to have the value of the electric dipole (p_i) in each element, whose formula is:

$$p_i = \epsilon_2 \sum_{i=0}^{n} \left[r_i \frac{\partial}{\partial n_j} \phi_{2s} d\Gamma - n_i \phi d\Gamma \right]$$

```
def calc_dipole(n, phi, dphi, area, center, normal):
print("Calculate dipole...")
dipole_x = []
dipole_y = []
dipole_z = []
for i in range(n):
dphi2 = dphi * er - (1 - er) * Amp * np.transpose(normal)[:,2]
I1 = np.sum(center * dphi2 * area, axis=1)
I2 = np.sum(normal * phi * area, axis=1)
dx,dy,dz = e0 * (I1-I2)
dipole_x.append(dx)
dipole_y.append(dy)
dipole_z.append(dz)
dipole =  np.array([dipole_x,dipole_y,dipole_z])
return dipole
```

The extinct cross section is calculated as follows:

$$E_{scattered} = \frac{1}{4\pi\epsilon_2} k^2 \frac{e^{ikr}}{r} (\hat{r} \times p) \times \hat{r}$$

$$E_{scattered}(r)_{r\to\infty} = \frac{e^{ikr}}{r} F(k, k_0)$$

$$C_{ext} = \frac{4\pi}{k} Im \left[\frac{\hat{e}_i}{|E_i|} F(k = k_0, k_0) \right]$$

```
def cross_extintion_section(grid, dipole):
elements = list(grid.leaf_view.entity_iterator(0))
n = (len(elements))
Cext = []
surf_Cext = []
r1 = (1,0,0)
r2 = (0,0,1)
n2 = (1,0,0)
cx = []
cy = []
cz = []
for i in range(n):
v1 = np.cross(r1, dipole[:,i])
v2 = np.cross(v1, r2)
C1 = np.dot(n2, v2) * k**2 / (e0 * Amp)
Cext.append(1 / k.real * C1.imag * 0.01 * 10**6)
surf_Cext.append(i)
print("Cross Extintion Section:")
Cext_conv = Cext[0]
print(Cext_conv)
print("_______________________________")
return Cext_conv
```

Additionally, we will add the measurement of the electric field in antennas located in 3 different positions.

```
def calculo_antena(DS,antena,solution_dirichl_s,solution_neumann_s):
slp_pot_ext = bempp.api.operators.potential.laplace.single_layer(DS, antena)
dlp_pot_ext = bempp.api.operators.potential.laplace.double_layer(DS, antena)
Campo_en_antena_dis = (dlp_pot_ext * solution_dirichl_s
                - slp_pot_ext * solution_neumann_s).ravel()
print("Calculate Electric potential in the antena.")
print("The electric potential measured by the antenna
                        is: ".format(Campo_en_antena_dis))
print(Campo_en_antena_dis)
print("-------------------------------------------------")
return Campo_en_antena_dis
```

All subroutines were defined as a function, another subroutine is created that calls all previously created functions. The values. The values of the electric field in the antennas and the extinction cross section are the parameters that return from the global subroutine.

```
def BEM(grid, order_neumann, order_dirichlet, antena):
NS,DS, n = function_spaces(grid, order_neumann, order_dirichlet)
slp, dlp, id = operators(NS,DS)
blocked = blocked_operator(slp, dlp, id)
rhs = rhs_equation(DS, slp, n)
soln, a, it = solving(blocked,rhs)
phi, dphi, phi_s, dphi_s = divide(soln, DS, NS, a)
ant_1 = calculo_antena(DS, antena, phi_s, dphi_s)
center, area = calc_centers(grid)
normal = calc_normal(grid,center)
dipole = calc_dipole(n, phi, dphi, area, center, normal)
Cext = cross_extintion_section(grid, dipole)
return Cext, phi, dphi, phi_s, dphi_s, DS, NS, ant_1, it
```

```
C_ext_an = []
antena_1 = []
antena_2 = []
antena_3 = []
iteraciones = []
for grid_item in mesh:
print('Calculating for ')
Cext, phi, dphi, phi_s, dphi_s, DS, NS, ant_med, it = BEM(grid_item,
                          order_neumann, order_dirichlet, antena)
C_ext_an.append(Cext)
ant_1, ant_2, ant_3 = ant_med
antena_1.append(ant_1)
antena_2.append(ant_2)
antena_3.append(ant_3)
iteraciones.append(it)
```

```
Calculating for
Setting the Dirichlet and Neumann spaces...
BEM dofs: 80
The grid has 80 elements.
Spaces defined.
-------------------------------------------------
Defining operators...
The operators are correctly defined.
-------------------------------------------------
Defininig the blocked operator...
The blocked operator is correctly defined.
-------------------------------------------------
Defining the right hand side of the equation...
```

```
The right hand side of the equation is ready to be used.
------------------------------------------------
Solving the equation...
Shape of matrix: (160, 160)
El sistema fue resuelto en 7 iteraciones
Equation Solve.
------------------------------------------------
Is necessary to separate the solution...
Solution divided.
------------------------------------------------
Calculate Scattered Electric Field in the antena.
The scattered electric field measured by the antenna is:
[  3.31090654e-05 +1.40393236e-06j   6.67444616e-06 +2.83043363e-07j
   3.34050764e-06 +1.41662522e-07j]
------------------------------------------------
Calculate element's center...
Calculate normals...
Calculate dipole...
Cross Extintion Section:
735.811621419
----------------------------
Calculating for
Setting the Dirichlet and Neumann spaces...
BEM dofs: 320
The grid has 320 elements.
Spaces defined.
------------------------------------------------
Defining operators...
The operators are correctly defined.
------------------------------------------------
Defininig the blocked operator...


The blocked operator is correctly defined.
------------------------------------------------
Defining the right hand side of the equation...
The right hand side of the equation is ready to be used.
------------------------------------------------
Solving the equation...
Shape of matrix: (640, 640)
iteration  10        rest = 0.009054365131443774
iteration  20        rest = 0.00037510837063818255
iteration  30        rest = 3.3374018677815633e-06
El sistema fue resuelto en 32 iteraciones
Equation Solve.
------------------------------------------------
Is necessary to separate the solution...
Solution divided.
```

```
------------------------------------------------
Calculate Scattered Electric Field in the antena.
The scattered electric field measured by the antenna is:
[  3.39235583e-05 +1.37598270e-06j   6.83550124e-06 +2.77264784e-07j
   3.42091924e-06 +1.38761424e-07j]
------------------------------------------------
Calculate element's center...
Calculate normals...
Calculate dipole...
Cross Extintion Section:
1149.29292933
---------------------------
Calculating for
Setting the Dirichlet and Neumann spaces...
BEM dofs: 1280
The grid has 1280 elements.
Spaces defined.
------------------------------------------------
Defining operators...
The operators are correctly defined.
------------------------------------------------
Defininig the blocked operator...
The blocked operator is correctly defined.
------------------------------------------------
Defining the right hand side of the equation...
The right hand side of the equation is ready to be used.
------------------------------------------------
Solving the equation...
Shape of matrix: (2560, 2560)
iteration  10        rest = 0.026675348520517637
iteration  20        rest = 0.011471051306879499
iteration  30        rest = 0.0027375914509101744
iteration  40        rest = 0.0007221831798288935
iteration  50        rest = 0.00018039348903727334
iteration  60        rest = 4.7111662340689676e-05
iteration  70        rest = 9.571118435396603e-06
iteration  80        rest = 1.7407364079251463e-06
El sistema fue resuelto en 82 iteraciones
Equation Solve.
------------------------------------------------
Is necessary to separate the solution...
Solution divided.
------------------------------------------------
Calculate Scattered Electric Field in the antena.
The scattered electric field measured by the antenna is:
[  3.39766191e-05 +1.36451792e-06j   6.84570926e-06 +2.74928966e-07j
   3.42599795e-06 +1.37590838e-07j]
------------------------------------------------
```

```
Calculate element's center...
Calculate normals...
Calculate dipole...
Cross Extintion Section:
1390.35524865
----------------------------
Calculating for
Setting the Dirichlet and Neumann spaces...
BEM dofs: 5120
The grid has 5120 elements.
Spaces defined.
-----------------------------------------------
Defining operators...
The operators are correctly defined.
-----------------------------------------------
Defininig the blocked operator...
The blocked operator is correctly defined.
-----------------------------------------------
Defining the right hand side of the equation...
The right hand side of the equation is ready to be used.
-----------------------------------------------
Solving the equation...
Shape of matrix: (10240, 10240)
iteration  10        rest = 0.03547349498683443
iteration  20        rest = 0.01816202363562909
iteration  30        rest = 0.012101892539027843
iteration  40        rest = 0.00954287552304474
iteration  50        rest = 0.005701455503897905
iteration  60        rest = 0.0028402273266719407
iteration  70        rest = 0.0018376208829977553
iteration  80        rest = 0.0008956099281580536
iteration  90        rest = 0.000511915164246382
iteration 100        rest = 0.00021546615720539346
iteration 110        rest = 0.00012854770193471658
iteration 120        rest = 6.324056834493666e-05
iteration 130        rest = 3.166237411465301e-05
iteration 140        rest = 1.579530315945104e-05
iteration 150        rest = 8.585407628780112e-06
iteration 160        rest = 3.647110932803966e-06
iteration 170        rest = 1.7894118966434337e-06
El sistema fue resuelto en 179 iteraciones
Equation Solve.
-----------------------------------------------
Is necessary to separate the solution...
Solution divided.
-----------------------------------------------
Calculate Scattered Electric Field in the antena.
The scattered electric field measured by the antenna is:
```

```
[  3.48409367e-05 +1.39595055e-06j   7.01835690e-06 +2.81200737e-07j
   3.51230871e-06 +1.40725819e-07j]
-------------------------------------------------
Calculate element's center...
Calculate normals...
Calculate dipole...
Cross Extintion Section:
1514.01958459
-----------------------------
```

Mishchenko derived the following analytical result, valid for lossy mediums:

$$C_{ext} = \frac{4\pi a^3}{k} Im\left[k^2 \frac{\varepsilon_{in}/\varepsilon_{out} - 1}{\varepsilon_{in}/\varepsilon_{out} + 2}\right]$$

Where a is the sphere's radio, k is the wave number, ε_{in} is the Sphere's dielectric constant, ε_{out} is the host medium's dielectric constant. The result is 1854.48 $[nm^2]$

```
C_ext_teo = 1854.48
C_ext_teo_list = [C_ext_teo, C_ext_teo, C_ext_teo, C_ext_teo]
```

The following are graphs of iterations, mesh convergence, relative error of the extinction cross section, the measurements of the electric field in the antennas and their relative variation with respect to the first measurement.

```
import matplotlib.pyplot as plt
import numpy as np
plt.show()
plt.ioff()
plt.title("Iterations to solve the system")
plt.plot(n_elem ,iteraciones, marker='o', linestyle='--', color='r')
plt.ion()
plt.grid(True)
#plt.yscale("log")
plt.xlabel("Number of elements")
plt.ylabel("Cross Extintion Section")
plt.legend()
plt.legend(loc="lower right")
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np
plt.show()
plt.ioff()
plt.title("Mesh Convergence")
plt.plot(n_elem ,C_ext_an, marker='o', linestyle='--', color='r',
                        label = "Analytical (Calculated)")
plt.plot( n_elem,C_ext_teo_list, marker='s', linestyle=':', color='b',
                        label = "Analytical (Mischenko)")
plt.ion()
plt.grid(True)
#plt.yscale("log")
plt.xlabel("Number of elements")
plt.ylabel("Cross Extintion Section")
plt.legend()
plt.legend(loc="lower right")
plt.show()
```

```
Error = np.zeros(len(C_ext_an))
for i in range(len(C_ext_an)):
Error[i] = abs((C_ext_an[i] - C_ext_teo_list[i]) / C_ext_teo_list[i])*100
import matplotlib.pyplot as plt
import numpy as np
plt.ioff()

plt.title("Relative Error Cross Extintion Section")
plt.plot(n_elem , Error, marker='o', linestyle='--', color='r')
plt.ion()
plt.grid(True)
plt.xlabel("Number of elements")
plt.ylabel("Relative Error [%]")
plt.legend()
plt.legend(loc="right")
plt.show()
print(Error)
```

```
[ 60.32248278  38.02613513  25.02721795  18.35880761]
```

```
antena_1 = [i * 10**6 for i in antena_1]
antena_2 = [i * 10**6 for i in antena_2]
antena_3 = [i * 10**6 for i in antena_3]
antena_1_real = np.zeros(len(antena_1))
antena_1_imag = np.zeros(len(antena_1))
antena_2_real = np.zeros(len(antena_1))
antena_2_imag = np.zeros(len(antena_1))
antena_3_real = np.zeros(len(antena_1))
antena_3_imag = np.zeros(len(antena_1))
for i in range(len(antena_1)):
antena_1_real[i] = antena_1[i].real
antena_1_imag[i] = antena_1[i].imag
antena_2_real[i] = antena_2[i].real
antena_2_imag[i] = antena_2[i].imag
antena_3_real[i] = antena_3[i].real
antena_3_imag[i] = antena_3[i].imag

import matplotlib.pyplot as plt
import numpy as np
plt.ioff()
plt.title("Real Part of the Calculated Electric
                        potential in the Antena")
plt.plot(n_elem , antena_1_real, marker='o', linestyle='--', color='r',
                        label = "Antena 1")
plt.plot(n_elem , antena_2_real, marker='o', linestyle='--', color='b',
                        label = "Antena 2")
plt.plot(n_elem , antena_3_real, marker='o', linestyle='--', color='g',
                        label = "Antena 3")
plt.ion()
plt.grid(True)
plt.ylim(0,40)
plt.xlabel("Number of elements")
plt.ylabel("Electric field [$\mu$V]")
plt.legend()
plt.legend(loc="right")
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np
plt.ioff()
plt.title("Imaginary Part of the Calculated Electric
                        potential in the Antena")
plt.plot(n_elem , antena_1_imag, marker='o', linestyle='--', color='r',
                        label = "Antena 1")
plt.plot(n_elem , antena_2_imag, marker='o', linestyle='--', color='b',
                        label = "Antena 2")
plt.plot(n_elem , antena_3_imag, marker='o', linestyle='--', color='g',
                        label = "Antena 3")
plt.ion()
plt.grid(True)
plt.xlabel("Number of elements")
plt.ylabel("Electric potential [V]")
plt.legend()
plt.legend(loc="right")
plt.show()
```

```
Var_1_real = np.zeros(len(antena_1))
Var_2_real = np.zeros(len(antena_1))
Var_3_real = np.zeros(len(antena_1))
for i in range(len(antena_1)):
Var_1_real[i] = abs((antena_1_real[i] - antena_1_real[0])
                        / antena_1_real[0])*100
Var_2_real[i] = abs((antena_2_real[i] - antena_2_real[0])
                        / antena_2_real[0])*100
Var_3_real[i] = abs((antena_3_real[i] - antena_3_real[0])
                        / antena_3_real[0])*100
import matplotlib.pyplot as plt
import numpy as np
plt.ioff()
plt.title("Relative Variation of the real part of the Electric
                        potential in the Antena",y=1.05)
plt.plot(n_elem , Var_1_real, marker='o', linestyle='--', color='r',
                        label = 'Antena 1')
plt.plot(n_elem , Var_2_real, marker='s', linestyle='--', color='b',
                        label = 'Antena 2')
plt.plot(n_elem , Var_3_real, marker='x', linestyle='--', color='g',
                        label = 'Antena 3')
plt.ion()
plt.grid(True)
plt.xlabel("Number of elements")
plt.ylabel("Relative Variation [%]")
plt.legend()
plt.legend(loc="right")
plt.show()
```

```
In [26]: Var_1_imag = np.zeros(len(antena_1))
Var_2_imag = np.zeros(len(antena_1))
Var_3_imag = np.zeros(len(antena_1))
for i in range(len(antena_1)):
Var_1_imag[i] = abs((antena_1_imag[i] - antena_1_imag[0])
                        / antena_1_imag[0])*100
Var_2_imag[i] = abs((antena_2_imag[i] - antena_2_imag[0])
                        / antena_2_imag[0])*100
Var_3_imag[i] = abs((antena_3_imag[i] - antena_3_imag[0])
                        / antena_3_imag[0])*100
import matplotlib.pyplot as plt
import numpy as np
plt.ioff()
plt.title("Relative Variation of the imaginary part of the Electric
                        potential in the Antena",y=1.05)
plt.plot(n_elem , Var_1_imag, marker='o', linestyle='--', color='r',
                        label = 'Antena 1')
plt.plot(n_elem , Var_2_imag, marker='s', linestyle='--', color='b',
                        label = 'Antena 2')
plt.plot(n_elem , Var_3_imag, marker='x', linestyle='--', color='g',
                        label = 'Antena 3')
plt.ion()
plt.grid(True)
plt.xlabel("Number of elements")
plt.ylabel("Relative Variation [%]")
plt.legend()
plt.legend(loc="upper right")
plt.show()
```

Anexo C

Código Python Caso 2 superficies.

In the following lines the following equation will be solved:

$$\begin{bmatrix} \frac{1}{2}I + K_{1-1} & -V_{1-1} & 0 & 0 \\ \frac{1}{2}I - K_{1-1} & \frac{\varepsilon_1}{\varepsilon_2}V_{1-1} & K_{1-2} & -V_{1-2} \\ -K_{2-1} & \frac{\varepsilon_1}{\varepsilon_2}V_{2-1} & \frac{1}{2}I + K_{2-2} & -V_{2-2} \\ 0 & 0 & \frac{1}{2}I - K_{2-2} & \frac{\varepsilon_2}{\varepsilon_3}V_{2-2} \end{bmatrix} \begin{bmatrix} \varphi_{1d} \\ \frac{\partial}{\partial n}\varphi_{1d} \\ \varphi_{2d} \\ \frac{\partial}{\partial n}\varphi_{2d} \end{bmatrix} = \begin{bmatrix} \frac{\varepsilon_2-\varepsilon_1}{\varepsilon_2}\frac{\partial \varphi_{i1}}{\partial n} + \frac{\varepsilon_2-\varepsilon_3}{\varepsilon_2}\frac{\partial \varphi_{i2}}{\partial n} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{C.1}$$

and the electric field will be calculated in antennas located in 3 different positions.

```
#Import libraries
import bempp.api
import numpy as np
import timeit
import scipy as sp
import math
bempp.api.set_ipython_notebook_viewer()
```

```
#Define grid
grid = bempp.api.import_grid("Cylinder_1.msh")
matrix = bempp.api.import_grid("Matrix.msh")
number_of_elements = grid.leaf_view.entity_count(0)
number_of_elements2 = matrix.leaf_view.entity_count(0)
print("The cylinder grid has {0} elements.".format(number_of_elements))
print("The matrix grid has {0} elements.".format(number_of_elements2))
grid.plot()
matrix.plot()
```

```
The cylinder grid has 1056 elements.
The matrix grid has 992 elements.
```

```
#Data

#Unidades de referencia
um = 1
mm = 1.e3
m = 1.e6

#Frecuencia
omega = 1 / 3800

#Permeabilidad magnética
mu0 = 4. * np.pi * 1e-7 * 1.e6 * um  #micrometros

#Propiedades del material
e0 = 8.854 * 1e-12*1e-18
e1 = 16*e0
mu = (-2.9214 + 0.5895j) * mu0
e2 = (82629.2677-200138.2211*1.j) * e0
cc = 6.5e4 * 1.e-18  #micrometros #Conductividad eléctrica

ep2 = [1.1*e2,e2,0.9*e2]
#Permitividad eléctrica relativa del material c/r al medio
er = e0 / e1
#Amplitud de la onda en micrometros
Amp = -1. * um #micrometros

#Longitud de onda
Diff_coef = np.sqrt(e0)
lamb = 3800
k = 2 * np.pi *Diff_coef/ lamb

#Antena
antena = np.array([[1.e5, 5.e5, 10.e5],[0., 0., 0.],[0., 0., 0.]])
```

```
#Set the Dirichlet and Neumann orders
def function_spaces(grid, grid2):
order_neumann = 0
order_dirichlet = 0
print("Setting the Dirichlet and Neumann spaces...")
n = grid.leaf_view.entity_count(0)
n2 = grid2.leaf_view.entity_count(0)
NS = bempp.api.function_space(grid, "DP", order_neumann)
DS = bempp.api.function_space(grid, "DP", order_dirichlet)
NS2 = bempp.api.function_space(grid2, "DP", order_neumann)
DS2 = bempp.api.function_space(grid2, "DP", order_dirichlet)
#Print the degrees of freedom
print("For the Grid 1:")
print("BEM dofs: {0}".format(NS.global_dof_count))
print("The grid has {0} elements.".format(n))
print("Spaces defined.")
print("-------------------------------------------------")
print("For the Grid 2:")
print("BEM dofs: {0}".format(NS2.global_dof_count))
print("The grid has {0} elements.".format(n2))
print("Spaces defined.")
print("-------------------------------------------------")
return NS, DS, NS2, DS2, n, n2
```

```
def stern_formulation(DS, NS, DS2, NS2, ep1, ep2, ep3):
# Functions to proyect the carges potential to the boundary with constants
def green_func(x, n, domain_index, result):
result[:] = ((ep1 - ep2) / ep1) * (Amp * n[0]) +
                        ((ep2 - ep3) / ep1) * (Amp * n[0])
print("\nProjecting charges over surface...")
charged_grid_fun  = bempp.api.GridFunction(DS, fun=green_func)
rhs = np.concatenate([charged_grid_fun.coefficients,
#rhs = np.concatenate([charged_grid_fun,
np.zeros(NS.global_dof_count),
np.zeros(DS2.global_dof_count),
np.zeros(NS2.global_dof_count)])
print("Defining operators...")
# OPERATOR FOR INTERNAL SURFACE
from bempp.api.operators.boundary import sparse, laplace, modified_helmholtz
idn_in  = sparse.identity(DS, DS, DS)
# Ec 1
slp_in = laplace.single_layer(NS, DS, DS)
dlp_in = laplace.double_layer(DS, DS, DS)
# Ec 2
# adj_1T1 = laplace.single_layer(NS, DS, DS)
# adj_1T1 = laplace.double_layer(DS, DS, DS)
slp_2T1 = laplace.single_layer(NS2, DS, DS)
dlp_2T1 = laplace.double_layer(DS2, DS, DS)
# OPERATOR FOR EXTERNAL SURFACE
idn_ex = sparse.identity(DS2, DS2, DS2)
# Internal Boudary
slp_1T2 = laplace.single_layer(NS, DS2, DS2)
dlp_1T2 = laplace.double_layer(DS, DS2, DS2)
slp_2T2 = laplace.single_layer(NS2, DS2, DS2)
dlp_2T2 = laplace.double_layer(DS2, DS2, DS2)
print("Creating operators...")
# Matrix Assemble
blocked = bempp.api.BlockedOperator(4, 4)
blocked[0, 0] = .5*idn_in + dlp_in
blocked[0, 1] = -slp_in
# blocked[0, 2] = 0
# blocked[0, 3] = 0
# Original formulation
blocked[1, 0] = .5*idn_in - dlp_in
blocked[1, 1] =  ep1/ep2 *slp_in
blocked[1, 2] =  dlp_2T1
blocked[1, 3] = -slp_2T1
```

```
# blocked[2, 0] = -dlp_1T2
blocked[2, 1] =  ep1/ep2*slp_1T2
blocked[2, 2] = .5*idn_ex + dlp_2T2
blocked[2, 3] = -slp_2T2
# blocked[3, 0] = 0
# blocked[3, 1] = 0
blocked[3, 2] = .5*idn_ex - dlp_2T2
blocked[3, 3] = ep2/ep3 * slp_2T2
A = blocked.strong_form()
#A = blocked
return A, rhs
```

```
class gmres_counter(object):
def __init__(self, disp=True):
self._disp = disp
self.niter = 0
def __call__(self, rk=None):
self.niter += 1
if self.niter %250 == 0:
if self._disp:
print('iteration %3i\trest = %s' % (self.niter, str(rk)))
```

```
def solving(blocked,rhs):
print("Solving the equation...")
#Sistema de ecuaciones
import inspect
from scipy.sparse.linalg import gmres
counter = gmres_counter()
print("Shape of matrix: {0}".format(blocked.shape))
aa = blocked.shape
a = aa[0]/2
soln,info = gmres(blocked, rhs, tol=1e-6, callback = counter,
                        maxiter = 5000, restart = 5000)
print("El sistema fue resuelto en {0} iteraciones".format(counter.niter))
np.savetxt("Solucion.out", soln, delimiter=",")
print("Equation Solve.")
print("-------------------------------------------------")
return soln, a
```

```
#Divide the solution
def divide(soln, DS, NS, DS2, NS2, n, n2):
print("Is necessary to separate the solution...")
phi_1=soln[0:n]
dphi_1=soln[n:2*n]
phi_2=soln[2*n:2*n+n2]
dphi_2=soln[2*n+n2:-1]
phi_1s = bempp.api.GridFunction(DS, coefficients=phi_1)
dphi_1s = bempp.api.GridFunction(DS, coefficients=dphi_1)
phi_2s = bempp.api.GridFunction(DS2, coefficients=phi_2)
dphi_2s = bempp.api.GridFunction(DS2, coefficients=dphi_2)
print("Solution divided.")
print("-------------------------------------------------")
return(phi_1s, dphi_1s, phi_2s, dphi_2s)
```

```
def calculo_antena(DS,antena,solution_dirichl_s,solution_neumann_s):
#Calculo campo en antena
slp_pot_ext = bempp.api.operators.potential.laplace.single_layer(DS, antena)
dlp_pot_ext = bempp.api.operators.potential.laplace.double_layer(DS, antena)
Campo_en_antena_dis = (dlp_pot_ext * solution_dirichl_s
                                - slp_pot_ext * solution_neumann_s).ravel()
print("Calculate Scattered Electric Field in the antena.")
print("The scattered electric field measured by the antenna
                                is: ".format(Campo_en_antena_dis))
print(Campo_en_antena_dis)
print("_____________________________________________________________")
return Campo_en_antena_dis
```

```
def BEM(grid, matrix,e2,e1,e0):
NS, DS, NS2, DS2, n, n2 = function_spaces(grid, matrix)
A, rhs = stern_formulation(DS, NS, DS2, NS2, e2, e1, e0)
soln, a  = solving(A, rhs)
phi, dphi, phi_s, dphi_s = divide(soln, DS, NS, DS2, NS2, n, n2)
calc_antena = calculo_antena(DS, antena, phi, dphi)
return calc_antena
```

```
antena_1 = []
antena_2 = []
antena_3 = []
n_elem = np.zeros(3)
print(n_elem)
for i in range(3):
print("Case Number:")
print(i+1)
n_elem[i] = i+1
e2 = ep2[i]
ant_med = BEM(grid, matrix,e2,e1,e0)
ant_1, ant_2, ant_3 = ant_med
antena_1.append(ant_1)
antena_2.append(ant_2)
antena_3.append(ant_3)
antena_1 = [i * 10**6 for i in antena_1]
antena_2 = [i * 10**6 for i in antena_2]
antena_3 = [i * 10**6 for i in antena_3]
```

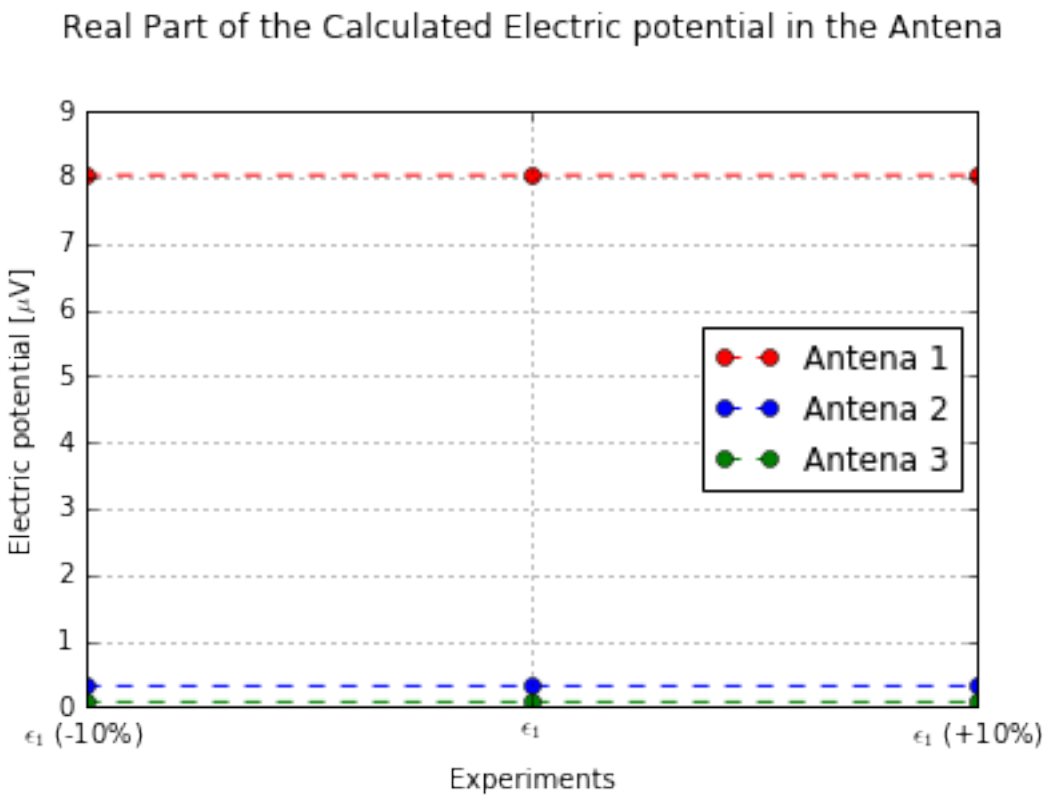
```
Case Number:
1
Setting the Dirichlet and Neumann spaces...
For the Grid 1:
BEM dofs: 1056
The grid has 1056 elements.
Spaces defined.
-------------------------------------------------
For the Grid 2:
BEM dofs: 992
The grid has 992 elements.
Spaces defined.
-------------------------------------------------

Projecting charges over surface...
Defining operators...
Creating operators...
Solving the equation...
Shape of matrix: (4096, 4096)
iteration 250        rest = 0.10144210328240723
iteration 500        rest = 0.02152028400457728
iteration 750        rest = 0.012588874326961336
iteration 1000        rest = 0.005048124995442141
iteration 1250        rest = 0.002421003173334851
iteration 1500        rest = 0.000856020644947324
iteration 1750        rest = 0.0004732394467689471
El sistema fue resuelto en 1821 iteraciones
Equation Solve.
```

```
------------------------------------------------
Is necessary to separate the solution...
Solution divided.
------------------------------------------------
Calculate Scattered Electric Field in the antena.
The scattered electric field measured by the antenna is:
[ -8.03275894e-06 -7.29935078e-09j  -3.22224361e-07 -1.57217314e-09j
  -8.07320200e-08 -7.93115034e-10j]
------------------------------------------------------
Case Number:
2
Setting the Dirichlet and Neumann spaces...
For the Grid 1:
BEM dofs: 1056
The grid has 1056 elements.
Spaces defined.
------------------------------------------------
For the Grid 2:
BEM dofs: 992
The grid has 992 elements.
Spaces defined.
------------------------------------------------

Projecting charges over surface...
Defining operators...
Creating operators...
Solving the equation...
Shape of matrix: (4096, 4096)
iteration 250        rest = 0.10030778901369052
iteration 500        rest = 0.021356902412355553
iteration 750        rest = 0.012448160312827423
iteration 1000        rest = 0.005042583483807689
iteration 1250        rest = 0.002420219048902603
iteration 1500        rest = 0.0008555883783780839
iteration 1750        rest = 0.00047323693380672333
El sistema fue resuelto en 1821 iteraciones
Equation Solve.
------------------------------------------------
Is necessary to separate the solution...
Solution divided.
------------------------------------------------
Calculate Scattered Electric Field in the antena.
The scattered electric field measured by the antenna is:
[ -8.03305747e-06 -8.03023342e-09j  -3.22288282e-07 -1.72972209e-09j
  -8.07642442e-08 -8.72601250e-10j]
------------------------------------------------------
Case Number:
3
```

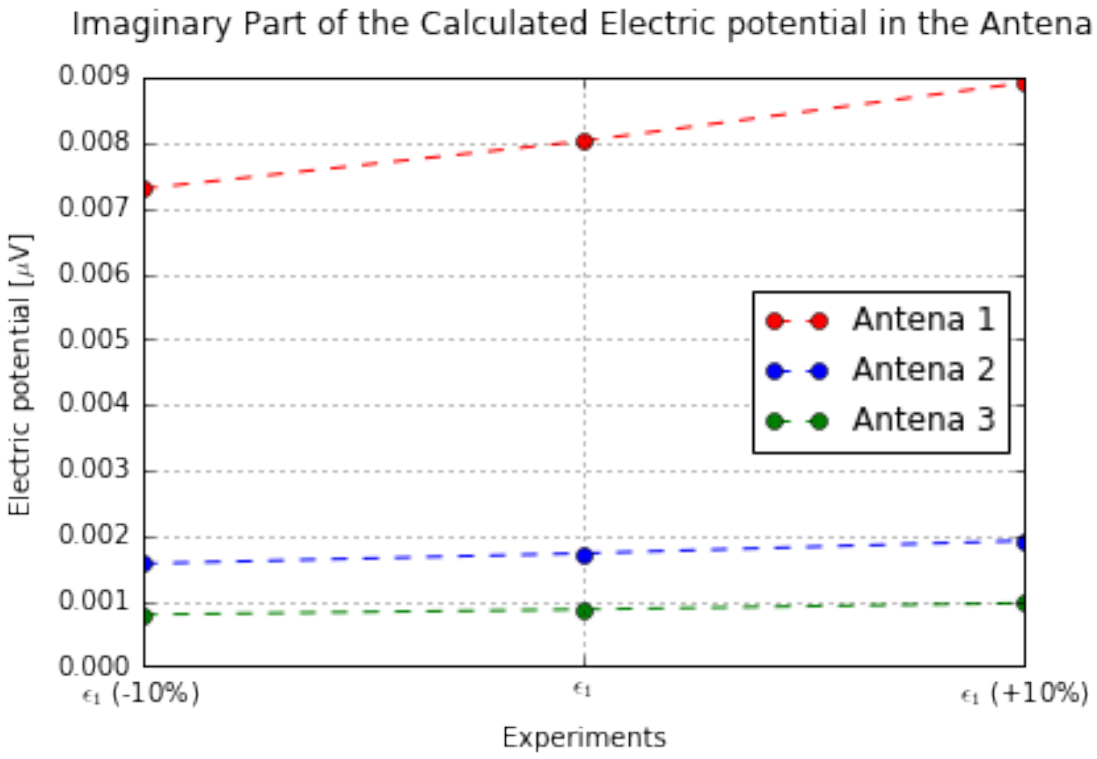
```
Setting the Dirichlet and Neumann spaces...
For the Grid 1:
BEM dofs: 1056
The grid has 1056 elements.
Spaces defined.
-------------------------------------------------
For the Grid 2:
BEM dofs: 992
The grid has 992 elements.
Spaces defined.
-------------------------------------------------

Projecting charges over surface...
Defining operators...
Creating operators...
Solving the equation...
Shape of matrix: (4096, 4096)
iteration 250        rest = 0.0988491734788232
iteration 500        rest = 0.021298651337824707
iteration 750        rest = 0.011734683546011518
iteration 1000        rest = 0.0050421282877942475
iteration 1250        rest = 0.002419767492243415
iteration 1500        rest = 0.0008545421900451696
iteration 1750        rest = 0.00047319494021564766
El sistema fue resuelto en 1821 iteraciones
Equation Solve.
-------------------------------------------------
Is necessary to separate the solution...
Solution divided.
-------------------------------------------------
Calculate Scattered Electric Field in the antena.
The scattered electric field measured by the antenna is:
[ -8.03342197e-06 -8.92374346e-09j  -3.22366280e-07 -1.92235503e-09j
  -8.08035623e-08 -9.69789582e-10j]
-------------------------------------------------
```

```
antena_1_real = np.zeros(len(antena_1))
antena_1_imag = np.zeros(len(antena_1))
antena_2_real = np.zeros(len(antena_1))
antena_2_imag = np.zeros(len(antena_1))
antena_3_real = np.zeros(len(antena_1))
antena_3_imag = np.zeros(len(antena_1))
n_elem = ["$\epsilon_1$ (-10%)", "$\epsilon_1$", "$\epsilon_1$ (+10%)"]
for i in range(len(antena_1)):
antena_1_real[i] = abs(antena_1[i].real)
antena_1_imag[i] = abs(antena_1[i].imag)
antena_2_real[i] = abs(antena_2[i].real)
antena_2_imag[i] = abs(antena_2[i].imag)
antena_3_real[i] = abs(antena_3[i].real)
antena_3_imag[i] = abs(antena_3[i].imag)
```

```
import matplotlib.pyplot as plt
import numpy as np
plt.ioff()
plt.title("Real Part of the Calculated Electric
                                potential in the Antena",y=1.1)
plt.plot(antena_1_real, marker='o', linestyle='--', color='r',
                        label = "Antena 1")
plt.plot(antena_2_real, marker='o', linestyle='--', color='b',
                        label = "Antena 2")
plt.plot(antena_3_real, marker='o', linestyle='--', color='g',
                        label = "Antena 3")
plt.xticks(np.arange(3), n_elem)
plt.ion()
plt.grid(True)
plt.xlabel("Experiments")
plt.ylabel("Electric potential [$\mu$V]")
plt.legend()
plt.legend(loc="right")
plt.show()
```

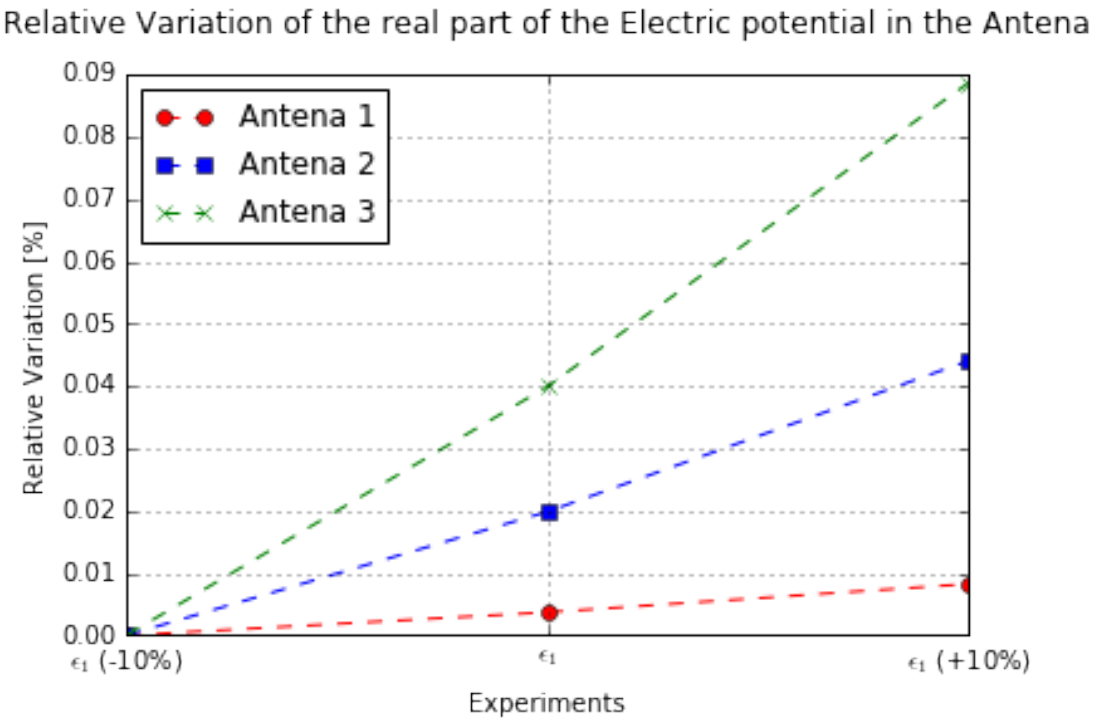



```
import matplotlib.pyplot as plt
import numpy as np
plt.ioff()
plt.title("Imaginary Part of the Calculated Electric
                        potential in the Antena",y=1.05)

plt.plot(antena_1_imag, marker='o', linestyle='--', color='r',
                        label = "Antena 1")
plt.plot(antena_2_imag, marker='o', linestyle='--', color='b',
                        label = "Antena 2")
plt.plot(antena_3_imag, marker='o', linestyle='--', color='g',
                        label = "Antena 3")
plt.xticks(np.arange(3), n_elem)
plt.ion()
plt.grid(True)
plt.xlabel("Experiments")
plt.ylabel("Electric potential [$\mu$V]")
plt.legend()
plt.legend(loc="right")
plt.show()
```

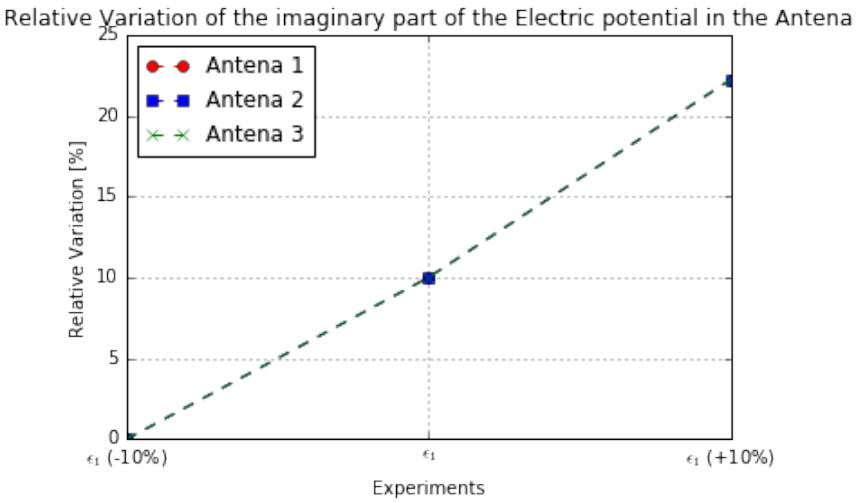



```
Var_1_real = np.zeros(len(antena_1))
Var_2_real = np.zeros(len(antena_1))
Var_3_real = np.zeros(len(antena_1))
for i in range(len(antena_1)):
Var_1_real[i] = abs((antena_1_real[i] - antena_1_real[0])
                        / antena_1_real[0])*100
Var_2_real[i] = abs((antena_2_real[i] - antena_2_real[0])
                        / antena_2_real[0])*100
Var_3_real[i] = abs((antena_3_real[i] - antena_3_real[0])
                        / antena_3_real[0])*100
```

```
import matplotlib.pyplot as plt
import numpy as np
plt.ioff()
plt.title("Relative Variation of the real part of the Electric
                        potential in the Antena",y=1.05)
plt.plot(Var_1_real, marker='o', linestyle='--', color='r',
                        label = 'Antena 1')
plt.plot(Var_2_real, marker='s', linestyle='--', color='b',
                        label = 'Antena 2')
plt.plot(Var_3_real, marker='x', linestyle='--', color='g',
                        label = 'Antena 3')
plt.xticks(np.arange(3), n_elem)
plt.ion()
plt.grid(True)
plt.xlabel("Experiments")
plt.ylabel("Relative Variation [%]")
plt.legend()
plt.legend(loc="upper left")
plt.show()
```



```
Var_1_imag = np.zeros(len(antena_1))
Var_2_imag = np.zeros(len(antena_1))
Var_3_imag = np.zeros(len(antena_1))
for i in range(len(antena_1)):
Var_1_imag[i] = abs((antena_1_imag[i] - antena_1_imag[0])
                        / antena_1_imag[0])*100
Var_2_imag[i] = abs((antena_2_imag[i] - antena_2_imag[0])
                        / antena_2_imag[0])*100
Var_3_imag[i] = abs((antena_3_imag[i] - antena_3_imag[0])
                        / antena_3_imag[0])*100
import matplotlib.pyplot as plt
import numpy as np
plt.ioff()
plt.title("Relative Variation of the imaginary part of the Electric
                        potential in the Antena")
plt.plot(Var_1_imag, marker='o', linestyle='--', color='r',
                        label = 'Antena 1')
plt.plot(Var_2_imag, marker='s', linestyle='--', color='b',
                        label = 'Antena 2')
plt.plot(Var_3_imag, marker='x', linestyle='--', color='g',
                        label = 'Antena 3')
plt.xticks(np.arange(3), n_elem)
plt.ion()
plt.grid(True)
plt.xlabel("Experiments")
plt.ylabel("Relative Variation [%]")
plt.legend()
plt.legend(loc="upper left")
plt.show()
```



Anexo D

Código Python Caso 3 superficies.

In the following lines the following equation will be solved:

$$\begin{bmatrix} \frac{1}{2}I + K_{1-1} & -V_{1-1} & 0 & 0 & 0 & 0 \\ \frac{1}{2}I - K_{1-1} & \frac{\varepsilon_4}{\varepsilon_3}V_{1-1} & K_{1-2} & -V_{1-2} & 0 & 0 \\ -K_{2-1} & \frac{\varepsilon_4}{\varepsilon_3}V_{2-1} & \frac{1}{2}I + K_{2-2} & -V_{2-2} & 0 & 0 \\ 0 & 0 & \frac{1}{2}I - K_{2-2} & \frac{\varepsilon_3}{\varepsilon_2}V_{2-2} & K_{2-3} & -V_{2-3} \\ 0 & 0 & -K_{3-2} & \frac{\varepsilon_3}{\varepsilon_2}V_{3-2} & \frac{1}{2}I + K_{3-3} & -V_{3-3} \\ 0 & 0 & 0 & 0 & \frac{1}{2}I - K_{2-2} & \frac{\varepsilon_2}{\varepsilon_1}V_{3-3} \end{bmatrix} \begin{bmatrix} \varphi_{1d} \\ \frac{\partial}{\partial n}\varphi_{1d} \\ \varphi_{2d} \\ \frac{\partial}{\partial n}\varphi_{2d} \\ \varphi_{3d} \\ \frac{\partial}{\partial n}\varphi_{3d} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{\varepsilon_2-\varepsilon_1}{\varepsilon_3}\frac{\partial \varphi_{i1}}{\partial n} + \frac{\varepsilon_2-\varepsilon_3}{\varepsilon_3}\frac{\partial \varphi_{i2}}{\partial n} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{D.1}$$

and the electric field will be calculated in antennas located in 3 different positions.

```
#Import libraries
import bempp.api
import numpy as np
import timeit
import scipy as sp
import math
bempp.api.set_ipython_notebook_viewer()
```

```
In [2]: #Define grid
grid = bempp.api.import_grid("Cylinder_1.msh")
grid2 = bempp.api.import_grid("Cylinder_2.msh")
matrix = bempp.api.import_grid("Matrix.msh")
number_of_elements = grid.leaf_view.entity_count(0)
number_of_elements2 = grid2.leaf_view.entity_count(0)
number_of_elements3 = matrix.leaf_view.entity_count(0)
print("The cylinder 1 grid has {0} elements.".format(number_of_elements))
print("The cylinder 2 grid has {0} elements.".format(number_of_elements2))
print("The matrix grid has {0} elements.".format(number_of_elements3))
#grid.plot()
#grid2.plot()
#matrix.plot()
```

```
The cylinder 1 grid has 1056 elements.
The cylinder 2 grid has 1056 elements.
The matrix grid has 992 elements.
```

```
#Data
#Unidades de referencia
um = 1
mm = 1.e3
m = 1.e6

#Frecuencia
omega = 1 / 3800

#Permitividad eléctrica
e0 = 8.854*1e-12*1e-18
e1 = 16*e0
e2 = (82629.2677-200138.2211*1.j) * e0
e3 = (82629.2677-200138.2211*1.j) * e0
#Permeabilidad magnética
mu0 = 4. * np.pi * 1e-7 * 1.e6 * um  #micrometros

#Propiedades del material
mu = (-2.9214 + 0.5895j) * mu0
e = (82629.2677-200138.2211*1.j) * e0
cc = 6.5e4 * 1.e-18  #micrometros #Conductividad eléctrica

#Amplitud de la onda en micrometros
Amp = -1. * um #micrometros

#Longitud de onda
Diff_coef = np.sqrt(e0)
lamb = 3800
k = 2 * np.pi *Diff_coef/ lamb

#Antena
antena = np.array([[1.e5, 5.e5, 10.e5],[0., 0., 0.],[0., 0., 0.]])
```

```
In [4]: #Set the Dirichlet and Neumann orders
def function_spaces(grid, grid2, grid3):
order_neumann = 0
order_dirichlet = 0
print("Setting the Dirichlet and Neumann spaces...")
n = grid.leaf_view.entity_count(0)
n2 = grid2.leaf_view.entity_count(0)
n3 = grid3.leaf_view.entity_count(0)
NS = bempp.api.function_space(grid, "DP", order_neumann)
DS = bempp.api.function_space(grid, "DP", order_dirichlet)
NS2 = bempp.api.function_space(grid2, "DP", order_neumann)
DS2 = bempp.api.function_space(grid2, "DP", order_dirichlet)
NS3 = bempp.api.function_space(grid3, "DP", order_neumann)
DS3 = bempp.api.function_space(grid3, "DP", order_dirichlet)
#Print the degrees of freedom
print("For the Grid 1:")
print("BEM dofs: {0}".format(NS.global_dof_count))
print("The grid has {0} elements.".format(n))
print("Spaces defined.")
print("-----------------------------------------------")
print("For the Grid 2:")
print("BEM dofs: {0}".format(NS2.global_dof_count))
print("The grid has {0} elements.".format(n2))
print("Spaces defined.")
print("-----------------------------------------------")
print("For the Grid 3:")
print("BEM dofs: {0}".format(NS3.global_dof_count))
print("The grid has {0} elements.".format(n3))
print("Spaces defined.")
print("-----------------------------------------------")
return NS, DS, NS2, DS2, NS3, DS3, n, n2, n3
```

```
def stern_formulation(DS, NS, DS2, NS2, DS3, NS3, e1, e2, e3, e4):

# Functions to proyect the carges potential to the boundary with constants
def function_1(x, n, domain_index, result):
result[:] = ((e1 - e3) / e3) * (Amp * n[0]) + ((e2 - e3) / e3) * (Amp * n[0])
                        + ((e4 - e3) / e3) * (Amp * n[0])

print("\nProjecting charges over surface...")
charged_grid_fun_1  = bempp.api.GridFunction(DS, fun=function_1)
charged_grid_fun_2  = bempp.api.GridFunction(DS2, fun=function_2)
print("Generatig the RHS of the equation...")
rhs = np.concatenate([np.zeros(NS.global_dof_count),
charged_grid_fun_1.coefficients,
np.zeros(NS.global_dof_count),
np.zeros(NS.global_dof_count),
np.zeros(DS3.global_dof_count),
np.zeros(NS3.global_dof_count)])

print("Defining operators...")
# OPERATOR FOR INTERNAL SURFACE
from bempp.api.operators.boundary import sparse, laplace, modified_helmholtz
idn_1T1  = sparse.identity(DS, DS, DS)
idn_2T2 = sparse.identity(DS2, DS2, DS2)

# Operators in microwires
slp_1T1 = laplace.single_layer(NS, DS, DS)
dlp_1T1 = laplace.double_layer(DS, DS, DS)

slp_2T1 = laplace.single_layer(NS2, DS, DS)
dlp_2T1 = laplace.double_layer(DS2, DS, DS)

slp_3T1 = laplace.single_layer(NS3, DS, DS)
dlp_3T1 = laplace.double_layer(DS3, DS, DS)

# Ec 1
slp_1T2 = laplace.single_layer(NS, DS2, DS2)
dlp_1T2 = laplace.double_layer(DS, DS2, DS2)

slp_2T2 = laplace.single_layer(NS2, DS2, DS2)
dlp_2T2 = laplace.double_layer(DS2, DS2, DS2)

slp_3T2 = laplace.single_layer(NS3, DS2, DS2)
dlp_3T2 = laplace.double_layer(DS3, DS2, DS2)

# Operator in matrix
idn_3T3 = sparse.identity(DS3, DS3, DS3)

# Internal Boudary
slp_1T3 = laplace.single_layer(NS, DS3, DS3)
```

```
dlp_1T3 = laplace.double_layer(DS, DS3, DS3)
slp_2T3 = laplace.single_layer(NS2, DS3, DS3)
dlp_2T3 = laplace.double_layer(DS2, DS3, DS3)

slp_3T3 = laplace.single_layer(NS3, DS3, DS3)
dlp_3T3 = laplace.double_layer(DS3, DS3, DS3)

print("Creating operators...")
# Matrix Assemble
blocked = bempp.api.BlockedOperator(6, 6)
blocked[0, 0] = .5*idn_3T3 + dlp_3T3
blocked[0, 1] = -slp_3T3
# blocked[0, 2] = 0
# blocked[0, 3] = 0
# blocked[0, 4] = 0
# blocked[0, 5] = 0

# Original formulation
blocked[1, 0] = .5*idn_3T3 - dlp_3T3
blocked[1, 1] =  (e4/e3)*slp_3T3
blocked[1, 2] =  dlp_2T3
blocked[1, 3] = -slp_2T3
#blocked[1, 4] = 0
#blocked[1, 5] = 0

blocked[2, 0] = -dlp_3T2
blocked[2, 1] =  (e4/e3)*slp_3T2
blocked[2, 2] = .5*idn_2T2 + dlp_2T2
blocked[2, 3] = -slp_2T2
#blocked[2, 4] = 0
#blocked[2, 5] = 0

#blocked[3, 0] = 0
#blocked[3, 1] = 0
blocked[3, 2] = .5*idn_2T2 - dlp_2T2
blocked[3, 3] = (e3/e2)*slp_2T2
blocked[3, 4] = dlp_1T2
blocked[3, 5] = -slp_1T2

#blocked[4, 0] = 0
#blocked[4, 1] = 0
blocked[4, 2] = .5*idn_1T1 - dlp_1T1
blocked[4, 3] = (e3/e2)*slp_2T1
blocked[4, 4] = .5*idn_1T1 + dlp_2T1
blocked[4, 5] = -slp_2T1

#blocked[5, 0] = 0
#blocked[5, 1] = 0
```

```
#blocked[5, 2] = 0
#blocked[5, 3] = 0
blocked[5, 4] = .5*idn_1T1 - dlp_1T1
blocked[5, 5] = (e2/e1)*slp_1T1

A = blocked.strong_form()

return A, rhs
```

```
class gmres_counter(object):
def __init__(self, disp=True):
self._disp = disp
self.niter = 0
def __call__(self, rk=None):
self.niter += 1
if self.niter %250 == 0:
if self._disp:
print('iteration %3i\trest = %s' % (self.niter, str(rk)))
```

```
def solving(blocked,rhs):
print("Solving the equation...")
#Sistema de ecuaciones
import inspect
from scipy.sparse.linalg import gmres
counter = gmres_counter()
print("Shape of matrix: {0}".format(blocked.shape))
aa = blocked.shape
a = aa[0]/2
soln,info = gmres(blocked, rhs, tol=1e-4, callback = counter,
                        maxiter = 5000, restart = 5000)
print("El sistema fue resuelto en {0} iteraciones".format(counter.niter))
np.savetxt("Solucion.out", soln, delimiter=",")
print("Equation Solve.")
print("-----------------------------------------------")
return soln, a
```

```
#Divide the solution
def divide(soln, DS, NS, DS2, NS2, DS3, NS3, n, n2, n3):
print("Is necessary to separate the solution...")
phi_1=soln[0:n]
dphi_1=soln[n:2*n]
phi_2=soln[2*n:2*n+n2]
dphi_2=soln[2*n+n2:2*n+2*n2]
phi_3=soln[2*n+2*n2:2*n+2*n2+n3]
dphi_3=soln[2*n+2*n2+n3:-1]
phi_1s = bempp.api.GridFunction(DS, coefficients=phi_1)
dphi_1s = bempp.api.GridFunction(DS, coefficients=dphi_1)
phi_2s = bempp.api.GridFunction(DS2, coefficients=phi_2)
dphi_2s = bempp.api.GridFunction(DS2, coefficients=dphi_2)
phi_3s = bempp.api.GridFunction(DS2, coefficients=phi_3)
dphi_3s = bempp.api.GridFunction(DS2, coefficients=dphi_3)
print("Solution divided.")
print("-------------------------------------------------")
return(phi_1s, dphi_1s, phi_2s, dphi_2s, phi_3s, dphi_3s)
```

```
def calculo_antena(DS,antena,solution_dirichl_s,solution_neumann_s):
#Calculo campo en antena
slp_pot_ext = bempp.api.operators.potential.laplace.single_layer(DS, antena)
dlp_pot_ext = bempp.api.operators.potential.laplace.double_layer(DS, antena)
Campo_en_antena_dis = (dlp_pot_ext * solution_dirichl_s
                       - slp_pot_ext * solution_neumann_s).ravel()
print("Calculate Scattered Electric Field in the antena.")
print("The scattered electric field measured by the antenna
                       is: ".format(Campo_en_antena_dis))
print(Campo_en_antena_dis)
print("-------------------------------------------------")
return Campo_en_antena_dis
```

```
def BEM(grid, matrix, ep_in, ep_ex):
NS, DS, NS2, DS2, NS3, DS3, n, n2, n3 = function_spaces(grid, grid, matrix)
A, rhs = stern_formulation(DS, NS, DS2, NS2, DS3, NS3, e0, e1, e2, e3)
soln, a  = solving(A, rhs)
phi, dphi, phi_2s, dphi_2s, phi_3s, dphi_3s = divide(soln, DS, NS, DS2, NS2,
                       DS3, NS3, n, n2, n3)
calc_antena = calculo_antena(DS, antena, phi, dphi)
return calc_antena
```

```
calc = BEM(grid, matrix, e1, e0)
```

```
Setting the Dirichlet and Neumann spaces...
For the Grid 1:
```

```
BEM dofs: 1056
The grid has 1056 elements.
Spaces defined.
-----------------------------------------------
For the Grid 2:
BEM dofs: 1056
The grid has 1056 elements.
Spaces defined.
-----------------------------------------------
For the Grid 3:
BEM dofs: 992
The grid has 992 elements.
Spaces defined.
-----------------------------------------------

Projecting charges over surface...
Generatig the RHS of the equation...
Defining operators...
Creating operators...
Solving the equation...
Shape of matrix: (6208, 6208)
iteration 250        rest = 0.7038874594554739
iteration 500        rest = 0.37718700985471604
iteration 750        rest = 0.3528730125166427
iteration 1000        rest = 0.13615873202729406
iteration 1250        rest = 0.1132367096154362
iteration 1500        rest = 0.04568073999584019
iteration 1750        rest = 0.01814937882499548
iteration 2000        rest = 0.0007841339963141831
El sistema fue resuelto en 2009 iteraciones
Equation Solve.
-----------------------------------------------
Is necessary to separate the solution...
Solution divided.
-----------------------------------------------
Calculate Scattered Electric Field in the antena.
The scattered electric field measured by the antenna is:
[ 0.76989615+0.13639286j  0.15407255+0.02729511j  0.07704211+0.01364859j]
-----------------------------------------------
```