TESIS

October 5, 2018

```
In [2]: import bempp.api
        import numpy as np
        import dolfin

        bempp.api.set_ipython_notebook_viewer()
        bempp.api.global_parameters.quadrature.near.double_order = 4
        bempp.api.global_parameters.quadrature.medium.double_order = 4
        bempp.api.global_parameters.quadrature.far.double_order = 4
```

```
In [3]: #grid = bempp.api.shapes.cylinder()
        grid = bempp.api.import_grid("Cilindro_005.msh")
        dirichlet_segments=[1]
        neumann_segments=[2]
        # Print out the number of elements
        number_of_elements = grid.leaf_view.entity_count(0)

        print("The grid has {0} elements.".format(number_of_elements))
        #grid.plot()
```

```
The grid has 7480 elements.
```

```
In [6]: order_neumann = 0
        order_dirichlet = 0

        global_neumann_space = bempp.api.function_space(grid, "DP", order_neumann)
        global_dirichlet_space = bempp.api.function_space(grid, "DP", order_dirichlet)

        NS = global_neumann_space
        DS = global_dirichlet_space

        ep1 = 200
        ep2 = 8
        k = 0.125

        print("BEM dofs: {0}".format(NS.global_dof_count))
```

```
BEM dofs: 7480
```

```
In [7]: #Dirichlet Segment
        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)


        #Formación del Operador de Calderón
        blocked = bempp.api.BlockedOperator(2, 2)

        blocked[0, 0] = 0.5 * id + dlp
        blocked[0, 1] = -slp
        blocked[1, 0] = 0.5 * id - dlp
        blocked[1, 1] = ep1/ep2 * slp

In [8]: def funcion1(x, n, domain_index, result):
            global ep1
            result[:] = ((ep2 - ep1) / ep2) * (1. * 1j * k * n[0] * np.exp(1j * k * x[0]))

        def cero(x, n, domain_index, result):
            result[:] = 0

        funcion_fun = bempp.api.GridFunction(DS, fun=funcion1)

        cero_fun = bempp.api.GridFunction(NS, fun=cero)

In [9]: sol, info, it_count = bempp.api.linalg.gmres(blocked, [cero_fun, funcion_fun],
                                              use_strong_form=True, return_iteration_count

        print("The linear system was solved in {0} iterations".format(it_count))

The linear system was solved in 155 iterations


In [10]: solution_dirichl, solution_neumann = sol
         solution_dirichl.plot()

/usr/lib/python3/dist-packages/matplotlib/font_manager.py:273: UserWarning:

Matplotlib is building the font cache using fc-list. This may take a moment.

/usr/lib/python3/dist-packages/matplotlib/font_manager.py:273: UserWarning:

Matplotlib is building the font cache using fc-list. This may take a moment.



In [11]: n_grid_points = 200
         xmin, xmax, ymin, ymax=[-3,3,-3,3]
```

```
plot_grid = np.mgrid[xmin:xmax:n_grid_points*1j,ymin:ymax:n_grid_points*1j]
points = np.vstack((plot_grid[0].ravel(),
                    plot_grid[1].ravel(),
                    np.zeros(plot_grid[0].size)))
```

In [12]:
```
dp0_space = bempp.api.function_space(grid, "DP", 0)
p1_space = bempp.api.function_space(grid, "DP", 0)
```

In [13]:
```
slp_pot = bempp.api.operators.potential.laplace.single_layer(dp0_space, points)
dlp_pot = bempp.api.operators.potential.laplace.double_layer(p1_space, points)
```

In [14]:
```
u_evaluated = slp_pot * solution_neumann - dlp_pot * solution_dirichl
```

In [15]:
```
# The next command ensures that plots are shown within the IPython notebook
%matplotlib inline


from matplotlib import pylab as plt


fig,ax = plt.subplots()
ax.scatter(u_evaluated.real,u_evaluated.imag)
# Filter out solution values that are associated with points outside the unit circle.
u_evaluated = (u_evaluated).reshape((n_grid_points,n_grid_points))
radius = np.sqrt(plot_grid[0]**2 + plot_grid[1]**2)
u_evaluated[radius>2] = np.nan
fig = plt.figure(figsize=(10, 8))
plt.imshow(((u_evaluated.real)), extent=(-3,3,-3,3))
plt.title('Computed solution')
plt.colorbar()
```

Out[15]:
```
<matplotlib.colorbar.Colorbar at 0x7fe67fcf3c18>
```

In [16]:
```
#radius = np.sqrt(plot_grid[0]**2 + plot_grid[1]**2)
#u_evaluated[radius>2] = np.nan
#fig = plt.figure(figsize=(10, 8))
#plt.imshow(((u_evaluated.imag)), extent=(-3,3,-3,3))
#plt.title('Computed solution')
#plt.colorbar()
```

In [18]:
```
#Preambulo
import numpy as np
import bempp.api
omega = 2.*np.pi*10.e9
e0 = 8.854*1e-12*1e-18
mu0 = 4.*np.pi*1e-7*1e6
mue = (1.)*mu0
ee = (16.)*e0
mui = (-2.9214+0.5895j)*mu0
```

```
ei = (82629.2677-200138.2211j)*e0
k = omega*np.sqrt(e0*mu0)
lam = 2*np.pi/k
nm = np.sqrt((ee*mue)/(e0*mu0))
nc = np.sqrt((ei*mui)/(e0*mu0))
alfa_m = mue/mu0
alfa_c = mui/mue
antena = np.array([[1e4],[0.],[0.]])
print("Numero de onda exterior:", k)
print("Indice de refraccion matriz:", nm)
print("Indice de refraccion conductor:", nc)
print("Numero de onda interior matriz:", nm*k)
print("Numero de onda interior conductor:", nm*nc*k)
print("Indice de transmision matriz:", alfa_m)
print("Indice de transmision conductor:", alfa_c)
print("Longitud de onda:", lam, "micras")
```

```
Numero de onda exterior: 0.0002095822793
Indice de refraccion matriz: 4.0
Indice de refraccion conductor: (510.829219424+619.966251289j)
Numero de onda interior matriz: 0.000838329117198
Numero de onda interior conductor: (0.428243008559+0.519735760136j)
Indice de transmision matriz: 1.0
Indice de transmision conductor: (-2.9214+0.5895j)
Longitud de onda: 29979.5637693 micras
```

In [19]:
```
#Importando mallas
grid_0 = bempp.api.import_grid("Cilindro_005.msh")
```

In [20]:
```
#Funciones de dirichlet y neumann
def dirichlet_fun(x, n, domain_index, result):
        result[0] = 1. * np.exp(1j * k * x[0])
def neumann_fun(x, n, domain_index, result):
        result[0] = 1. * 1j * k * n[0] * np.exp(1j * k * x[0])
```

In [21]:
```
#Operadores multitrazo
Ai_0 = bempp.api.operators.boundary.helmholtz.multitrace_operator(grid_0, nm * nc * k)
Ae_0 = bempp.api.operators.boundary.helmholtz.multitrace_operator(grid_0, nm * k)

#Transmision en Multitrazo
Ai_0[0,1] = Ai_0[0,1]*alfa_c
Ai_0[1,1] = Ai_0[1,1]*alfa_c

#Acople interior y exterior
op_0 = (Ai_0 + Ae_0)
```

In [22]:
```
#Espacios
dirichlet_space_0 = Ai_0[0,0].domain
neumann_space_0 = Ai_0[0,1].domain
```

In [23]:
```
#Operadores identidad
ident_0 = bempp.api.operators.boundary.sparse.identity(neumann_space_0, neumann_space_0
```

In [26]:
```
#Matriz de operadores
blocked = bempp.api.BlockedOperator(2,2)
```

In [27]:
```
#Diagonal
blocked[0,0] = op_0[0,0]
blocked[0,1] = op_0[0,1]
blocked[1,0] = op_0[1,0]
blocked[1,1] = op_0[1,1]
blocked[1,1] = blocked[1,1] + 0.5 * ident_0 * (alfa_c - 1)
```

In [30]:
```
#Condiciones de borde
dirichlet_grid_fun_0 = bempp.api.GridFunction(dirichlet_space_0, fun=dirichlet_fun)
neumann_grid_fun_0 = bempp.api.GridFunction(neumann_space_0, fun=neumann_fun)

#Discretizacion lado derecho
rhs = np.concatenate([dirichlet_grid_fun_0.coefficients, neumann_grid_fun_0.coefficient
```

In [ ]:
```
#Discretizacion lado izquierdo
blocked_discretizado = blocked.strong_form()
```

In [ ]:
```
#Sistema de ecuaciones
import inspect
from scipy.sparse.linalg import gmres
array_it = np.array([])
array_frame = np.array([])
it_count = 0
def iteration_counter(x):
        global array_it
        global array_frame
        global it_count
        it_count += 1
        frame = inspect.currentframe().f_back
        array_it = np.append(array_it, it_count)
        array_frame = np.append(array_frame, frame.f_locals["resid"])
        print it_count, frame.f_locals["resid"]
print("Shape of matrix: {0}".format(blocked_discretizado.shape))
x,info = gmres(blocked_discretizado, rhs, tol=1e-5, callback = iteration_counter, maxite
print("El sistema fue resuelto en {0} iteraciones".format(it_count))
np.savetxt("Solucion.out", x, delimiter=",")
```

In [ ]:
```
#Campo interior
interior_field_dirichlet_m = bempp.api.GridFunction(dirichlet_space_m, coefficients=x[:d
interior_field_neumann_m = bempp.api.GridFunction(neumann_space_m,coefficients=x[dirichl

#Campo exterior
exterior_field_dirichlet_m = interior_field_dirichlet_m
```

```
exterior_field_neumann_m = interior_field_neumann_m*(1./alfa_m)

#Calculo campo en antena
slp_pot_ext_m = bempp.api.operators.potential.helmholtz.single_layer(dirichlet_space_m,
dlp_pot_ext_m = bempp.api.operators.potential.helmholtz.double_layer(dirichlet_space_m,
Campo_en_antena = (dlp_pot_ext_m * exterior_field_dirichlet_m - slp_pot_ext_m * exterior
print "Valor del campo en receptor:", Campo_en_antena
```