In [ ]:

```
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 [ ]:

```
#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()
```

In [ ]:

```
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))
```

In [ ]:

```
#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 [ ]:

```
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 [ ]:

```
sol, info, it_count = bempp.api.linalg.gmres(blocked, [cero_fun, funcion_fun],
                                             use_strong_form=True, return_iteration_count=True, tol=
e-3)

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

In [ ]:

```
#Discretizacion lado izquierdo
blocked_discretizado = blocked.strong_form()

#Discretizacion lado derecho
rhs = np.concatenate([cero_fun.coefficients, funcion_fun.coefficients])

#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, maxiter = 50000)
print("El sistema fue resuelto en {0} iteraciones".format(it_count))
np.savetxt("Solucion.out", x, delimiter=",")
```

In [ ]:

```
solution_dirichl, solution_neumann = sol
solution_dirichl.plot()
```

In [ ]:

```
n_grid_points = 300
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 [ ]:

```
dp0_space = bempp.api.function_space(grid, "DP", 0)
p1_space = bempp.api.function_space(grid, "DP", 0)
```

In [ ]:

```
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 [ ]:

```
u_evaluated = slp_pot * solution_neumann - dlp_pot * solution_dirichl
```

In [ ]:

```
# 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()
```