END-SEMESTER EXAMINATION

Nithin Babu [EE18B021]

July 30, 2020

Abstract

The objective of the question is the following:

- To analyze the system and calculate the potential matrix, electric field and charges.
- To develop an algorithm to determine height of the fluid from the observed resonant frequency.
- To find out the variation of Q_{top} and Q_{fluid} with respect to the height of the fluid.
- To check for the continuity of the the electric displacement field at the boundary of the fluid.
- To find out if Snell's Law will be valid at the fluid boundary.

Theoretical concepts required

The electric field is related to the potential as:

$$E = -\nabla V$$

The Gauss law can be expressed as:

$$\nabla \cdot E = \rho/\epsilon$$

The capacitance of a capacitor can be expressed as:

$$C = Q/V$$

The Laplace's equation can be solved in python by using the expressions:

$$\phi_{m,n} = \frac{\phi_{m-1,n} + \phi_{m+1,n} + \phi_{m,n-1} + \phi_{m,n+1}}{4} \quad , m \neq k$$

$$\phi_{k,n} = \frac{\epsilon_r \phi_{k-1,n} + \phi_{k+1,n}}{1 + \epsilon_r} \quad , m = k$$

Algorithm to determine the height from the observed resonant frequency

The algorithm to determine the height of the fluid from the observed resonant frequency is as follows:

1. From the resonant frequency, $\omega_o = \frac{1}{\sqrt{LC}}$, we can get the value of C.
2. Using C, we can calculate the Charge on the plates by using the relation $Q_{top} = CV$.
3. We already have the plots and the values of h/L_y vs Q_{top}, we can use **scipy.interpolate.CubicSpline()** to find the best fit for the curves.
4. Once, this is done, for any value of Q_{top}, we can find the corresponding h/L_y, and in turn find the corresponding h, the height of the fluid.

Parallelizing the computation to calculate the potential matrix

We have already computed the solution for the Laplace's equation during one of the previous assignments. But in this case, there will be a small catch.

The expression changes during the boundary and we will have to take that into account. The python code snippet to take care of this situation is as follows.

```
for p in range(N_max):
        N_iter = N_iter + 1
        oldphi = phi.copy()

# This is the parallelize algorithm followed.
# The matrix is first divided into two parts and calculated accordingly.
        phi[1:k,1:-1] = 0.25*(phi[1:k,0:-2] + phi[1:k,2:]
                              + phi[0:k-1,1:-1] + phi[2:k+1,1:-1])

        phi[k+1:-1,1:-1] = 0.25*(phi[k+1:-1,0:-2] + phi[k+1:-1,2:]
                                 + phi[k:-2,1:-1] + phi[k+2:,1:-1])

# The boundary conditions are also being assigned here.
        phi[k,1:-1] = (E_r*phi[k-1,1:-1] + phi[k+1,1:-1])/(1 + E_r)
        phi[-1,1:-1] = 1.0
```

What is happening here is that inside the loop, $phi_{m,n}$ is divided into three branches, the first one is $m < k$, $m > k$, and $m > k$. Hence, individual

parallelization takes place for each of these cases.

This way of parallelizing reduces the need of any conditional or looping statements, and hence, the code will be much more faster and efficient when compared to the conditional or loop cases.

Function to solve the Laplace's equation and calculate $\phi_{m,n}$

The following python code snippet is used to calculate $\phi_{m,n}$:

```
# Declaring the function to solve the Laplace's Equation
# and calculate the Potential Matrix.
def potential_calculator(M, N, delta, k, accuracy, N_max, graph = 'No'):

# Declaring all the necessary variables and making the meshgrid.
        E_r = 2; N_iter = 0
        Ly = arange(0, delta*M, delta)
        Lx = arange(0, delta*N, delta)

        X,Y = meshgrid(Ly,Lx)

        error = zeros(N_max)
        phi = zeros((M,N))

# This for loop is run for N_max number of times
# so that the potential matrix converges.
        for p in range(N_max):
                N_iter = N_iter + 1
                oldphi = phi.copy()

# This is the parallelize algorithm followed.
# The matrix is first divided into two parts and calculated accordingly.
                phi[1:k,1:-1] = 0.25*(phi[1:k,0:-2] + phi[1:k,2:]
                                      + phi[0:k-1,1:-1] + phi[2:k+1,1:-1])
                phi[k+1:-1,1:-1] = 0.25*(phi[k+1:-1,0:-2] + phi[k+1:-1,2:]
                                         + phi[k:-2,1:-1] + phi[k+2:,1:-1])

# The boundary conditions are also being assigned here.
                phi[k,1:-1] = (E_r*phi[k-1,1:-1] + phi[k+1,1:-1])/(1 + E_r)
                phi[-1,1:-1] = 1.0

# The error matrix is updated so that, once the error is less than the accuracy,
```

```
# the loop is broken.
                error[p] = abs(oldphi - phi).max()

                if error[p] <= accuracy:
                        break;

# A spacial contour plot for the potential matrix.
        if graph == 'Yes':

                figure(0)
                contourf(Y,X,phi.T)
                colorbar()
                title("The Potential Diagram")
                xlabel(r'$X(cm)\rightarrow$')
                ylabel(r'$Y(cm)\rightarrow$')
                grid(True)

        show()

# The function returns the potential matrix, the error vector
# and the number of iterations necessary.
        return phi, error, N_iter
```

The spacial plot of $\phi_{m,n}$ is as shown in Figure 1.

The error vector calculated is also plotted against the iteration number. It can be seen that as N tends to ∞, the error tends to 0. The plot is as shown in Figure 2:

Q_{top} and Q_{fluid} for different values of h/L_y

h/L_y goes from 0, 0.1, 0.2, ..., 0.9. We need to find how Q_{top} and Q_{fluid} varies with h/L_y and check if it's linear or not. The python code snippet below calculates the values of Q_{top} and Q_{fluid} and also plots them against h/L_y.

```
# Declaring the function to calculate the two components of the Electric field
# from the potential Matrix.
def field_calculator(phi, delta):

# The below piece of code will calculate the components of Electric field
# at the centre of the mesh cells.
        Ex = 100*(phi[1:,0:-1] - phi[1:,1:])/(delta)
        Ey = 100*(phi[0:-1,1:] - phi[1:,1:])/(delta)
```

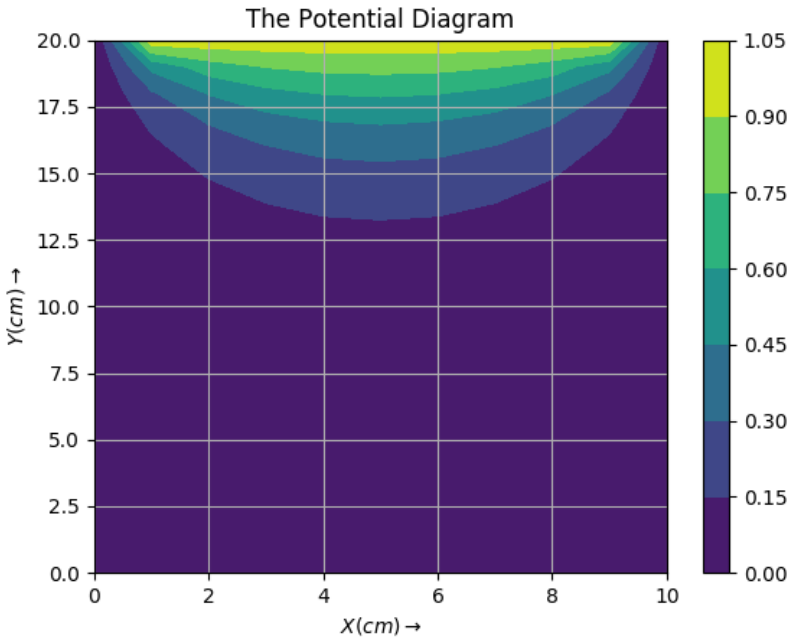

Figure 1: The Spacial plot of $\phi_{m,n}$

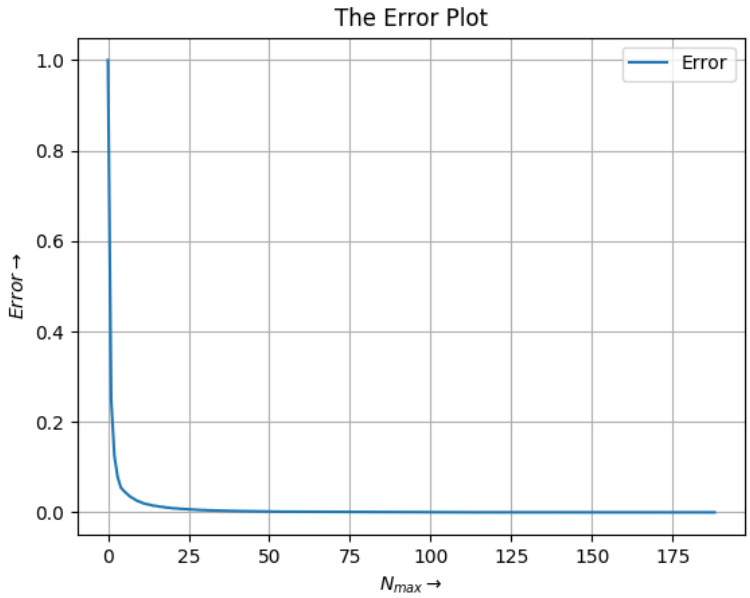

Figure 2: The Error Plot

```
        return Ex, Ey

# Declaring the function to calculate the top plate charge and
# the fluid charge using the Electric field components.
def charge_calculator(Ex,Ey,delta,Lz):

# The below piece of code is an approximate to the integration
# required to get the charge from Gauss Law.
        Qtop = abs(sum(Epsilon*Ey[-1,:]*delta))*Lz*1e-4

        Qfluid = -abs(sum(Epsilon*Ey[0,:]*delta))*Lz*1e-4
                 - (abs(sum(Epsilon*Ex[0:k-1,-1]*delta))*Lz*1e-4)

        return Qtop, Qfluid

#Calculating the values of the two charges for different values of h.
Qtop = []
Qfluid = []
for h in arange(0.1,1,0.1):

# Calling all the functions one by one to calculate the 2 charge values.
        k = int(h*(M-1))
        phi,_,_ = potential_calculator(M,N,delta,k,accuracy,N_max)
        Ex, Ey = field_calculator(phi, delta)
        Qt, Qf = charge_calculator(Ex, Ey, delta, Lz)
        Qtop.append(Qt)
        Qfluid.append(Qf)

# The plot of the top plate charge vs h/Ly.
figure(3)
plot(arange(0.1,1,0.1),Qtop,label='Top Plate Charge')
title("The Top plate Charge Plot")
xlabel(r'$h/L_y\rightarrow$')
ylabel(r'$Charge\rightarrow$')
legend()
grid(True)

# The plot of the fluid charge vs h/Ly.
figure(4)
plot(arange(0.1,1,0.1),Qfluid,label='Fluid Charge')
title("The Fluid Charge Plot")
xlabel(r'$h/L_y\rightarrow$')
ylabel(r'$Charge\rightarrow$')
legend()
```

```
grid(True)

show()
```

The plots of Q_{top} and Q_{fluid} are as shown in Figure 3 and Figure 4.

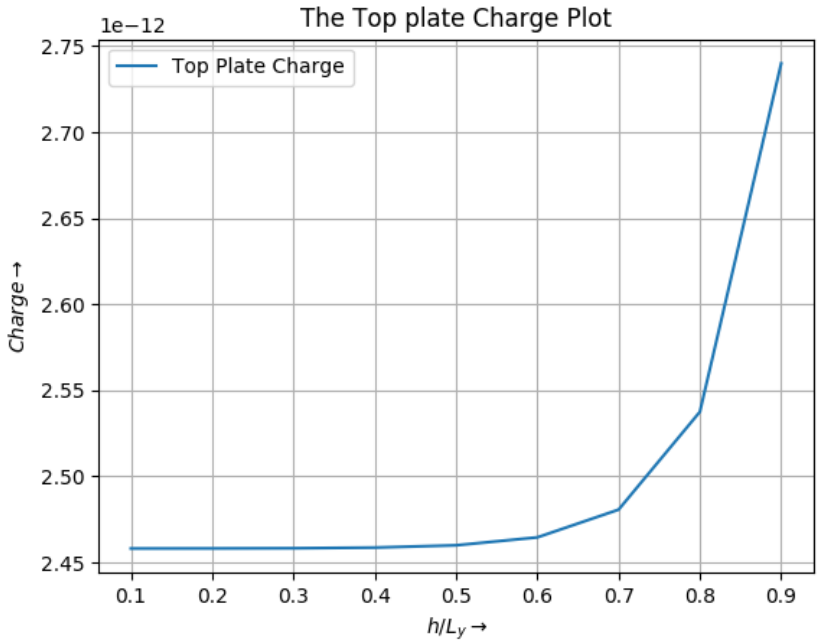

Figure 3: Q_{top} vs h/L_y

It is clearly evident that the plots are not linear. We can tell that the plots could not have been linear because the capacitance of the tank does not depend linearly to the height of the dielectric liquid. Hence, The charges also will not depend linearly with the height of the liquid.

Continuity of D_n

From the boundary conditions of the Electromagnetic theory and the Maxwell's equations, we can tell that, at any certain boundary with no charges, the normal component of the electric displacement field must be continuous. The following python code snippet proves it:

```
# Declaring all the necessary variables required for
# answering the other parts of the question.
# Let us assume the other dimension Lz to be 10cm.
Lz = 10
```

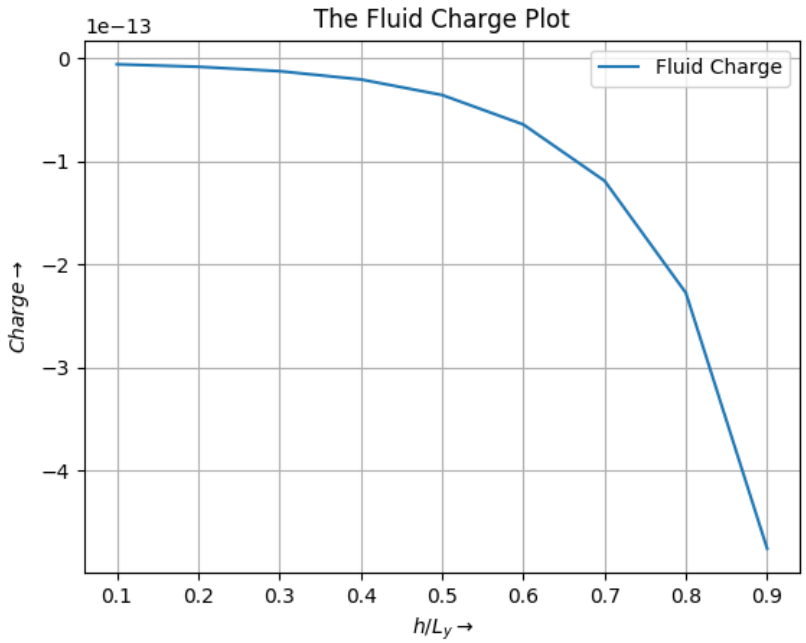

Figure 4: Q_{fluid} vs h/L_y

```
Epsilon = 8.85e-12
M = 21; N = 11; delta = 1
k = 10; accuracy = 1e-5; N_max = 1000

# Declaring the function to calculate the two components of
# the Electric field from the potential Matrix.
def field_calculator(phi, delta):

# The below piece of code will calculate the components of
# Electric field at the centre of the mesh cells.
        Ex = 100*(phi[1:,0:-1] - phi[1:,1:])/(delta)
        Ey = 100*(phi[0:-1,1:] - phi[1:,1:])/(delta)

        return Ex, Ey

# The below piece of code is to show that Dn remains continuous
# before and after the fluid boundary.
k = 10
phi,_,_ = potential_calculator(M,N,delta,k,accuracy,N_max)
Ex, Ey = field_calculator(phi, delta)
```

```
# Calculating the values of Dn0 (Dn after the boundary),
# and Dn1 (Dn before the boundary)
Dn0 = abs(Ey[k-1,:-1]*2*Epsilon); Dn1 = abs(Ey[k,:-1]*Epsilon)
err_D = abs((Dn1 - Dn0))/Dn0;

# Printing the two values of Dn along with the relative mean error.
print("\nThe values of Dn0 and Dn1 at the surface Yk are: ")
print("\nDno: ",Dn0)
print("\nDn1: ",Dn1)
print("\nThe relative mean error is :",mean(err_D))

if mean(err_D)<=1e-15:
        print("Hence, Dn is continuous.")

else:
        print("Hence, Dn is not continuous.")
```

The spacial electric field plot is as shown:

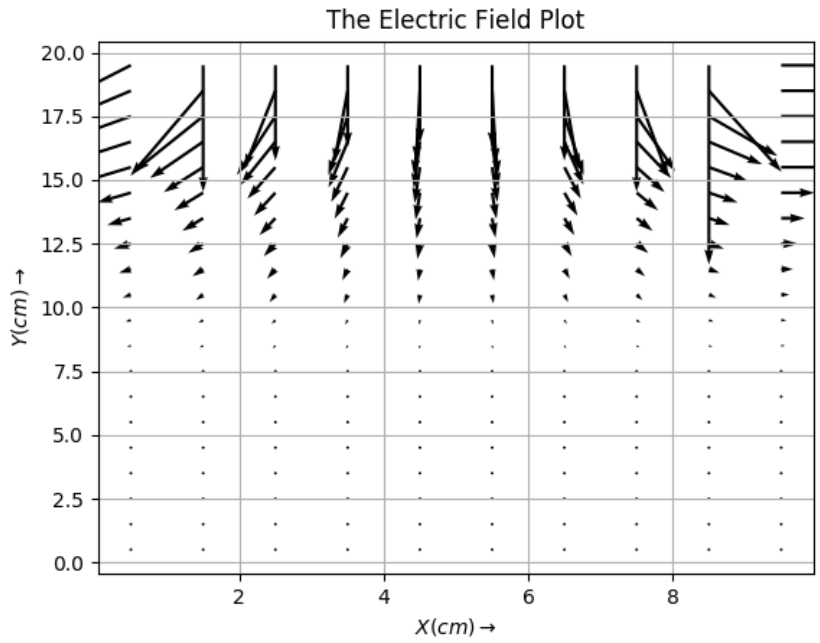

Figure 5: The Electric field plot

The ouput for the following code snippet is as shown:

The average values of Dn0 and Dn1 at the surface Yk are:

Dno: 1.4714781852494838e-11
Dn1: 1.4714781852494828e-11

The relative mean error is : 4.538648931883607e-16
Hence, Dn is continuous.
Hence, it has been proved that D_n is continuous at the boundary.

Testing Snell's Law

The electric fields tend to change at the boundary. Hence the, angles can be calculated to verify if Snell's Law is valid at this boundary. The following code snippet computes the angles to verify Snell's Law.

```
# The below piece of code is to check if Snell's Law is valid at the boundary.

# The Snell's terms are calculated using Ex, Ey.
tetha_1 = arctan(Ey[k-1][5]/Ex[k-1][5]); tetha_2 = arctan(Ey[k][5]/Ex[k][5])
Snell_term_1 = sqrt(2)*sin(tetha_1); Snell_term_2 = sin(tetha_2)

err_snell = abs((Snell_term_2 - Snell_term_1)/Snell_term_1)

print("\nThe change in the angle of the Electric field is :",abs(tetha_2-tetha_1))

# Printing the relative error in between the Snell terms.
print("\nThe Snell's Law relative error is: ",err_snell)

if err_snell<=1e-15:
        print("Hence, Snell's Law is valid.")

else:
        print("Hence, Snell's Law is invalid.")
```

The output of the code snippet is as follows:
The change in the angle of the Electric field is : 0.04269207456192414

The Snell's Law relative error is: 0.2878621347660341
Hence, Snell's Law is invalid.

Hence, It is evident that Snell's Law is invalid. We can also that the Electric field wont follow Snell's Law as it governs the properties of only electromagnetic waves, There must be present of a mutually perpendicular Electric and magnetic fields for Snell's Law to be true, which is absent in this case.