End Semester Examination

Abhishek Sekar
EE18B067

August 1, 2020

Contents

0.1 Set Up

- The problem involves a rectangular conducting tank, of dimensions Ly x Lx where Ly is 20cm and Lx is 10cm.
- This tank is filled with a fluid, to a height h. This fluid behaves like a dielectric with dielectric constant K=2.
- The sides of the tank are grounded as shown in the figure below.So for now, it vaguely resembles a capacitor as given in the question.It is also given that the other side of the tank has a potential of 1V.
- This tank is further connected to a resistor, inductor and an AC voltage source to mimic an RLC network.

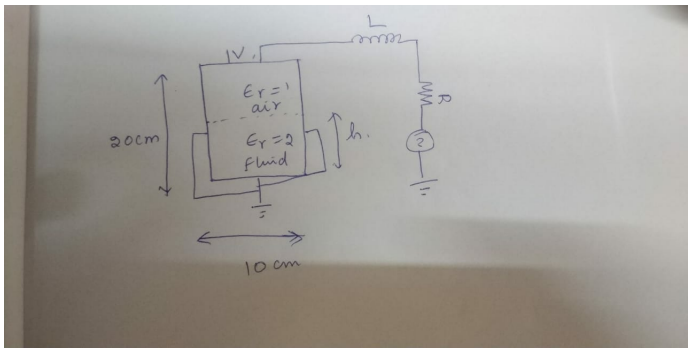

Figure 1: The Set Up

0.2 The Questions

0.2.1 Creating a function to solve the laplace equations

The Question

Create a python function that solves Laplace's Equation and obtains mn on the grid. The function accepts as arguments: M, the number of nodes along x, including the boundary nodes. N, the number of nodes along y, including the boundary nodes. , the distance between nodes (assumed same along x and along y) k, the height given as the index k corresponding to h. , The desired accuracy . N0 the maximum number of iterations to complete. The top boundary is assumed to be at 1 volt. It should return phi[M,N] The array of solved potential values correct to N, Number of iterations actually carried out err[N], The vector of errors Note that the error in phi should be extrapolated to

Theory

This section will be approached in a similar manner to experiment 5,where we solved the laplace equation with Finite Difference Method. This method is discussed below.

- Given below is the laplace equation for potential ϕ

$$\nabla^2\phi = 0 \tag{1}$$

- Here we use a 2-D setup(assuming the tank's third dimension is uniform) so the Numerical solutions in 2D can be easily transformed into a difference equation. The equation can be written as

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0 \tag{2}$$

$$\frac{\partial\phi}{\partial x}_{(x_i,y_j)} = \frac{\phi(x_{i+1/2}, y_j) - \phi(x_{i-1/2}, y_j)}{\Delta x} \tag{3}$$

$$\frac{\partial^2\phi}{\partial x^2}_{(x_i,y_j)} = \frac{\phi(x_{i+1}, y_j) - 2\phi(x_i, y_j) + \phi(x_{i-1}, y_j)}{(\Delta x)^2} \tag{4}$$

- Using above equations we get

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4} m \neq k, 0 < m < M, 0 < n < N \tag{5}$$

The above equation is only valid for the cases where the permittivity doesn't change. This is not the end of the story. We've to account for the potential at the fluid air interface,where the permittivities change on either side.
So for that we include this equation which computes the potential at the interface.

$$\phi_{k,j} = \frac{\epsilon_r \phi_{k-1,j} + \phi_{k+1,j}}{1+\epsilon_r} m = k, 0 < n < N \tag{6}$$

- Thus, the potential at any point should be the(weighted)average of its neighbours. This is a very general result and the above calculation is just a special case of it. So the solution process is to take each point and replace the potential by the average of its neighbours and keep iterating till the solution converges i.e., the maximum change in elements of ϕ which is denoted by $errors[k]$ in the code ,where 'k' is the no of iteration, is less than some tolerance which is taken as 10^{-9}.

- The tank's sides are grounded while its top is at a potential of 1V. So this has to be included in the function and care must be taken so as to not disturb these values. Moreover since there's an electrode leading to every side , we need not worry about the current boundary conditions like we did in the fifth experiment.

Python Code

Brief description of subsidiary functions

- A matrix(nrow,X) This function takes two inputs, the number of rows and a vector x It then creates and returns a two column matrix hosting 1 in the first column and x in the other.

- Error fitter lstsq(errors,x) This function takes two arguments as well. All it does is finds the best fit for the errors using the least squares algorithm and return that along with the matrix hosting the error vectors in the first column and the iterations in the second.

- cum error(error,N,A,B) This returns the form in which the error is modelled after extrapolating it to infinity.

- find Stop Condn(errors,Niter,error tol,c1) This compares the error to the error tolerance we've given and once the error breaches the tolerance,returns the index of the vector.

Some stuff regarding the Laplace Solver function given below:

- The error in phi is extrapolated to infinity and modeled as an exponential.This is used in the above function cum error.

- Note that we don't pass to the function since it is a redundant condition already determined by M and N.

```
#Q: Create a function that solves the laplace equation and obtains phi(m,n)

#Defining all the arguments of the function
M=20 #no of nodes along x axis
N=40  #no of nodes along y axis, taken in this manner to ensure there's a no
delta=pow(10,-9) #error tolerance/desired accuracy
Niter=2500  # max no of iterations to complete
k=20 #value of height of the fluid
Er=2 #The relative permittivity or the dielectric constant of the fluid


# function to create Matrix for finding the Best fit using lstsq
# with no_of rows, 2 columns by default  and vector x as arguments

def A_matrix(nrow, x):
    A = np.zeros((nrow, 2))  # initializing the A matrix
    A[:, 0] = 1
    A[:, 1] = x
    return A

# function to find best fit for errors using lstsq
def Error_fitter_lstsq(errors, x):
    A = A_matrix(len(errors), x)
    return A, np.linalg.lstsq(A, np.log(errors))[0]

#function calculating cumulative error
def cum_error(error, N, A, B):
    return -(A/B)*exp(B*(N+0.5)) #the error is modeled in this form

#finds the stopping condition of the iterations based on tolerance value
def find_Stop_Condn(errors, Niter, error_tol,c1):
    cumerror = []
```

```
    for n in range(1, Niter):
        cumerror.append(cum_error(errors[n], n, exp(c1[0]), c1[1])) #creates
        if(cumerror[n-1] <= error_tol):  #compares the cumulative error and t
            return n   #deduces the number of iterations based on tolerance

def Laplace_solver(M,N,k,delta,Niter):
    y=linspace(0,20,N)
    x=linspace(0,10,M)
    phi=np.zeros((N,M))
    X,Y=np.meshgrid(x,y)# creates a coordinate system
    phi[39,:]=1#making the top of the tank have 1V

    errors = zeros(Niter)  # initialise error array to zeros
    iterations = []  # array from 0 to Niter used for findind lstsq

    for i in range(Niter):
        #copy the old phi
        oldphi=phi.copy()
        #updating the full phi matrix
        phi[1:-1,1:-1]=0.25 *(phi[1:-1, 0:-2]+phi[1:-1, 2:]+phi[0:-2, 1:-1]+ph
        #selectively doing it for n=k alone so it overrides the previous dat
        phi[k,1:-1]=(Er*phi[k-1,1:-1]+phi[k+1,1:-1])/(1+Er)
        #Appending errors for each iteration
        errors[i] = (abs(phi-oldphi)).max()
        iterations.append(i)

    #The corresponding least square fitting of the errors
    c = Error_fitter_lstsq(errors, iterations)[1]

    #error tolerance
    error_tol = delta
    Nstop = find_Stop_Condn(errors, Niter, error_tol,c)
    return Nstop,phi,errors
```

Now for the code that contains the observations and the plots obtained. These are done from the outputs of the Laplace Solver function given above.

```
iterations=np.linspace(1,Niter,Niter) #an array containing the number of iter
#initialization
errors=np.zeros(Niter)
phi=np.zeros((N,M))
Nstop=Laplace_solver(M,N,k,delta,Niter)[0]
y=linspace(0,20,N)
x=linspace(0,10,M)
X,Y=np.meshgrid(x,y)# creates a coordinate system or a grid of sorts
phi=Laplace_solver(M,N,k,delta,Niter)[1]
errors=Laplace_solver(M,N,k,delta,Niter)[2]
print("The number of iterations executed =",Nstop)

###### Contour Plot of the Potential:
plt.contourf(X, Y, phi, cmap=cm.jet)#cmap gives the colour(cmap.jet=default)
plt.title("Figure 1: Contour plot of Updated Potential $\phi$")
plt.colorbar()
plt.xlabel("x ->")
plt.ylabel("y ->")
plt.grid()
plt.show()

#plots of the errors: This will explain why we modelled the errors as an exp
plt.semilogy(iterations[0::50], errors[0::50],'go', markersize=8, label="Origi
plt.semilogy(iterations[0::50], errors[0::50], markersize=8, label="Original E
plt.legend()
plt.title("Figure 2a : Error Vs No of iterations (Semilog)")
plt.xlabel("Niter ->")
plt.ylabel("Error ->")
plt.grid()
plt.show()

plt.loglog(iterations[0::50], errors[0::50],'go', markersize=8, label="Origina
plt.loglog(iterations[0::50], errors[0::50], markersize=8, label="Original Err
```

```
plt.legend()
plt.title("Figure 2b : Error Vs No of iterations (Loglog)")
plt.xlabel("Niter ->")
plt.ylabel("Error ->")
plt.grid()
plt.show()
```

Now for the plots.

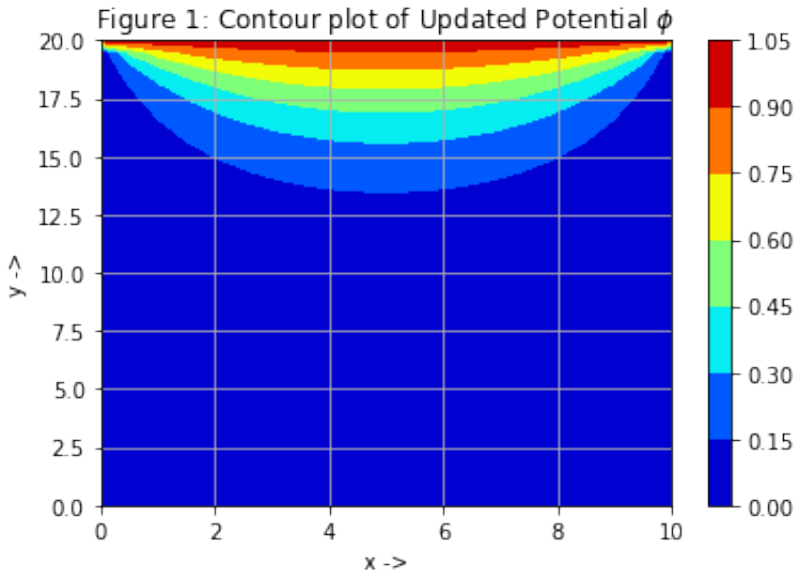

Figure 2: Contour plot of initial potential

Results and Discussion :

- The potential dies down as it progresses further from the plate of potential 1V.And it looks like its clustered around the two opposite conducting plates of the "capacitor". This is a trend that is expected.

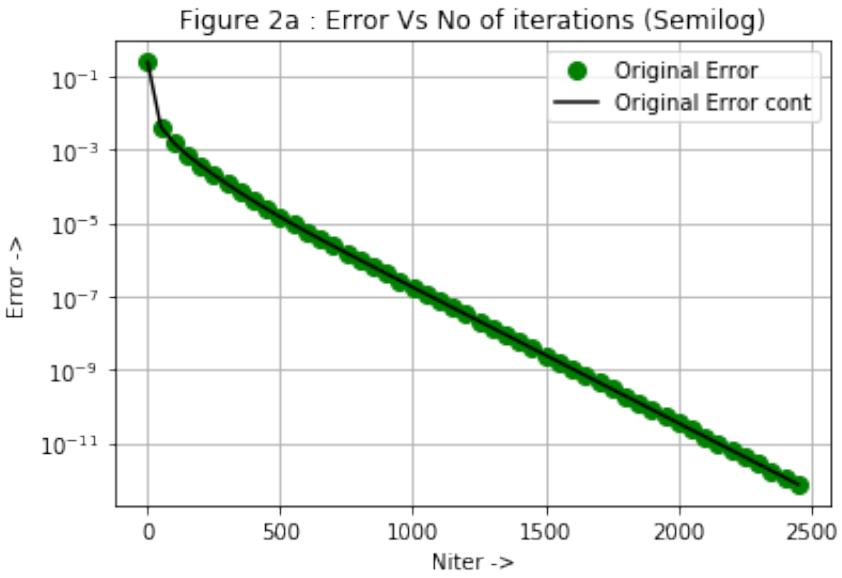

Figure 3: Semilog plot of error

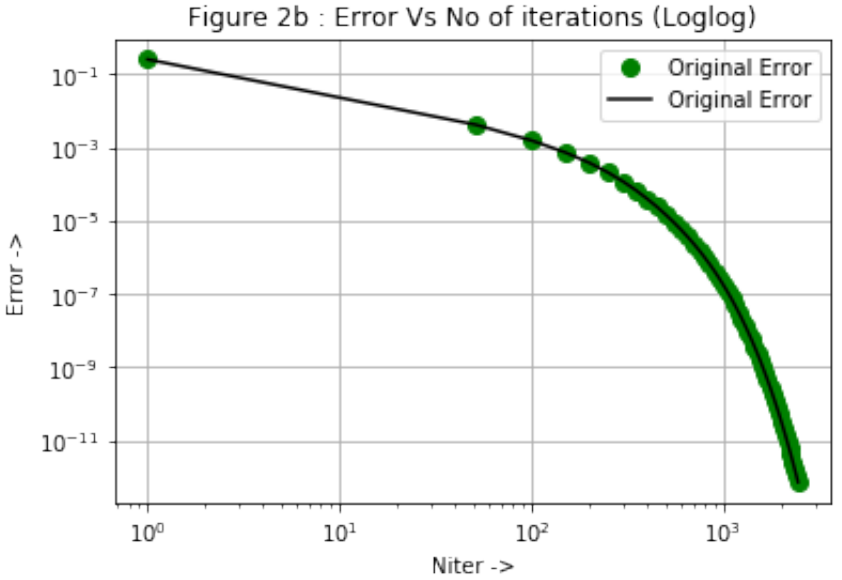

Figure 4: Exponential plot of error

Results and Discussion :

- The error plots too hold true to what we predicted. Especially the linear semilog plot tells us that the error is exponential in nature.This

reiterates that our model is right as well.

- The output for the number of iterations is given below The number of iterations executed = 2140

0.2.2 Parallelizing the computation

The Question

How will you parallelize the computation? The m = k row has to be handled differently. Explain why your algorithm is efficient.

Explanation

Consider the below segment from the for loop in the above Laplace Solver function.

```
#updating the full phi matrix
phi[1:-1,1:-1]=0.25 *(phi[1:-1, 0:-2]+phi[1:-1, 2:]+phi[0:-2, 1:-1]+
phi[2:, 1:-1])
#selectively doing it for n=k alone so it overrides the previous data
phi[k,1:-1]=(Er*phi[k-1,1:-1]+phi[k+1,1:-1])/(1+Er)
```

If we were to do this the conventional way or the way in an other language like C++, the code will be written as below.

```
for(i=1 ; i<N-1 ; i++ ){ // interior nodes
for(j=1 ; j<M-1 ; j++ ){ // interior nodes
phinew[i,j]=0.25*(phi[i,j-1]+phi[i,j+1]
+phi[i-1,j]+phi[i+1,j]);
phinew[k,j]=(Er*phi[k-1,j]+phi[k+1,j])/(1+Er)
```

- The code becomes very concise in the first case and whatever is being done in a nested for loop in the latter is done in one step.
- Essentially we parallelize the above code by considering the concerned slice of the matrix and then updating all these values at one go.
- Note that we do this in two steps. First update all the values of phi(except the boundaries) in a single step with the general equation. Then apply the specific equation at the fluid air interface to ensure that this overwrites the output of the above equation. Therefore, this process is indeed treated differently for m=k.

- The code becomes more efficient as on first glance, the time complexity reduces from $O(n^2)$to O(n). This improves the performance of the algorithm in a profound manner.

0.2.3 Computing Ex and Ey at the center of the mesh cells

The Question

For h = 0.5, compute Ex and Ey on the at (m+0.5,n+0.5), i.e., at the centre of mesh cells. Show that Dn is continuous at m = k.

Theory

We know that:

$$E = -\nabla\phi \tag{7}$$

From this, splitting the $\nabla\phi$ into partial derivatives,

$$E_x = -\frac{\partial\phi}{\partial x} \tag{8}$$

$$E_y = -\frac{\partial\phi}{\partial y} \tag{9}$$

- To use this as an algorithm we use these equations as follows by an approximation of the partial derivative.

$$E_{x,ij} = \frac{1}{2}(\phi_{i,j-1} - \phi_{i,j+1}) \tag{10}$$

$$E_{y,ij} = \frac{1}{2}(\phi_{i-1,j} - \phi_{i+1,j}) \tag{11}$$

- We use the above set of equations to find the value of Ex and Ey
- Similarly, in order to find the Electric fields at the centre of the mesh, we use the below algorithm.

$$Ex_{m+0.5,n+0.5} = \frac{1}{4}(Ex_{m,n} + Ex_{m+1,n} + Ex_{m,n+1} + Ex_{m+1,n+1}) \tag{12}$$

$$Ey_{m+0.5,n+0.5} = \frac{1}{4}(Ey_{m,n} + Ey_{m+1,n} + Ey_{m,n+1} + Ey_{m+1,n+1} \tag{13}$$

Now lets come to the second part of the problem.

- Let us look at the conservation of Dn at the air fluid interface.
- Using the boundary conditions that come from Maxwell's equations,we get

$$\hat{n}.(\vec{D_2} - \vec{D_1}) = \rho_s \tag{14}$$

- Now we can assume that $\rho_s = 0$ since all the charge will be accumulated on the conductor sheets.
  Therefore,we get

$$\frac{D_2}{D_1} = 1 \tag{15}$$

All this is shown in the below python code

Python Code

```
#Question F
#For h/Ly=0.5 compute Ex and Ey for m+0.5 and n+0.5  and prove that Dn is co
#Now for a conductor like the tank,the volume charge density will be zero in
#We could use this to our advantage to calculate the Ex and Ey at the centre
def field_finder(phi,N,M):
    Ex = np.zeros((N,M))
    Ey = np.zeros((N,M))
    #finding the E field components using approximation of partial derivativ
    Ex[1:-1,1:-1]= 0.5*(phi[1:-1, 0:-2] - phi[1:-1, 2:])
    Ey[1:-1,1:-1]= 0.5*(phi[0:-2, 1:-1] - phi[2:, 1:-1])
    return Ex,Ey

#finding the electric fields as an average of its surrounding electric field
def centred_field_finder(Ex,Ey):
    Exmn=np.zeros((N,M))
    Eymn=np.zeros((N,M))
    Exmn=0.25*(Ex[:-1,:-1]+Ex[:-1,1:]+Ex[1:,:-1]+Ex[1:,1:])
    Eymn=0.25*(Ey[:-1,:-1]+Ey[:-1,1:]+Ey[1:,:-1]+Ey[1:,1:])
    return Exmn,Eymn

#Quiver Plot of the Electric Field
Ex,Ey=field_finder(phi,N,M)
```

```
Exmn,Eymn=centred_field_finder(Ex,Ey)
cs=plt.contour(2*X,2*Y,phi)  #to create the contour graph
plt.clabel(cs, cs.levels[:5], inline=1, fontsize=10) #this is to write values
plt.quiver(Exmn,Eymn,label="electric field")  #quiver plot with the Electric
plt.xlabel('x ->')
plt.ylabel('y ->')
plt.legend(loc='lower center')
plt.grid()
plt.title("Figure 4:The Vector plot of the Electric Field")
plt.show()
#now lets consider Dn right above and below the interface. For an ideal case
print("The ratio of Dn on either side is",
np.around(Eymn[k].mean()/(Er*Eymn[k-1].mean()),4))
```

The obtained quiver plot of the Electric Field is given below.

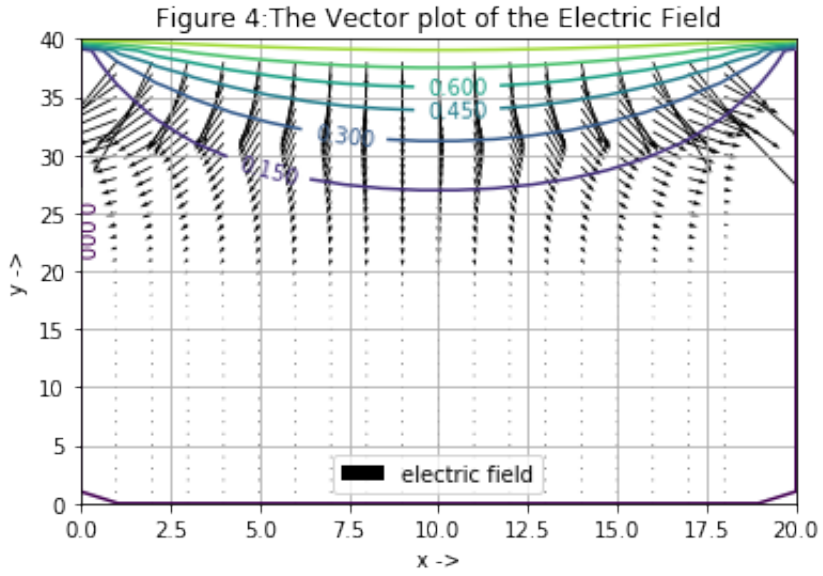

Figure 5: Quiver Plot Of Electric Field

Results and Discussion

- We can observe that the arrows point towards the direction of decrease of ϕ or -ϕ. Therefore, this just reiterates that E is indeed -ϕ.
- We can also notice that the arrows get smaller as the difference in ϕ gets smaller and smaller.
- Interestingly, the Electric Field behaviour is not identical to that of an ideal capacitor.
- The output we get for the ratios of the normal components of D at the interface, ie $\frac{Dn_{above}}{Dn_{below}}$ is,
  The ratio of Dn on either side is 0.746.
- This is not exactly 1, but this is reasonably accurate for us to tell that Dn is consistent across the interface.

0.2.4 Change in angle of Electric Field at interface

Question

Obtain the change in angle of the Electric field at m = k. Does this agree with Snell's Law? Should it? Explain.

Explanation

First let us look at the respective code and output.

```
theta1=np.zeros(9)
theta2=np.zeros(9)
for i in range (1,9):   # as the angle is approx symmetric with respect
    theta1[i]=math.atan(Ey[k+1,i]/Ex[k+1,i])
    theta2[i]=math.atan(Ey[k-1,i]/Ex[k-1,i])
print("The value of theta 1 is",np.around(theta1.mean(),5)) #angle incide
print("The value of theta 2 is",np.around(theta2.mean(),5)) #angle transm
```

- The output we get is given below.
  The value of theta 1 is 0.66638
  The value of theta 2 is 0.66597
  Note that the angles are in radians.

- From this, θ_1-θ_2= 0.00041 radians or 0.0235 degrees

- Lets see if this agrees with Snell's law.
  Snell's law is given as

$$n_1 sin(\theta_1) = n_2 sin(\theta_2) \tag{16}$$

  Where n is proportional to $\sqrt{\epsilon_r}$
  Therefore, for air and the fluid, we can formulate the equations as

$$sin(\theta_1) = \sqrt{\epsilon_r} sin(\theta_2) \tag{17}$$

- Plugging in the values into this equation we get 0.618 for LHS and 0.873 for the RHS.
  Note, this is done for the average value of half the angles.
  If we do this for every individual point we'll get different answers.
  From this it appears as though Snell's law isn't valid.

- Snell's law is not valid as depicted by the previous equation. This is because, Snell's law is only applicable to electro-magnetic waves such as light waves. Our field distribution is not a wave and isn't time varying as well(not necessary for validity) and therefore we can't apply Snell's law for this.

0.2.5 Charge distribution vs h

The Question

Run the code for h/Ly = 0.1,0.2,...,0.9 and obtain Qtop vs h and Qfluid vs h where Qtop is the charge on the top plate held at 1 volt, and Qfluid is the charge on the portion of the wall touching the dielectric fluid. Plot the same. Is it linear? Should it be? Explain
Note: Q is charge per unit length and the third dimension can be assumed to be uniform.

Theory

- For conductors, the surface charge density can be found as follows using the boundary conditions.

$$\hat{n}.(\vec{D_2} - \vec{D_1}) = \rho_s \tag{18}$$

- Now, since the third dimension is uniform we can safely say that $\rho_s = \sigma L$, where L is a constant arising from the third dimension.

- Also, recollecting that $D = \epsilon E$, we can rewrite the above equation as

$$\hat{n}.(\epsilon_{r2}\vec{E_2} - \epsilon_{r2}\vec{E_1}) = \frac{\sigma}{\epsilon_0 L} \tag{19}$$

- Noting that $Q = \sigma AL$ and incorporating that in the equation we get.

$$\hat{n}.(\epsilon_{r2}\vec{E_2} - \epsilon_{r2}\vec{E_1}) = \frac{Q}{A\epsilon_0} \tag{20}$$

- $A\epsilon_0$ is just a uniform constant and therefore for simplicity, let's assume that it's 1. If it isn't, all we need to do is scale the graph appropriately. Therefore our final equation is:

$$\hat{n}.(\epsilon_{r2}\vec{E_2} - \epsilon_{r2}\vec{E_1}) = Q \tag{21}$$

- Now doing this specifically for Qtop and Qfluid we get,

$$E_{air} - E_{cond} = Qtop \tag{22}$$

We know that, $E_{air} = 0$
Therefore,

$$-E_{cond} = Qtop \tag{23}$$

Similarly,

$$E_{air} - \epsilon_r E_{condleft} + E_{air} - \epsilon_r E_{condright} + E_{air} - \epsilon_r E_{condbottom} = Qfluid \tag{24}$$

or

$$\epsilon_r(-E_{condleft} - E_{condright} - E_{condbottom}) = Qfluid \tag{25}$$

- These terms will be clear by following the below image. Note:The outward normal is taken as $\hat{n}$ here.
  Also,only the portion of the conductors touching the fluid is considered in the second case.

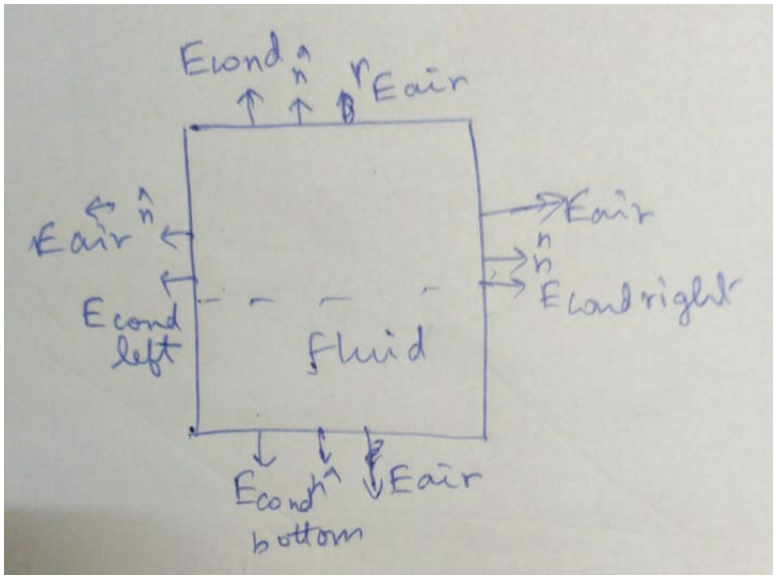

Figure 6: Drawing with fields

Python Code

The code for this part is given below.

```
# Question E
#Run code for different values of h and plot Qtop and Qfluid vs h
# Use the expression Q=CV to find Qtop and Qfluid
# We can get the value of V from phi so all that's required is C
#using for loop to test this across multiple values of k
Qtop=zeros(9)
Qfluid=zeros(9)
iterations_1=[]
for j in range(2,20,2):#k starts from 2cm and advances to 18cm in steps of 2c
    phi_1=Laplace_solver(M,N,2*j,delta,Niter)[1] #2j since everything is scal
    Ex,Ey=field_finder(phi_1,N,M)
    Ex_1,Ey_1=centred_field_finder(Ex,Ey)
    Qtop[int(j/2-1)]=-(Ey_1[-2]).mean()
    Qfluid[int(j/2-1)]= Er*(-(Ex_1[:2*j,2]).mean()-(Ex_1[:2*j,-2]).mean()-(Ey_
    iterations_1.append(j)

#Plotting Q vs h
plt.plot(iterations_1,Qtop,'go-',label="Qtop")
plt.plot(iterations_1,Qfluid,'bo-',label='Qfluid')
plt.title("Figure 3: The relationship of Q vs h")
plt.legend()
plt.xlabel("h->")
plt.ylabel("Q->")
plt.grid()
plt.show()
```

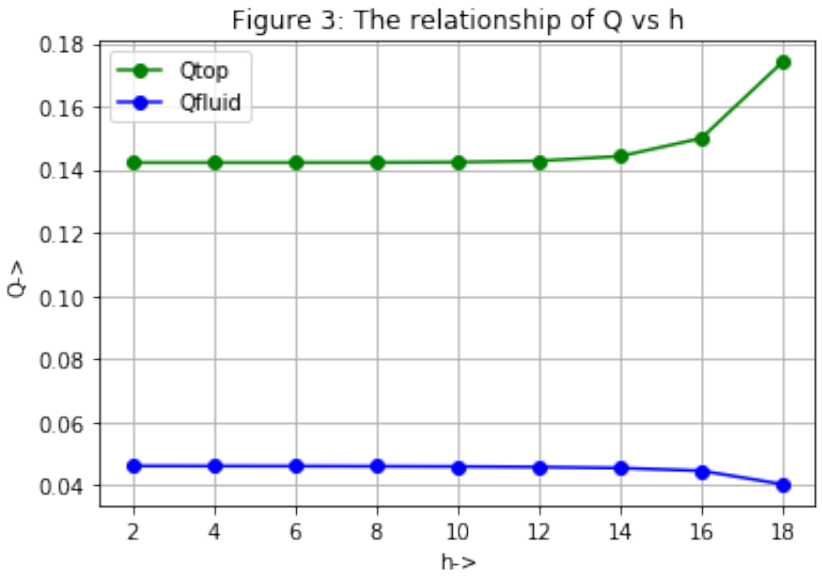

Figure 7: Drawing with fields

Results and Discussion

- Both the charges don't show a linear variation with respect to h. This is observed from the graph.
- There is no reason as to why these relationships should be linear. Even for an ideal capacitor with a dielectric portion, there is an inverse dependance of h on charge.

0.2.6 Develop Algorithm to find h from resonant frequency

The Question

Develop your algorithm to determine h from the observed resonant frequency. Derive all necessary equations.

Theory

As established before, the given circuit mimics an RLC network. This RLC network oscillates with a resonant frequency ω_0. This frequency depends on L and C as follows.

$$\omega_0 = \frac{1}{\sqrt{LC}} \tag{26}$$

So the way we progress for this is as follows.

- Firstly, by using the above formula we get the value of C in terms of L and ω_0.
- Now use the below formula.

$$Q = CV \tag{27}$$

  As far as V is concerned, we can take Vtop since its constantly maintained at 1V.
  Therefore from this, $Qtop = C$.

- We now compare this value of C to values from the Qtop vs h plot(after scaling it appropriately as per the given values) and then find the corresponding h. I'm doing these with the numpy functions polyfit and poly1d (which was taught in class).Polyfit models h as an nth degree polynomial of Q. Poly1d is then used to create this polynomial so that the h value can be found based on the error tolerance the user wants.

The Algorithm

The pseudocode is given below.

```
initialise errornew and error old and hvalnew with zero
and have some error tolerance and n with 1

iterate while error is less than tolerance:
{poly_coeff=np.polyfit(Qtop,iterations_1,n)
h=np.poly1d(poly_coeff)
hvalold=h(C)
error=hvalnew-hvalold
hvalnew=hvalold
n=n+1}
print(hvalnew)
```