Final Exam: Applied Programming Lab

Srivenkat A (EE18B038)

July 30, 2020

1 Introduction

The aim is to analyse the potential and electric field distribution in a metal tank with a partially filled dielectric. The task setup involves a RLC circuit connected to an alternating current source, where the metal tank acts as the capacitor.

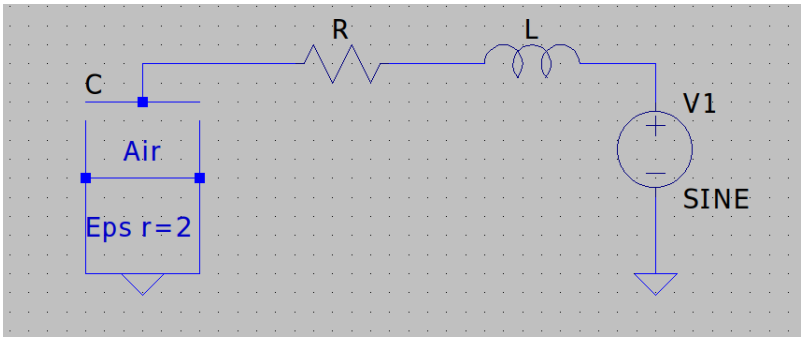


Figure 1: Circuit Setup

2 Laplace Equations

Derived from Maxwell's equations, the potential distribution can be numerically obtained by solving the Laplace's equation. The Laplace equation in 2 dimensions can be written as:

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

So, the potential distribution inside the metal tank follows this equation and numerically solving it gives the distribution in the uniform region as:

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

But, at the dielectric interface, to ensure continuity of normal component of electric-displacement D,

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

The metal tank has dimensions Lx = 10cm and Ly = 20cm. We numerically solve the above equations by assuming a coordinate mesh or grid across the cross-section of the tank in the x-y plane and iteratively enforcing the above condition until the distribution function saturates.

3 RLC circuit Analysis

For the series RLC circuit, the resonant frequency is known to be

$$w_0 = \frac{1}{\sqrt{LC}} \tag{1}$$

For a parallel plate capacitor with cross section area A and plate separation d,

$$C = \frac{A\epsilon_0}{d} \tag{2}$$

Here, for the 2 capacitors in series have d, ϵ values as d = h, ϵ_r=1 and d = L_y -h, ϵ_r=2. So, the net capacitance comes out to be:

$$C = \frac{2A\epsilon_0}{2L-h} \tag{3}$$

So, the resonant frequency and height of dielectric fluid have the relation:

$$w_0 = \frac{\sqrt{L_y - \frac{h}{2}}}{\sqrt{L_y A \epsilon_0}} \tag{4}$$

Since the depth of the tank(along z-axis) is not given, A cannot be calculated. So, given 1 combination of w_0 and h values, the constants can be calculated and then, for any w_0 value, the corresponding height h can be calculated.

4 Vectorized Computation

The phi matrix is a M x N grid representing the cross sectional area of the tank. Numerically solving the Laplace Equation involves iterating over the phi matrix and enforcing the constraints on each element.

To iteratively update the elements of the potential phi matrix with the average of the nearby 4 elements, we vectorize the computations to parallely compute the average for multiple elements at once.

Ideally, first we need to update the top half of the matrix , then update the row where m=k, i.e., at the dielectric interface and then, update the bottom half of the matrix.

This involves 2 independent vector computations, for the top and bottom of the phi matrix.

Instead, We can update over the whole matrix in one go and then update the row corresponding to m=k. Since the effect of that k'th row gets carried onto the other rows over successive iterations, there will only be a delay of 1 iteration in this approach.

Since the second approach updates the entire matrix in one go, with the help of larger vectors, it is computationally more efficient. For N-iterations in the range of 1000s, both the approaches give the same final distribution.

So, the code for this task will implement the 2nd approach everywhere.

5 Solving for Potential Distribution

The python implementation for numerically solving Laplace equations by the above mentioned approach is given below:

```
def SolveLaplacian(k,Niter,delta=0.1,M=100,N=200,acc=1e-6):
    x = np.linspace(0,Lx,M)
    y = np.linspace(0,Ly,N)
    Y,X = np.meshgrid(y,x)

    phi = np.ones(shape=[M,N])
    phi[:,N-1] = 1.0
    phi[:,0] = 0.0
    phi[0,:] = 0.0
    phi[M-1,:] = 0.0
    err = []
    cum_err = []
    for i in range(Niter):
        oldphi = phi.copy()
        phi[1:-1,1:-1] = 0.25*(oldphi[0:-2,1:-1]
                               + oldphi[2:,1:-1]
                               + oldphi[1:-1,2:]
                               + oldphi[1:-1,0:-2] )
        phi[1:-1,k] = (Er*phi[1:-1,k-1] + phi[1:-1,k+1])/(1+Er)

        err.append(abs(phi-oldphi).max())
        if(err[-1]<acc):
            break
        cum_err.append(np.sum(err))

    Ex = np.zeros(shape=[M,N])
    Ey = np.zeros(shape=[M,N])
    Ex[1:-1,:] = (phi[2:,:]-phi[:-2,:])/delta
    Ey[:,1:-1] = (phi[:,2:]-phi[:,:-2])/delta

    return(X,Y,phi,Ex,Ey,err,i)
```

From the above function, the obtained potential distribution has the form:

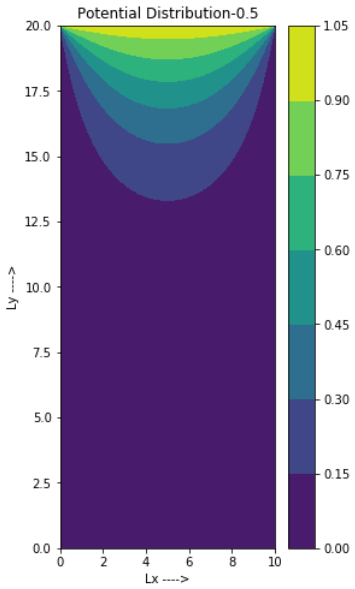


Figure 2: Potential Distribution

From the above function, the obtained electric field distribution has the form:

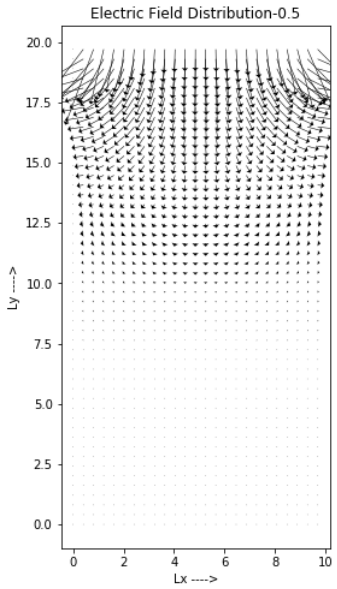


Figure 3: Field Distribution

The observed plots are as expected, where the potential gradually decreases from 1V at the top to 0V at the bottom. Similarly, The electric field are strong and downward facing at the top, directed into the capacitor.

While iteratively updating the phi matrix, we update the error vector with the max change occurred at any coordinate in the phi matrix.

On plotting the error,

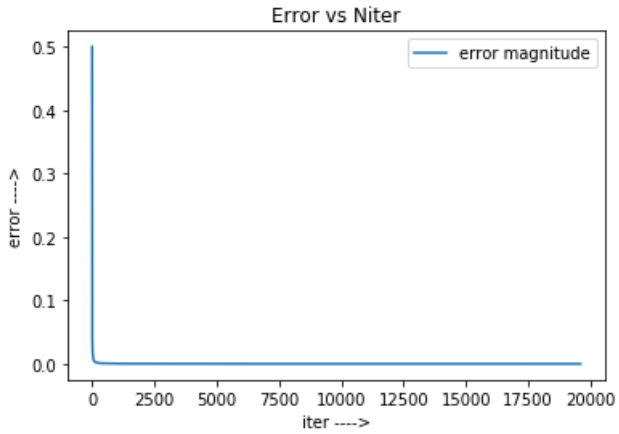


Figure 4: Error plot

To get a better trend of how error decreases with Niter, we plot a semilog plot. Since the error is observed to exponentially reduce with the no. of

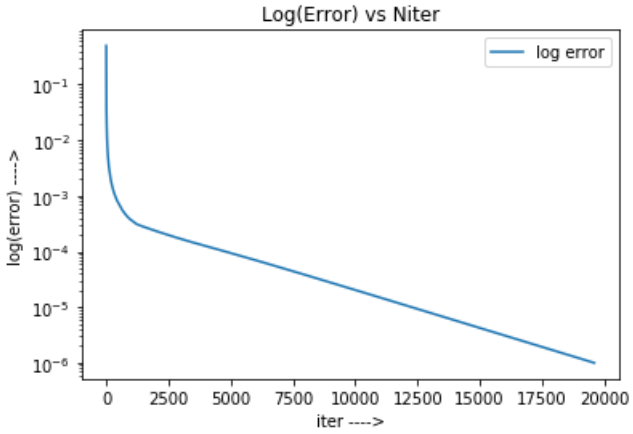


Figure 5: Log Error plot

iterations, we try to model the error function as

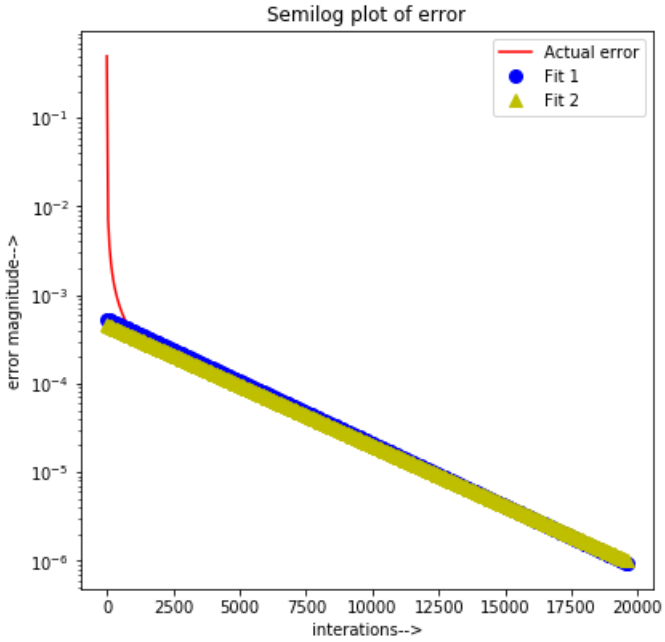
$$y = Ae^{Bx}$$

$$logy = logA + Bx$$

$logA$ and B can be estimated using the least squares method. The following code block accomplishes the same :

```
log_y_mat = np.log(np.transpose(err))        #modelling the
    error in a exponential function
x_mat = np.c_[np.arange(0,Nover+1),np.ones(shape = [Nover+1,1])
    ]
fit1 = sp.lstsq(x_mat,log_y_mat)[0]      #lstsq fit for whole
    err array
fit2 = sp.lstsq(x_mat[500:-1],log_y_mat[500:-1])[0]
```

Fit1 is the line obtained by fitting the entire array while Fit2 is obtained for the array beyond 500 iterations. Plotting them in a semilog plot we get:



where the red line is the actual fit, the circles is the exponential fit for higher iterations and the triangle marker is an exponential fit for for the entire set. It can be noticed that they almost coincide. So, based on the function, we can predict that, as Niter goes to infinity, error gets to 0.

6 Analysing Accumulated Charge

Since there is a potential difference between the sides of the tank, charge gets accumulated on the walls of the tank and the dielectric. The charge depends on the capacitance offered by the tank as $\mathbf{Q = CV}$. Here, the

capacitance of the tank is known to vary as

$$C = \frac{2A\epsilon_0}{2L - h} \tag{5}$$

So, the charge accumulated also varies with h as:

$$Q \propto \frac{1}{L - \frac{h}{2}} \tag{6}$$

To verify this equation, the charge accumulated was calculated for different heights of dielectric fluid and Q vs h was plotted. To calculate the accumulated charge Q, Gauss law is used.

Since the walls and top of the tank are made of conductors, they can only sustain a normal component of electric field. So, the charge accumulated can be calculated as:

$$Q_{top} = \sum_{n=N, 0<m<M} E_{n,top} \times delta \tag{7}$$

$$Q_{fluid} = \sum_{0<n<k, m=0,M} E_{m,walls} \times \epsilon_r \times delta \tag{8}$$

where delta is the length of a unit square in the grid and Q_{top} and Q_{fluid} are the charges per unit depth accumulated on the lid and on the walls in contact with the dielectric fluid respectively.

The following codes are used to calculate the accumulated charge:

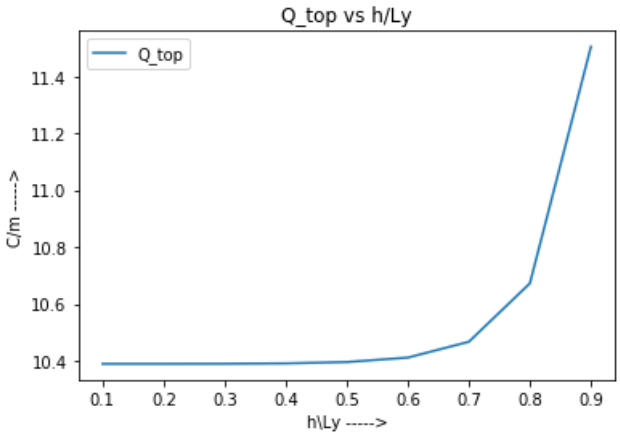
```
#units: 100(cm in E)*10^(-2) = C/m
E_top = Ey[:,-2]
Q_top = np.sum(E_top)*delta

E_bottom = -Ey[:,1]
Q_bottom = 2*np.sum(E_bottom)*delta

E_side = (Ex[-2,:] - Ex[1,:])
Q_side_fluid = 2*np.sum(E_side[0:k-1])*delta
Q_side_air = np.sum(E_side[k:-1])*delta

Q_fluid = Q_bottom + Q_side_fluid
```

The obtained plots for $Qtop$ and Q_{fluid} follow the same hyperbolic relation with h. They are not linear and increase with Q increases with an increasing slope as h increases.



7 Continuity of D

From Maxwells laws, the normal component of electric displacement, D should be continuous across the dielectric interface, i.e., at m=k.

Analysing the tank when it is half-filled with dielectric fluid, i.e., h/Ly = 0.5:

Ex and Ey are calculated by a double ended differential using the following code.

```
Ex[1:-1,:] = (phi[2:,:]-phi[:-2,:])/(2*delta)
Ey[:,1:-1] = (phi[:,2:]-phi[:,:-2])/(2*delta)
```

The value of Ex the middle is found to be: Ex = -0.0254296 V/m
Ey = 1.74099 V/m

To verify continuity of D, we calculate the normal component of the electric fields, above and below the dielectric and find their ratio.

```
En_air = Ey[1:-1,k+1]
En_di = Ey[1:-1,k-1]

E_ratio = En_air/En_di
print(E_ratio.mean(),E_ratio.var())
```

We find that $\frac{E_{air}}{E_{di}} = 2.04$ so, the below equation holds.

$$E_{di} \times \epsilon_o \epsilon_r = Eair \times \epsilon_o \tag{9}$$

So, it is verified that the normal component of D is continuous at the interface.

8 Bending of Electric Field

At the dielectric interface, the angle of incidence and angle of refraction are calculated as:

```
theta = np.arctan(Ex/Ey)
```

To analyse the change in angle of the electric field, we plot the ratios sine(I)/sine(R) and tan(I)/tan(R).

The ratios along the whole inteface are calculated as:

```
sine_ratio = np.sin(theta[:,k+1])/np.sin(theta[:,k-1])
```

On calculating the sin(i)/sin(r) value on the interface, across cross section, we get the plot as:

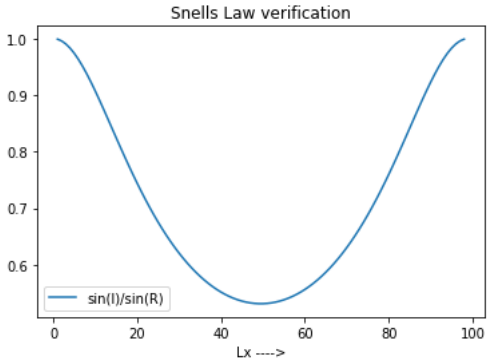


Figure 8: Snell's Law verification

So, it is observed that, at the centre of tank, Snell's law holds, but at the the edges of the tank, due to to the accumulated surface charges, Snell's

law is not valid. To identify the amount of bending of the electric field, we plot the change in incident angle at the interface.

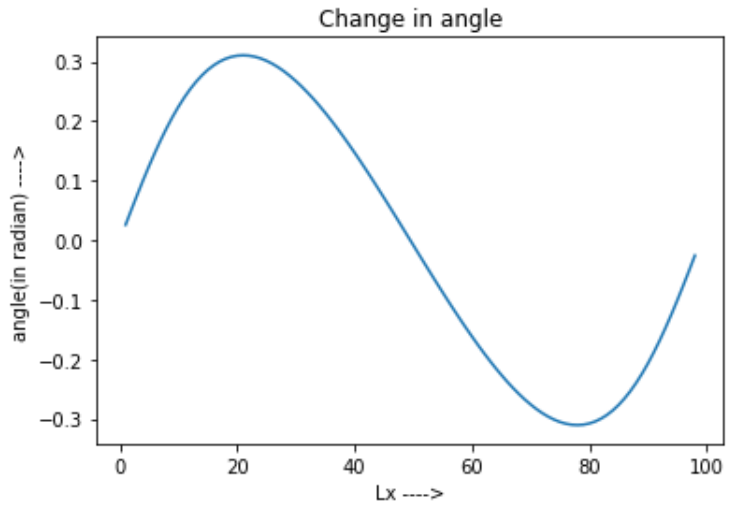


Figure 9: Change in Angle of E-field

At the edges and at the centre, the rays are incident in a normal direction. So, the ray doesn't get refracted much. But, in the surrounding regions, the ray refracts by different amounts due to varying effects of the surface charges from the walls. So, the change in angle varies from a maximum of $\pi/10$ to 0, as observed from the plot.

But, still, as the normal component of electric displacement is constant across the interface,

$$\frac{tan\theta_1}{tan\theta_2} = \frac{\epsilon_1}{\epsilon_2} \tag{10}$$

As we have calculated the ratio o tan thetas,

```
tan_ratio = np.tan(theta[:,k+1])/np.tan(theta[:,k-1])
print("tan(I)/tan(R) mean is {0}, variance is {1}".format(
    tan_ratio[1:-1].mean(),tan_ratio[1:-1].var()))
```

it is observed that the ratio has a constant value of 0.52.

9 Code

```
"""
@author: srivenkat
"""

import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg as sp
```

```
8
9  #pre-defined constants
10 Lx = 10
11 Ly = 20
12 Er = 2
13
14 #user-defined constants
15 delta = 0.1    #resolution of grid
16 Niter = 30000    #No. of iterations
17
18 M = int(Lx/delta)    #No. of nodes parallel to x
19 N = int(Ly/delta)    #No. of nodes parallel to y
20
21 def SolveLaplacian(k,Niter,delta=0.1,M=100,N=200,acc=1e-6):
22     x = np.linspace(0,Lx,M)
23     y = np.linspace(0,Ly,N)
24     Y,X = np.meshgrid(y,x)   #Taking a grid across the tank
25
26     phi = np.ones(shape=[M,N])   #Defining a initial potential
       distribution
27
28     #Enforcing boundary conditions
29     phi[:,N-1] = 1.0
30     phi[:,0] = 0.0
31     phi[0,:] = 0.0
32     phi[M-1,:] = 0.0
33
34     #Defining error vectors
35     err = []
36     cum_err = []
37     for i in range(Niter):
38         oldphi = phi.copy()
39         #Enforcing Laplacian Equation at all coordinates
40         phi[1:-1,1:-1] = 0.25*(oldphi[0:-2,1:-1]
41                                + oldphi[2:,1:-1]
42                                + oldphi[1:-1,2:]
43                                + oldphi[1:-1,0:-2] )
44
45         #Updating at the interface separately
46         phi[1:-1,k] = (Er*phi[1:-1,k-1] + phi[1:-1,k+1])/(1+Er)
47
48         #Taking max change as error
49         err.append(abs(phi-oldphi).max())
50         if(err[-1]<acc):
51             break
52         cum_err.append(np.sum(err))
53
54     #Taking a double ended differential for Electric field
55     Ex = np.zeros(shape=[M,N])
56     Ey = np.zeros(shape=[M,N])
57     Ex[1:-1,:] = (phi[2:,:]-phi[:-2,:])/delta
58     Ey[:,1:-1] = (phi[:,2:]-phi[:,:-2])/delta
59
60     return(X,Y,phi,Ex,Ey,err,i)
```

```
61
62 ratio = 0.5    #H/Ly ratio
63 h = ratio*Ly
64 k = int(ratio*N)
65 X,Y,phi,Ex,Ey,err,Nover = SolveLaplacian(k,Niter)
66
67 En_air = Ey[1:-1,k+1]    #Normal compinent in air
68 En_di = Ey[1:-1,k-1]    #Normal compinent in dielectric
69
70 Et_air = Ex[1:-1,k+1]    #Tangential compinent in air
71 Et_di = Ex[1:-1,k-1]    #Tangential compinent in dielectric
72
73 E_ratio = En_air/En_di
74 print("\n E2/E1 mean is {0}, variance is {1}\n ".format(E_ratio
       .mean(),E_ratio.var()))
75
76 theta = np.arctan(Ex/Ey)    #Angle of incidence
77 sine_ratio = np.sin(theta[:,k+1])/np.sin(theta[:,k-1])    #
       Check snell's law
78
79 tan_ratio = np.tan(theta[:,k+1])/np.tan(theta[:,k-1])
80 print("\n tan(I)/tan(R) mean is {0}, variance is {1}\n ".format
       (tan_ratio[1:-1].mean(),tan_ratio[1:-1].var()))
81
82 #Calculate accumulated charge
83 E_top = Ey[:,-2]
84 Q_top = np.sum(E_top)*delta   #units: 100(cm in E)*10^(-2) = C/
       m
85
86 E_bottom = -Ey[:,1]
87 Q_bottom = 2*np.sum(E_bottom)*delta   #units: 100(cm in E)
       *10^(-2) = C/m
88
89 E_side = -(Ex[-2,:] - Ex[1,:])
90 Q_side_fluid = 2*np.sum(E_side[0:k-1])*delta   #units: 100(cm
       in E)*10^(-2) = C/m
91 Q_side_air = np.sum(E_side[k:-1])*delta     #units: 100(cm in E
       )*10^(-2) = C/m
92 Q_fluid = Q_bottom + Q_side_fluid
93
94 #plot phi
95 plt.subplots(figsize=(4,8))
96 plt.title("Potential Distribution-{}".format(ratio))
97 cont = plt.contourf(X,Y,phi)
98 plt.xlabel("Lx ---->")
99 plt.ylabel("Ly ---->")
100 plt.colorbar(cont)
101 plt.show()
102
103 #plot field
104 plt.subplots(figsize=(4,8))
105 plt.title("Electric Field Distribution-{}".format(ratio))
106 skip=(slice(None,None,4),slice(None,None,4))
107 plt.quiver(X[skip],Y[skip],Ex[::-1,:][skip],-Ey[::-1,:][skip],
```

```
    headwidth=10,scale=5)
plt.xlabel("Lx ---->")
plt.ylabel("Ly ---->")
plt.show()

#plot error
plt.title("Error vs Niter")
plt.plot(err,label="error magnitude")
plt.xlabel("iter ---->")
plt.ylabel("error ---->")
plt.legend()
plt.show()

plt.title("Log(Error) vs Niter")
plt.semilogy(err,label="log error")
plt.xlabel("iter ---->")
plt.ylabel("log(error) ---->")
plt.legend()
plt.show()

#Fit a log expression for error
log_y_mat = np.log(np.transpose(err))       #modelling the
    error in a exponential function
x_mat = np.c_[np.arange(0,Nover+1),np.ones(shape = [Nover+1,1])
    ]
fit1 = sp.lstsq(x_mat,log_y_mat)[0]     #lstsq fit for whole
    err array
fit2 = sp.lstsq(x_mat[500:-1],log_y_mat[500:-1])[0]

plt.semilogy(np.arange(0,Nover,50),err[::50],'r',label='Actual
    error')
plt.semilogy(np.arange(0,Nover,50),np.exp(np.dot(x_mat,fit1))
    [::50],'bo',markersize=8,label='Fit 1')
plt.semilogy(np.arange(0,Nover,50),np.exp(np.dot(x_mat,fit2))
    [::50],'y^',markersize=8,label='Fit 2')
plt.xlabel("interations-->")
plt.ylabel("error magnitude-->")
plt.title('Semilog plot of error')
plt.legend()
plt.show()

#plot sin(I)/sin(R)
plt.title("Snells Law verification")
plt.plot(sine_ratio,label="sin(I)/sin(R)")
plt.xlabel("Lx ---->")
plt.legend()
plt.show()

#plot change in angle
plt.plot(theta[:,k-1]-theta[:,k+1])
plt.title("Change in angle")
plt.xlabel("Lx ---->")
plt.ylabel("angle(in radian) ---->")
plt.show()
```