EE2703 Applied Programming Lab - Assignment No 5

Name: Abishek S
Roll Number: EE18B001

March 3, 2020

1 Abstract

The goal of this assignment is the following :

- To plot contour and 3-D plots.
- To solve 2-D Laplace equation in an iterative manner.
- To use vectorized code in Python.
- To observe the variation of error with iteration.

2 Assignment

For DC current, the 2-D Laplace equation is :

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

OR

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

2.1 Setting the parameters and making contour plot of Potential

Importing the standard libraries

```
import pylab as pl
import mpl_toolkits.mplot3d.axes3d as p3
import argparse
import numpy as np
```

```
from scipy.linalg import lstsq
import sys
import matplotlib
```

Using argparse, getting optional inputs for Nx,Ny,radius and Niter from the command line and also defined their default values.

```
parser = argparse.ArgumentParser(description='Defining Parameters')
parser.add_argument('-Nx',dest='Nx',type=int,default=25,help = 'Size along x')
parser.add_argument('-Ny',dest='Ny',type=int,default=25,help = 'Size along y')
parser.add_argument('-r',dest='radius',type=int,default=8,
                                help = 'Radius of central lead')
parser.add_argument('-Ni',dest='Niter',type=int,default=1500,
                                help = 'Number of interations to perform')
args = parser.parse_args()
Nx,Ny,Niter,radius = args.Nx,args.Ny,args.Niter,args.radius
```

We define a plotting function to simplify our code.

```
def PLOT(x,y,fig_no = 0,label_x = r'$\rightarrow$',label_y = r'$\rightarrow$',
        fn = pl.plot,\\arg3 = 'b-',title = "Plot",grids = True,
        cmap = matplotlib.cm.jet,label = ''):
    pl.figure(fig_no)
    pl.grid(grids)
    if fn == pl.contourf:
        fn(x,y,arg3,cmap = cmap)
        pl.colorbar()
    else:
        if label == '':
            fn(x,y,arg3)
        else:
            fn(x,y,arg3,label = label)
    pl.xlabel(label_x)
    pl.ylabel(label_y)
    pl.title(title)
```

We allocate and initialize the potential array.

```
Phi = np.zeros(shape = (Ny,Nx))
x = np.linspace(-0.5,0.5,Nx)
y = np.linspace(-0.5,0.5,Ny)
Y,X = np.meshgrid(y,x)
```

We find the points with in the radius of 0.35 cm from the center in 1cm x 1cm grid and plot the contour of the potential.

```
ii = np.where(X*X + Y*Y <= 0.35*0.35)
Phi[ii] = 1

PLOT(x,y,1,"X-axis","Y-axis",pl.contourf,Phi,"Contour Plot of Potential",
        True,matplotlib.cm.hot)
pl.plot(X[X*X + Y*Y < 0.35*0.35],Y[X*X + Y*Y < 0.35*0.35],'ro',
        label = 'Points with potential 1V')
pl.legend()
pl.show()
```

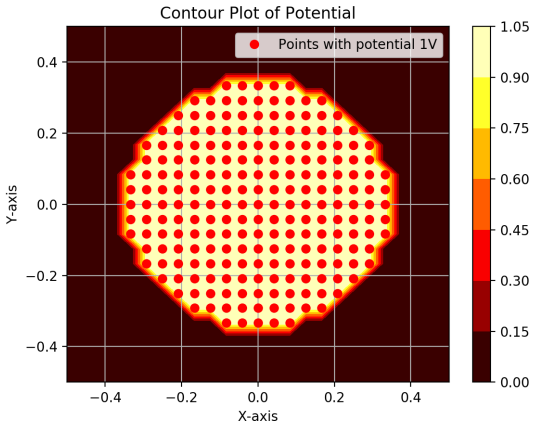

Figure 1: Contour Plot of Potential

2.2 Solving the 2-D Laplace equation

We solve the 2-D laplace equation by updating the potential in many iterations so that it converges. This isn't the best approach, but is the easiest to understand and implement, hence we use it.
We use the below formula for updating the potential in each iteration :

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4} \tag{3}$$

```
errors = np.zeros(Niter)
for k in range(Niter):
    oldphi = Phi.copy()
    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])
    Phi[0,:] = 0
    Phi[1:-1,0] = Phi[1:-1,1]
    Phi[1:-1,-1] = Phi[1:-1,-2]
    Phi[-1,:] = Phi[-2,:]
```

```
    Phi[ii] = 1.0
    errors[k] = (abs(Phi-oldphi)).max()
```

2.3 Plotting the Error

We plot the error vs iteration number- in loglog anf semilog plots. We notice that the error varies linearly in semilog plot for higher iterations.

```
#Semilog Plot
PLOT(np.arange(1,Niter+1),errors,2,"Iteration Number","Log of Error",
    fn = pl.semilogy,title='Semilogy Plot of Error vs Iteration Number',
    label = 'Line')
pl.semilogy(np.arange(1,Niter+1,30),errors[0:Niter:30],'yo',label = 'Points')
pl.legend()
pl.show()

#Loglog plot
PLOT(np.arange(1,Niter+1),errors,3,"Log of Iteration Number","Log of Error",
    fn = pl.loglog,title='Loglog Plot of Error vs Iteration Number',
    label = 'Line')
pl.loglog(np.arange(1,Niter+1,30),errors[0:Niter:30],'yo',label = 'Points')
pl.legend()
pl.show()

#Semilog plot after 500 iterations
PLOT(np.arange(500,Niter+1),errors[499:],4,"Iteration Number","Log of Error",
    fn = pl.semilogy,title='Semilogy Plot of Error vs Iteration Number
    after 500 iterations',label = 'Line')
pl.legend()
pl.show()
```

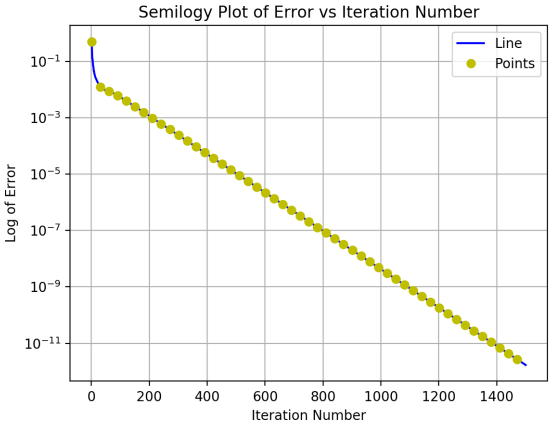

Figure 2: Error vs Iteration - Semilog Plot

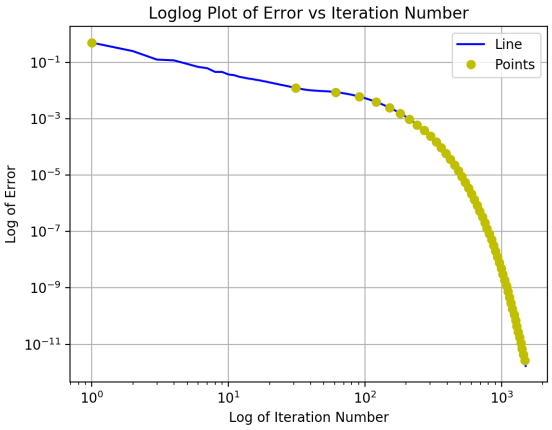

Figure 3: Error vs Iteration - Loglog Plot

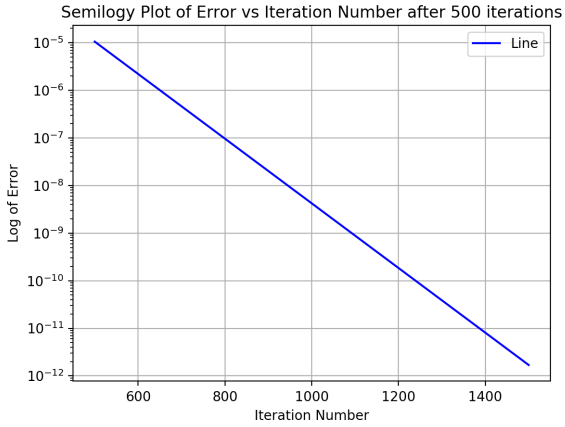

Figure 4: Error vs Iteration for Higher iterations (>= 500) - Semilog Plot

2.4 Obtaining a fitting for the Error and plotting it

We noticed how the error varies linearly with iteration number for higher iterations (>= 500). Thus, we can assume the error (E) to vary as : Ae^{Bk} where **A,B** are parameters we need to figure out, and **k** is the iteration number. We use Least linear fitting to obtain parameters A and B since taking log of the above equation, we obtain :

$$logE = logA + By \tag{4}$$

We obtain two fittings :
Fit 1 - Fitting the nearly linear portion of semilog plot (>= 500) iterations using :

$$\begin{pmatrix} 1 & 500 \\ 1 & 501 \\ ... & ... \\ 1 & 1500 \end{pmatrix} \begin{pmatrix} logA \\ B \end{pmatrix} = \begin{pmatrix} E_{500} \\ E_{501} \\ ... \\ E_{1500} \end{pmatrix} \tag{5}$$

Fit 2 - Fitting the entire error vector (for all iterations) using :

$$\begin{pmatrix} 1 & 1 \\ 1 & 2 \\ ... & ... \\ 1 & 1500 \end{pmatrix} \begin{pmatrix} logA \\ B \end{pmatrix} = \begin{pmatrix} E_{1} \\ E_{2} \\ ... \\ E_{1500} \end{pmatrix} \tag{6}$$

We plot Fit 1 and Fit 2 along with the original error.

```
a,b = lstsq(np.c_[np.ones(Niter-499),np.arange(500,Niter+1)],
    np.log(errors[499:]))[0]
a = np.exp(a)
```

```
print('The values of A and B for which Ae^(Bk) fits
    the error after 500 iterations are:')
print(a,b)
lerr = a * np.exp(b*np.arange(500,Niter+1))

A,B = lstsq(np.c_[np.ones(Niter),np.arange(1,Niter+1)],np.log(errors))[0]
A = np.exp(A)
print('The values of A and B for which Ae^(Bk) fits
    the entire error vector are:')
print(A,B)
err = A * np.exp(B*np.arange(1,Niter+1))

PLOT(np.arange(1,Niter+1),errors,5,"Iteration Number","Log of Error",
    fn = pl.semilogy,arg3 = 'r-',title='Semilogy Plot of Error vs
    Iteration Number',label = 'Original Errors')
pl.semilogy(np.arange(500,Niter+1),lerr,'b-',
    label = 'Linearly fitted error for >500 iterations (Fit 1)')
pl.semilogy(np.arange(1,Niter+1),err,'g-',
    label = 'Linearly fitted error for entire error vector (Fit 2)')
pl.legend()
pl.show()
```

We obtain the parameters **A,B** for FIt 1 and Fit 2 using least squares approach.

```
The values of A and B for which Ae^(Bk) fits the error after 500 iterations are:
0.026454702632631128 -0.015648068091187815
The values of A and B for which Ae^(Bk) fits the entire error vector are:
0.026629204755201308 -0.015655263499125868
```

Figure 5: The parameters obtained for Fit 1 and Fit 2

We can observe how closer the A's and B's are which happens because the log of error varies non linearly with iteration number only at the beginning for a few iterations, as before updating the potential can be initialised with any values.

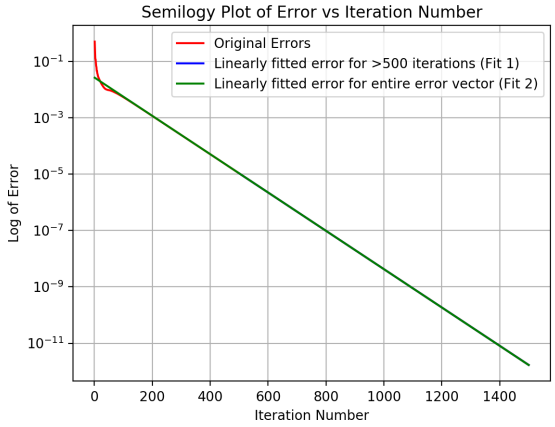

Figure 6: Original and fitted Error vs Iteration - Semilog Plot

2.5 Plotting the obtained Potential

We solved the laplace equation and obtained the potential over the region. We plot it as a 3-D surface plot and contour plot.

```
fig1 = pl.figure(6)
ax = p3.Axes3D(fig1)
pl.title('The 3-D Surface plot of the potential')
surf = ax.plot_surface(Y,X,Phi,rstride = 1,cstride = 1,
    cmap = matplotlib.cm.viridis,linewidth = 0)
cbaxes = fig1.add_axes([0.96, 0.1, 0.03, 0.8])
pl.colorbar(surf,cax = cbaxes,pad = 0.2)
pl.show()

PLOT(x,y,7,"X-axis","Y-axis",pl.contourf,Phi,"Contour Plot of Potential",
    cmap = matplotlib.cm.viridis)
pl.plot(X[X*X + Y*Y < 0.35*0.35],Y[X*X + Y*Y < 0.35*0.35],'ro',
    label = 'Points with potential 1V')
pl.legend()
pl.show()
```

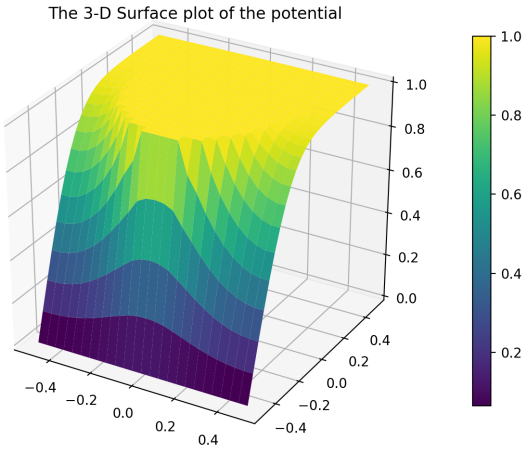

Figure 7: 3-D Surface plot of Potential

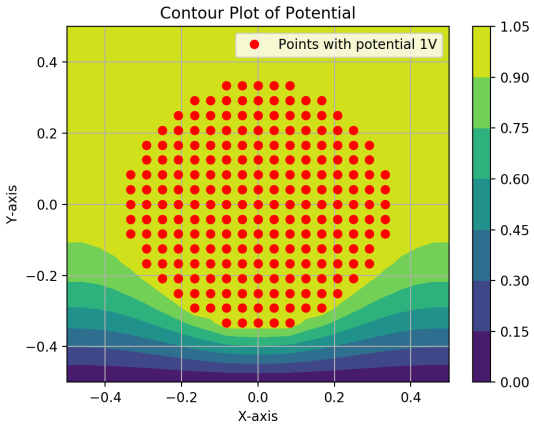

Figure 8: Contour plot of Potential

2.6 Plotting the Current

We can calculate the current density from the potential, the relation for which is given by :

$$\begin{aligned} J_x &= -\frac{\partial \phi}{\partial x} \\ J_y &= -\frac{\partial \phi}{\partial y} \end{aligned} \tag{7}$$

$$\begin{aligned} J_{x,ij} &= -\frac{1}{2}(\phi_{i,j-1} - \phi_{i,j+1}) \\ J_{y,ij} &= -\frac{1}{2}(\phi_{i-1,j} - \phi_{i,+1j}) \end{aligned} \tag{8}$$

We use quiver plot for highlighting the **vector quantity** current density's directions.

```
Jy = pl.zeros((Ny,Nx))
Jx = pl.zeros((Ny,Nx))
Jx[:,1:-1] = 0.5*(Phi[:,0:-2]-Phi[:,2:])
Jy[1:-1,:] = 0.5*(Phi[0:-2,:]-Phi[2:,:])
pl.figure(8)
pl.quiver(x,y,Jx,Jy)
pl.plot(X[X*X + Y*Y < 0.35*0.35],Y[X*X + Y*Y < 0.35*0.35],'ro',
    label = 'Points with potential 1V')
pl.xlabel('X-axis')
pl.ylabel('Y-axis')
pl.legend(loc = 1)
pl.title('Vector plot of current density')
pl.show()
```

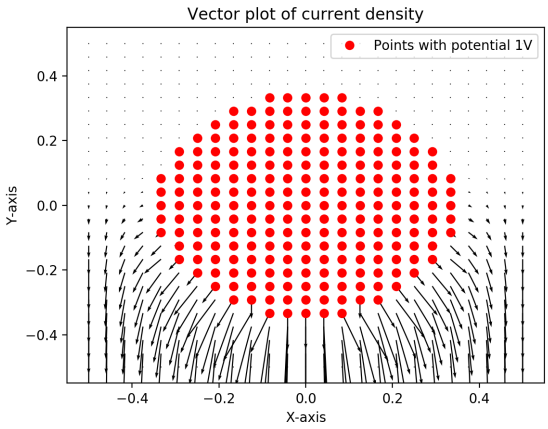

Figure 9: Quiver plot of Current Density

We can notice that the current is concentrated in the bottom half between the plate and the ground. This is because of high Electric field in the region between the two compared to other places, which is clear from the Potential contour plot, depicting how rapidly potential varies there.
This is similar to a capacitor under break- down condition. Thus most of the current flows in that region.

2.7 Calculating Temperature

The current density and equivalently current, as we saw, is high in the bottom region between plate and ground. Thus there will be more **ohmic loss**

occurring there compared to other places. We, therefore expect that region to heat up strongly and have a higher temperature.
We calculate a 2-D Laplace Equation for Temperature, similar to how we did for Potential in an iterative approach. The Laplace equation for Temperature is given by :

$$\nabla.(\kappa \nabla T) = q = \frac{1}{\sigma}|J|^2 \tag{9}$$

```
Temp = np.zeros((Ny,Nx))*300
for k in range(Niter):
    Temp[1:-1,1:-1] = 0.25*(Temp[1:-1,0:-2]+Temp[1:-1,2:]+
        Temp[0:-2,1:-1]+Temp[2:,1:-1]+(Jx[1:-1,1:-1]**2)+(Jy[1:-1,1:-1]**2))
    Temp[0,:] = 300
    Temp[1:-1,0] = Temp[1:-1,1]
    Temp[1:-1,-1] = Temp[1:-1,-2]
    Temp[-1,:] = Temp[-2,:]
    Temp[ii] = 300

#Contour plot of the Temperature
PLOT(x,y,9,"X-axis","Y-axis",pl.contourf,Temp,
    "Contour Plot of Temperature",cmap = matplotlib.cm.hot)
pl.show()
```

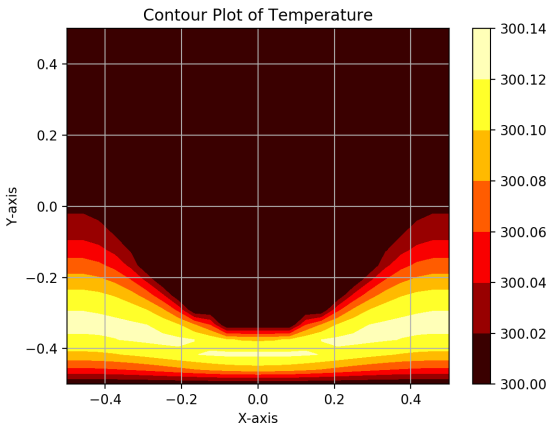

Figure 10: Contour Plot of Temperature

3 Conclusions

- We made 2-D contour and 3-D plots for better visual understanding of the physical properties (like Potential, Temperature).
- We solved 2-D Laplace equation for Potential and Temperature using iterative approach.
- We used Least Squares Fitting to estimate the parameters of the exponential form of error.
- We used quiver plot to observe Current density (a vector quantity) over the region.
- We also quantified the Temperature and analysed the heating effects due to current.