Assignment No 5

Shivanshu Shekhar

March 11, 2022

1 Abstract

In this weeks assignment we are going to solve for the current in a resistor, current depends on the shape of the resistor and we want to find which part gets the hottest.

2 Introduction

A wire is soldered to the middle of a copper plate and its voltage is held at 1 Volt. One side of the plate is grounded, while the remaining are floating. The plate is 1 cm by 1 cm in size.

From Ohms law:

$$\vec{j} = \sigma \vec{E}$$

The Continuity Equation:

$$\nabla \cdot \vec{j} = -\frac{\partial \rho}{\partial t}$$

From the above equation and using $\vec{E} = -\nabla V$ we get

$$\nabla^2 \phi = \frac{1}{\rho}\frac{\partial \rho}{\partial t}$$

For DC currents RHS is zero:

$$\nabla^2 \phi = 0$$

3 Problems and Solutions

3.1 Import the libraries

We imported the necessary libraries with is shown below

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

We used argparse for taking command line inputs, and pylab for the necessary math and plotting functions. We used numpy's lstsq function to calculated the best fit later in the questions.

3.2 Define the Parameters

We defined the stated parameters i.e. Nx, Ny, Radius, Niter which have defaults values as 25,25,8,1500 respectively and these values are also changed to user input values if present.

```
try:
    parser = argparse.ArgumentParser(description = "Takes inputs")
    parser.add_argument("Nx", metavar = "Nx", nargs = "?",
                        const = 1, default=25, type = int)
    parser.add_argument("Ny", metavar = "Ny", nargs = "?",
                        const = 1, default=25, type = int)
    parser.add_argument("Radius", metavar = "R", nargs = "?",
                        const = 1, default=8, type = float)
    parser.add_argument("Niter", metavar = "Niter", nargs = "?",
                        const = 1, default=1500, type = int)

    args = parser.parse_args()


except:
    parser = argparse.ArgumentParser(description = "Takes inputs")
    parser.add_argument("-Nx", metavar = "--Nx", default = 25, type = int)
    parser.add_argument("-Ny", metavar = "--Ny", default = 25, type = int)
    parser.add_argument("-Radius", metavar = "--R", default = 8, type = float)
    parser.add_argument("-Niter", metavar = "--Niter", default = 1500, type = int)

    args = parser.parse_args()

Nx, Ny, Radius, Niter = args.Nx, args.Ny, args.Radius, args.Niter
```

3.3 Allocate the potential array and initialize it

We created the phi array as out potential array and we defined it to have shape of (Ny, Nx), we also used the **contour()** function to get a contour plot of the potential, with the nodes in the V = 1 region marked in red color as shown below

```
phi = np.zeros((Ny, Nx))
x = np.linspace(-0.5, 0.5, Nx);y =np.linspace(-0.5, 0.5, Ny)
X,Y= meshgrid(x, y)
radius = 1.01*Radius/(min(Nx, Ny) -1) #Tolerance of 1% in radius
ii = np.where(X ** 2 + Y ** 2 <= radius** 2)
phi[ii] = 1.0
```

For plotting:

```
# Plotting Countours
figure(figsize=(10, 8))
title("Contour Plot of the Potential", fontsize=16)
clabel(plt.contour(X, Y, phi))
xlabel("$x$", fontsize=16)
```

```
ylabel("$y$", fontsize=16)
scatter(X[0, ii[1]], Y[ii[0], 0], color="r", label="V=1")
legend()
grid()
show()
```

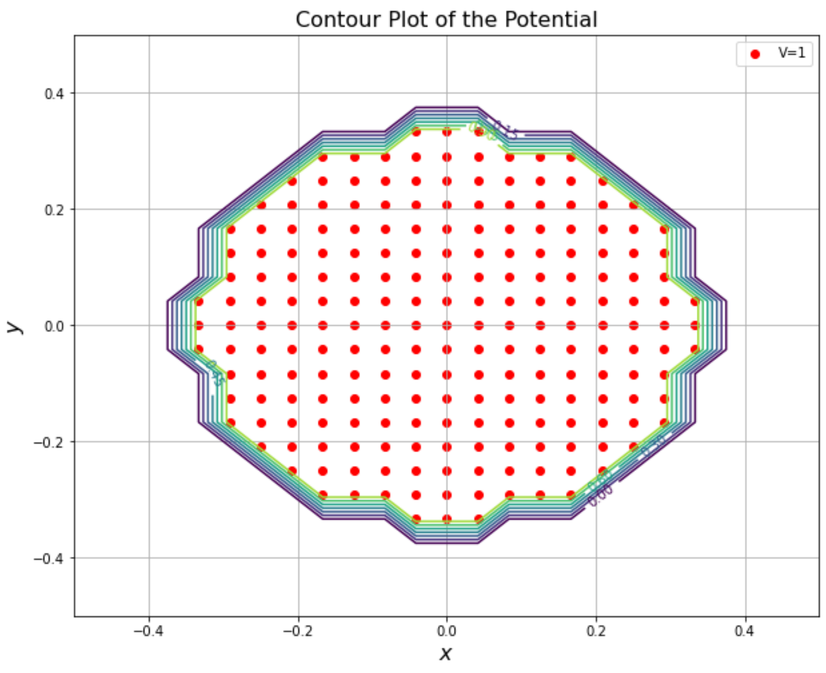


Figure 1: Question 3

3.4 Perform the iteration

We implemented the logic given in the problem statement using the code below

```
#Interations
error = np.zeros((Niter, ))
for k in range(Niter):
    oldphi = phi.copy()
    phi[1:-1,1:-1]=0.25*(oldphi[1:-1,0:-2]+ oldphi[1:-1,2:]+
                         oldphi[:-2,1:-1] + oldphi[2:,1:-1])

    phi[1:-1, 0], phi[1:-1, -1], phi[-1, :] = phi[1:-1, 1], phi[1:-1, -2], phi[-2, :]
    phi[ii] = 1.0
    error[k] = abs(phi-oldphi).max()
```

3.5 Updating the Potential

We updated the potential every iteration with accordance to the following equation:

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

We did this using the following code:

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

3.6 Boundary Conditions

The bottom boundary is grounded and the other 3 sides have no potential change in normal direction of the boundaries of electrode.

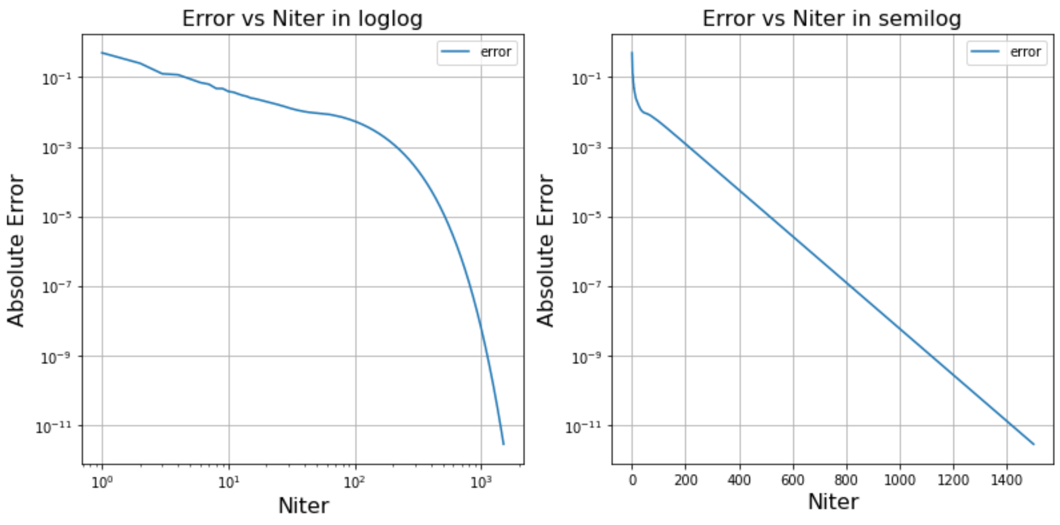
This is accomplished with the following code:

```
phi[1:-1, 0], phi[1:-1, -1], phi[-1, :] = phi[1:-1, 1], phi[1:-1, -2], phi[-2, :]
phi[ii] = 1.0
error[k] = abs(phi-oldphi).max()
```

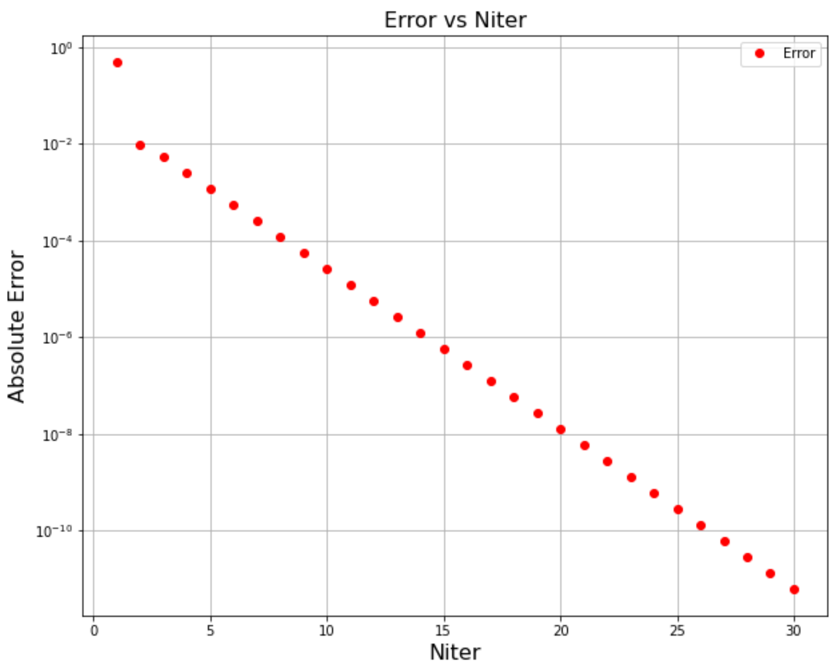
We also made sure that the potential of the electrode doesn't change.

3.7 Graph the results

We plotted Error vs Number of iterations in loglog and semilog scale as shown



We also plotted every 50th point to see individual data points



We fitted the data to an exponential i.e. $y = A\exp^{Bx}$ by taking its log first and fitting a straight line to it.

So we fitted a string line for the equation:

$$\log y = \log A + Bx$$

We did the following by using the following code:

```
#fitting
A = np.concatenate([np.arange(1,Niter+1).reshape(-1,1),np.ones((Niter,1))]
                        , axis =1)
 fit1 = np.linalg.lstsq(A, np.log(error).reshape(-1,1), rcond=-1)[0]
#For pointer after 500 iterations
 fit2 = np.linalg.lstsq(A[500:],
                              np.log(error)[500:].reshape(-1,1), rcond=-1)[0]
```

We got the following fit:

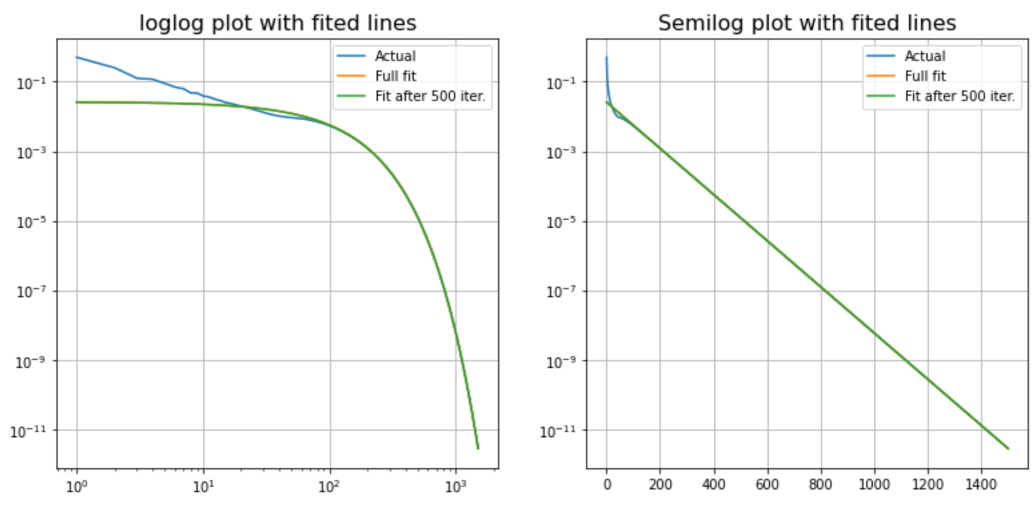


Figure 2: Caption

3.8 Stopping Condition

We calculated and plotted the cumulatively sum using the formula provided to us that was

$$Error = -\frac{A}{B} \exp B(N + 0.5)$$

We did this using the following code:

```
#Cummulative sum
figure(figsize=(70,70))
subplot(10,10,1)
title("Cummulative Sum in loglog", fontsize = 16)
loglog(x_, -np.exp(fit2[1])/fit2[0]*np.exp(fit2[0]*(x_+0.5)), label = "Error")
grid()
xlabel("Niter", fontsize=16)
ylabel("Error", fontsize=16)
legend()
subplot(10,10,2)
title("Cummulative Sum in semilog", fontsize = 16)
semilogy(x_, -np.exp(fit2[1])/fit2[0]*np.exp(fit2[0]*(x_+0.5)), label = "Error")
grid()
xlabel("Niter", fontsize=16)
ylabel("Error", fontsize=16)
legend()
show()
```

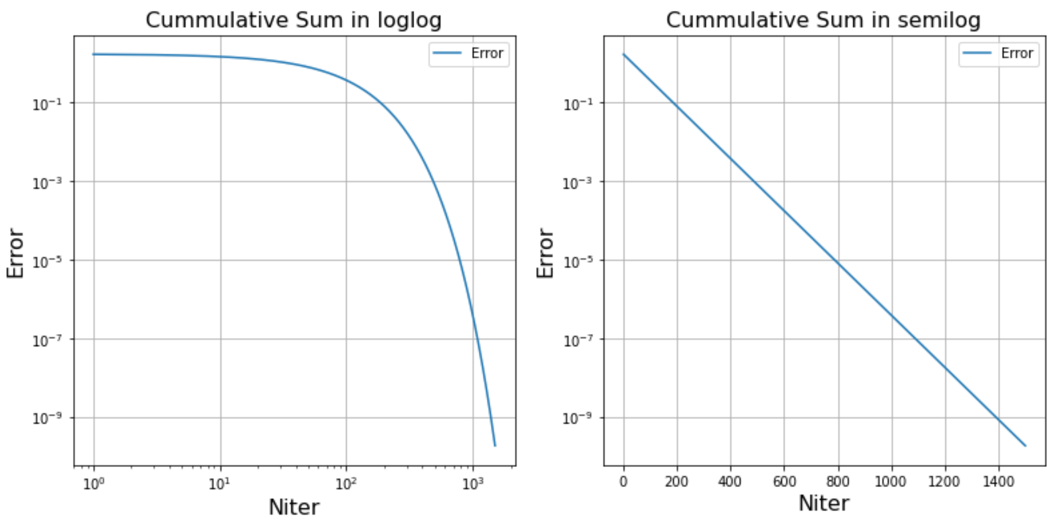


Figure 3: Caption

3.9 Surface Plot of Potential

We used the template code provided to us to plot the 3D surface plot of the potential:

We use the following code:

```
#3D plot
fig = plt.figure(figsize=(10,8))
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, phi,cmap = cm.jet,linewidth=0, antialiased=False)
fig.colorbar(surf, shrink=0.5, aspect=5)
title('3-D Surface plot of the potential')
xlabel('x')
ylabel('y')
show()
```

3.10 Contour Plot of the Potential

We used the same plotting logic from 2.3 to draw this plot:

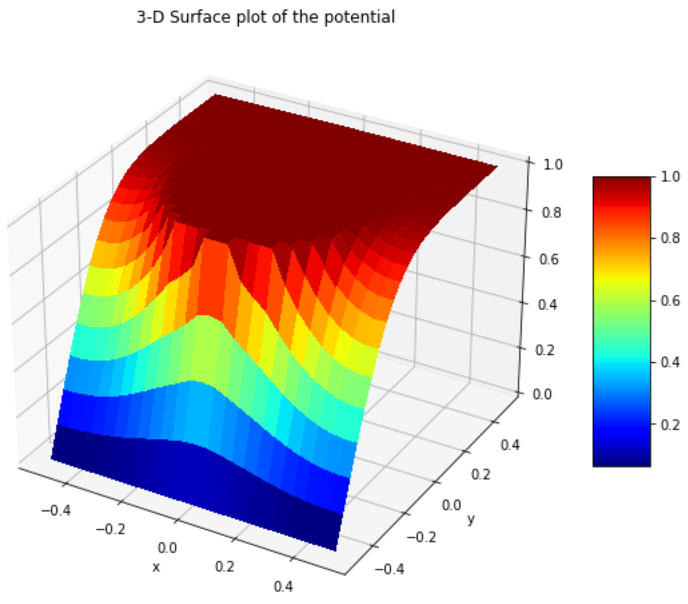


Figure 4: Caption

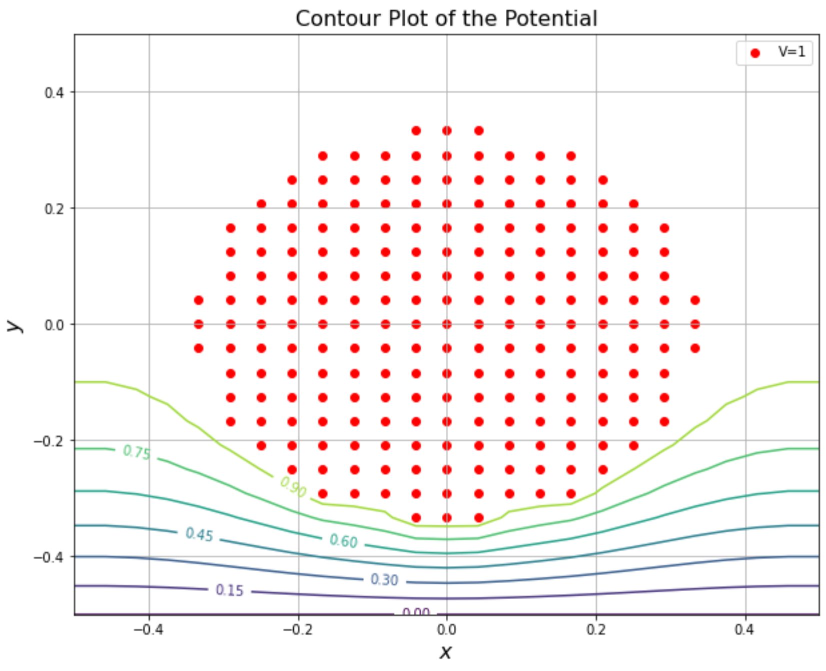


Figure 5: Caption

3.11 Vector Plot of Currents

As given in the problem statement the currents are given by:

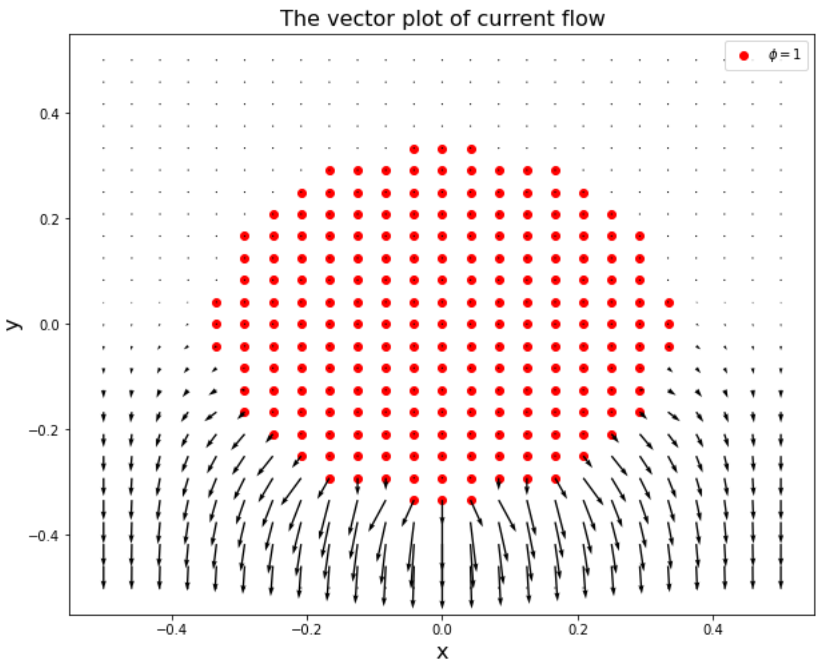
$$J_{x,ij} = \frac{1}{2}(\phi_{i,j-1} - \phi_{i,j+1})$$
$$J_{y,ij} = \frac{1}{2}(\phi_{i-1,j} - \phi_{i+1,j})$$

We used the following to calculate the J's:

```
#Quiver Plot
Jx = np.zeros_like(phi)
Jy = np.zeros_like(phi)
Jx[:,1:-1] = (phi[:,:-2] - phi[:,2:])/2
Jy[1:-1,:] = (phi[:-2,:] - phi[2:,:])/2
```

We plotted the plots using the following:

```
figure(figsize=(10, 8))
title("The vector plot of current flow", fontsize=16)
scatter(X[0, ii[1]], Y[ii[0], 0], color="r", label="$\phi=1$")
quiver(X, Y, Jx[:,::-1], Jy[:,::-1], scale=4)
xlabel("x", fontsize=16)
ylabel("y", fontsize=16)
legend()
show()
```



4 Conclusion

We used discrete differentiation to solve for Laplace's Equations, this method is know to be very slow convergence rate.