Assignment 5

Sagnik Ghosh, EE19B132

March 2021

1 Introduction

In this assignment, we aim to solve for the spacial currents in a 2D resistive region. This can be done by solving the Laplace's equation. We implement a naive method to solve the Laplace Equation in Python.

The Laplace's equation in 2D is as follows:

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

This simplifies to,

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

Hence, the potential at any point is the average of its neighbours.

We use the above equation extensively for solving the Laplace equation. We iterate and update each point as the mean of its neighbours during each iteration.

2 Initializing and Solving for the Potential

We take in arguments for the resolution of the matrix, radius of the wire and the number of iterations using sys.argv() as follows,

```
nx = int(sys.argv[1])
ny = int(sys.argv[2])
radius = (float(sys.argv[3])/nx + float(sys.argv[3])/ny)/2
niter = int(sys.argv[4])
```

We then initialize the potential matrix as follows,

```
phi = np.zeros((ny, nx))

y = np.linspace(-0.5, 0.5, ny)
x = np.linspace(-0.5, 0.5, nx)
Y, X = np.meshgrid(y, x)

phi[np.where(X*X+Y*Y <= radius*radius)]=1
```

Now that we have initialized the matrix, we iteratively solve for solve for the final potential matrix and record the error after every iteration.

```
errors = np.zeros(niter)

for i 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[1:-1,0] = phi[1:-1,1]
    phi[1:-1,-1] = phi[1:-1,-2]
    phi[0,0:] = phi[1,0:]
    phi[-1,1:-1] = 0
    phi[np.where(X*X+Y*Y <= radius*radius)]=1
    errors[i] = np.max(np.abs(phi-oldphi))
```

We now have the final potential matrix and the list of errors.

3 Error Analysis

We have recorded the evolution in error after every iteration in the previous section. Let us plot this to derive more insights.

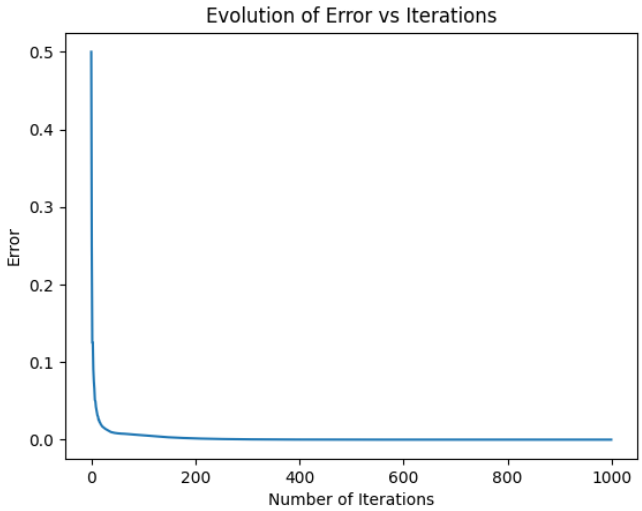

Figure 1: Error vs Iterations

As the error decays rapidly, it would be in order to use a semilogy plot or a loglog plot for better analysis.

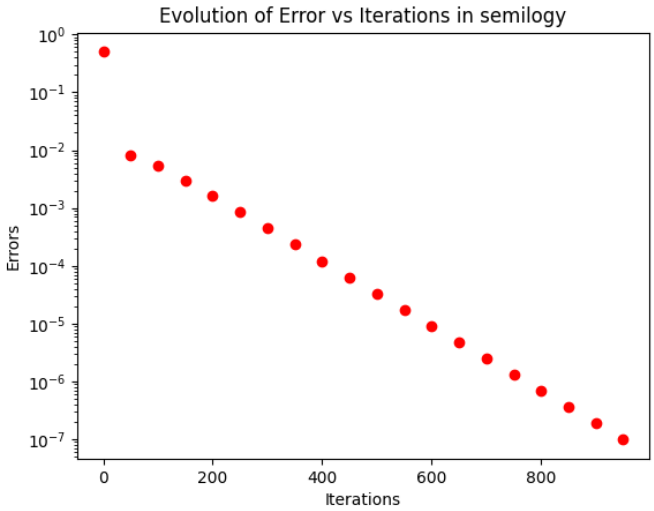

Figure 2: Error vs Iterations in Semilogy

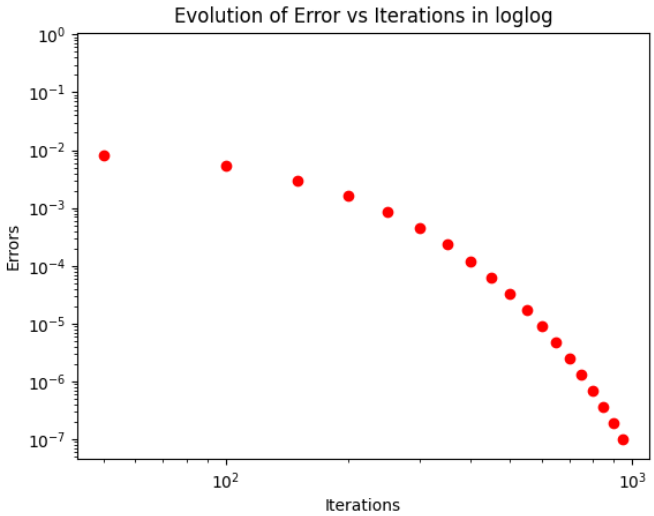

Figure 3: Error vs Iterations in loglog

It can be noticed that the errors vary exponentially with respect to the number of iterations as follows,

$$y = Ae^{Bx}$$
$$\log y = \log A + Bx$$

We can use Least Squares Method to estimate the values of A and B as follows,

```
def generate_mat(niter):
    a = np.ones((niter, 1))
    b = np.arange(0, niter)
    M = np.c_[a,b]

    return M

log_error = np.log(errors)
M = generate_mat(niter)

x = np.linalg.lstsq(M, log_error, rcond = None)[0]

A = np.exp(x[0])
B = x[1]
```

The above code fits the error for all the iterations. As the error plot behaves exponentially only after the 500th iteration.

```
M2 = generate_mat(niter-500)
M2[:,1]+=500
x2 = np.linalg.lstsq(M2, log_error[500:], rcond = None)[0]

A2 = np.exp(x2[0])
B2 = x2[1]
```

Now let us plot the above estimated error functions.
It can be noticed that both the plots overlap, they are nearly identical.

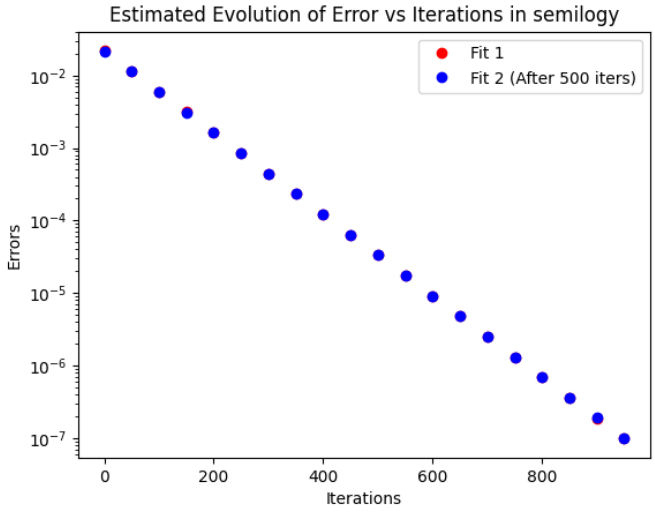

Figure 4: Estimated Error vs Iterations in semilog

4 Visualising the Final Potential

The 3D plot of the final potential using mplot3d.axes3d and the contour plot of the potential are as follows,

```
#3D Plot
fig1=plt.figure(4)
ax=p3.Axes3D(fig1)
plt.title('The 3-D surface plot of the potential')
surf = ax.plot_surface(Y, X, phi.T, rstride=1, cstride=1, cmap=cm.jet)
plt.xlabel(r'x')
plt.ylabel(r'y')
plt.show()

#Contour Plot
plt.contour(X,Y, phi.T)
plt.gca().invert_yaxis()
plt.title('Contour Plot of potential')
plt.xlabel(r'x')
plt.ylabel(r'y')
plt.plot(X[ii], Y[ii], 'ro')
plt.savefig('Figure_6.png')
```

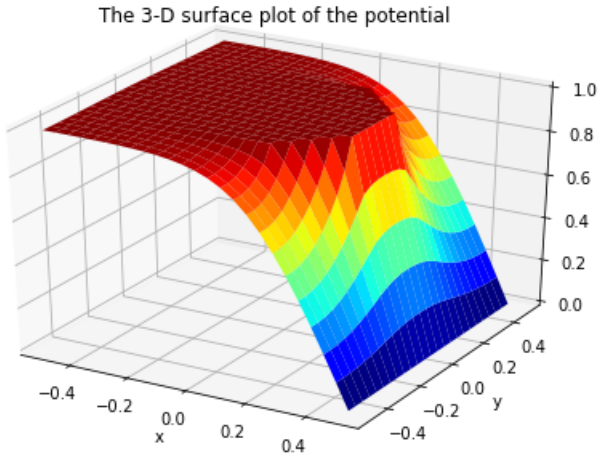

Figure 5: 3D Plot of Potential

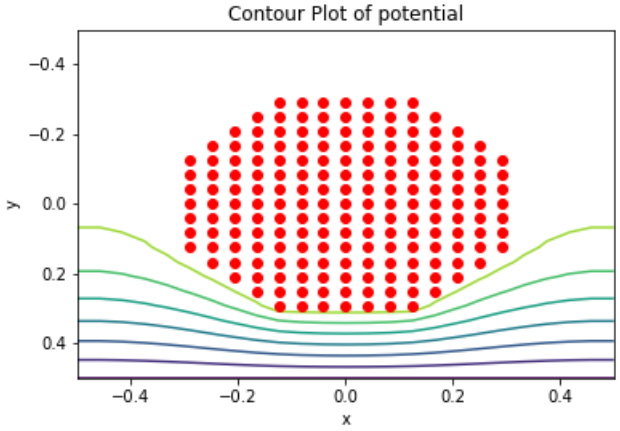

Figure 6: Contour Plot of Potential

5 Plot of Currents

The equations governing the current vectors are as follows,

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

J_x and J_y can be computed as follows,

```
Jx = np.zeros((ny,nx))
Jy = np.zeros((ny,nx))
Jx[:,1:-1] = 0.5*(phi[:,0:-2]-phi[:,2:])
Jy[1:-1,:] = 0.5*(phi[2:, :]-phi[0:-2,:])
```

This can be plotted using the quiver() function. The following is the plot,

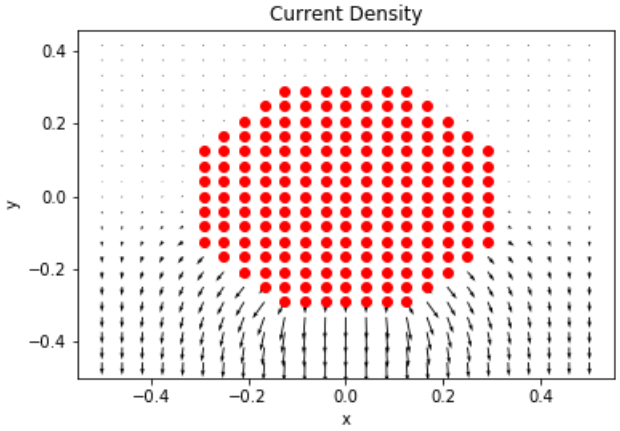

Figure 7: Current Plot

From the current density plot, we can notice that the current flows predominantly through the bottom half of the wire. This is because the electrons would choose a path of lower resistance, i.e., the shorter path from low voltage to high voltage.

6 Conclusion

Using a finite differentiation approximation, we have found a solution to Laplace's equation for a given system. The error is seen to decay at a highly gradual pace. Thus the chosen method of solving Laplace's equation is inefficient. On analysing the quiver plot of the currents, it was noticed that the current was mostly restricted to the bottom of the wire, and was perpendicular to the surface of the electrode and the conductor.