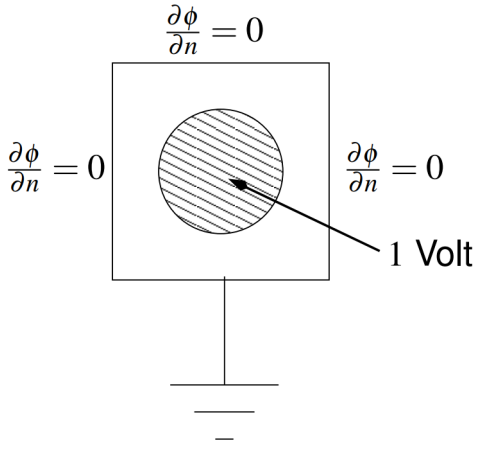
EE2703 : Applied Programming Lab
Assignment 5

P N Neelesh Sumedh
ee19b047

March 25, 2021

1 Introduction

In this assignment we wish to solve the potentials and current through different parts of a copper plate resistor with a wire soldered to the middle of the plate. The wire is at a potential of 1V and the bottom side is grounded.



According to the equation:

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

Assuming presence of only DC, the RHS of the above equantion becomes zero. Therefore, the final equation becomes

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

The Numerical Solution of the above equation for 2D case is:

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

2 Assignment

2.1 Defining the Parameters

The parameters which control the run are as follows. These parameters are given by the user in the commandline else the program runs on the default values.

```
Nx = 25 (Default) (The no of divisions on the X-Axis between -0.5 to 0.5)
Ny = 25 (Default) (The no of divisions on the Y-Axis between -0.5 to 0.5)
radius = 0.35 (Default) (The radius of the soldered wire)
Niter = 1500 (Default) (No of iterations to be performed)
```

Basically, we define an Ny X Nx matrix which is used to represent the plate. In the default case it is 25x25. In this matrix, the central part of matrix with elements which satisfy the condition $X * X + Y * Y \leq radius * radius$.

2.2 Allocate ϕ array and initializing it

We first initialize a zero numpy array of size Ny X Nx. In this matrix, the central part of the matrix which represents the circle with radius = radius_wire is initiated to value of 1V. This part of the matrix is found out by defining a meshgrid and using numpy.where() function. The contour plot of initialized ϕ array is plotted.

```
1    phi = np.zeros((Ny,Nx))
2    x = np.linspace(-0.5, 0.5, num = Nx)
3    y = np.linspace(-0.5, 0.5, num = Ny)[::-1]
4    X,Y = meshgrid(x,y)
5    phi[np.where(X*X + Y*Y <= radius**2)] = 1.0
6    X_con = X[np.where(X*X + Y*Y <= radius**2)]
7    Y_con = Y[np.where(X*X + Y*Y <= radius**2)]
8    plt.figure(1)
9    plt.contourf(X,Y, phi)    # Plotting the contour plot of initial potential
10   plt.plot(X_con, Y_con , 'o', color = 'red')  # Plotting red dots in the
     region enclosed by wire
11   plt.title('Contour Plot of Potential')
12   plt.xlabel('X')
13   plt.ylabel('Y')
14   plt.colorbar()
15   plt.show()
```

The contour plot is shown below.

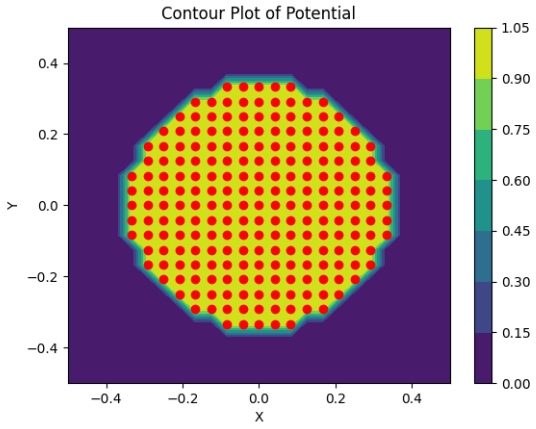


Figure 1: Contour of Initial Potential

2.3 Performing the Iteration

The now initialized potential array is updated using Equation(1) given above. After updating the potential except at the boundaries, the boundary condition is applied on this updated array. After applying the boundary conditions, the center circle of the array values are reassigned value 1 as there might be some changes in those values during the updating process. During the process, before the next iteration begins, error is calculated between the new updated potential and the old potential.

```
1    for k in range(Niter):
2        oldphi = phi.copy()
3        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])
4        phi[1:-1,0] = phi[1:-1,1]
5        phi[1:-1, -1] = phi[1:-1, -2]
6        phi[0, 1:-1] = phi[1, 1:-1]
7        phi[np.where(X*X + Y*Y <= radius_wire**2)] = 1.0
8        errors[k] = np.max(np.abs(phi-oldphi))
```

2.4 Plotting the Error

We will plot the error on loglog plot and semilog plot. We can see that the error reduces very slowly. The loglog plot and semilog plot of error is shown below.

```
1    plt.semilogy(range(Niter), errors ) # For plotting semilog plot
2    plt.show()
3    plt.loglog(range(Niter), errors ) # For Plotting loglog plot
4    plt.loglog(range(Niter)[::50], errors[::50],'o') # Datapoints after every 50
     iterations
5    plt.show()
```

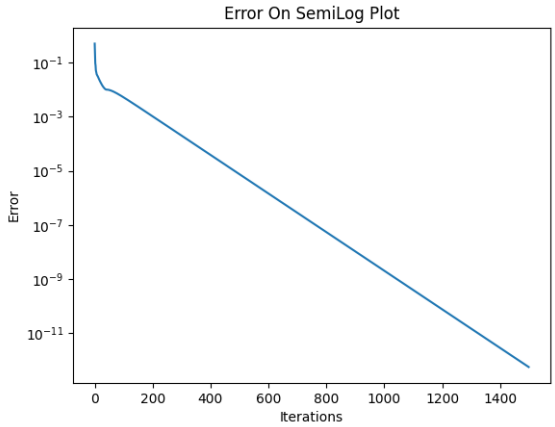


Figure 2: Error on SemiLog Plot

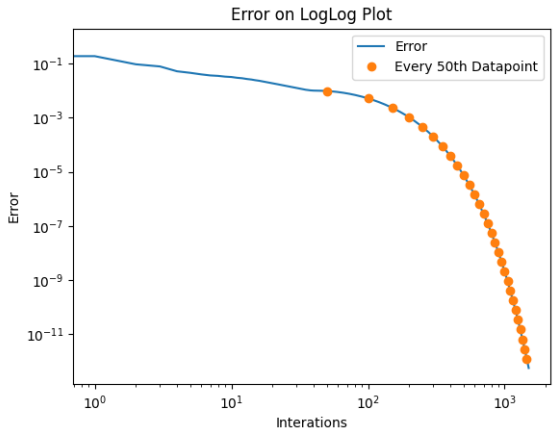


Figure 3: Error on LogLog Plot

In the loglog plot, we can see that for larger iterations, the graph is more like an decreasing exponential.

$$y = Ae^{Bx}$$

Taking log of the above equation, least squares method is used to find out A and B.

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

We extract the above fit for all iterations and also for error entries only after 500th iteration. The data is plotted with labels fit1 and fit2. The plotted graphs are shown below.

```
    def fitting(err, ite):
        ite = ite.reshape(len(ite),1)
        ones = np.ones(np.shape(ite))
        M = np.hstack((ones, ite))
        b = np.log(err)
```

```
        out = lstsq(M, b)[0]
        B = out[1]
        A = np.exp(out[0])
        return A, B

    iteration = np.array(list(range(Niter)))
    fit1_vals = fitting(errors, iteration)
    fit2_vals = fitting(errors[500:], iteration[500:])
    fit1 = fit1_vals[0]*np.exp(fit1_vals[1]*iteration)
    fit2 = fit2_vals[0]*np.exp(fit2_vals[1]*iteration)
```

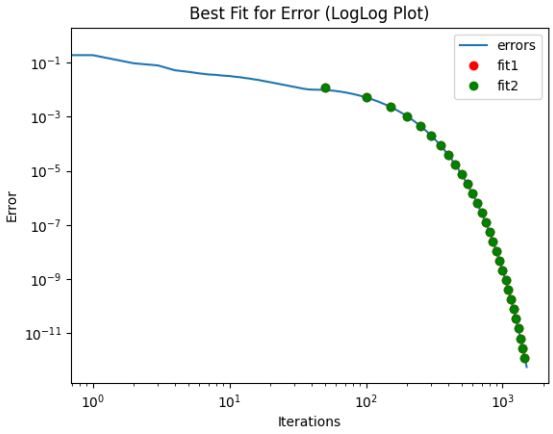


Figure 4: Best Fit plot for Error

2.5 Surface Plot of Potential

The potential array which we obtained is used to plot the surface plot of voltage. The plotting is done using *plot_surface* function of matplotlib. The code to plot:

```
    fig = plt.figure(5)
    ax = plt.axes(projection = '3d')
    surf = ax.plot_surface(X,Y,phi, cmap = cm.jet, rstride=1, cstride=1)
    ax.set_title('The 3D surface plot of the Potential')
    plt.show()
```

The surface plot is shown below.

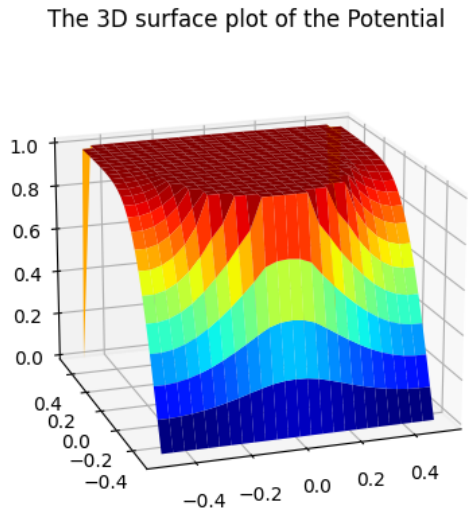


Figure 5: Surface plot of ϕ

2.6 Contour Plot of Potential

The Contour Plot of Potential after interation is plotted. The plot can be seen below.

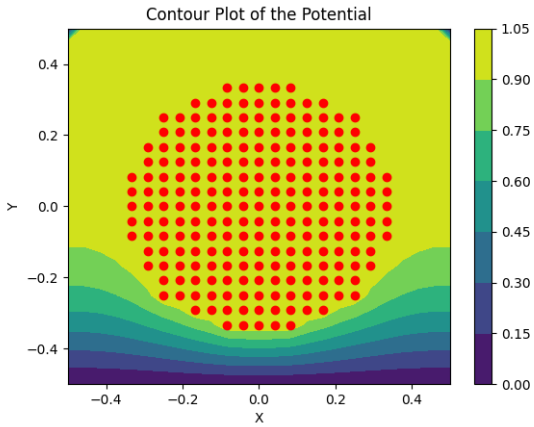


Figure 6: Contour of Potential

2.7 Vector Plot of Currents

For computing the current, we have to find the gradient of potential.

$$j_x = -\frac{\partial \phi}{\partial x}$$
$$j_y = -\frac{\partial \phi}{\partial y}$$

Again my solving numerically, these equations can be written as

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

This is done using the ϕ array, and the J is found out. The code is as follows:

```
    Jx = np.zeros((Ny,Nx))
    Jy = Jx.copy()
    Jx[1:-1,1:-1] = 0.5*(phi[1:-1,0:-2] - phi[1:-1, 2:])
    Jy[1:-1,1:-1] = 0.5*(phi[2:,1:-1] - phi[0:-2, 1:-1])
    plt.plot(X_con, Y_con, 'o', color = 'red')
    plt.quiver(X,Y,Jx,Jy, scale=5)
    plt.show()
```

The vector plot of currents is shown below. We can see that the current is more dense at the bottom where it is grounded and the current at the top is almost negligible. The reason is the voltage at the top is more or less equal to 1V and the soldered wire is at 1V therefore no current flows upward. The potential of the bottom part is almost 0V as it is grounded. Therefore, due to more potential difference the current flow downward is more.

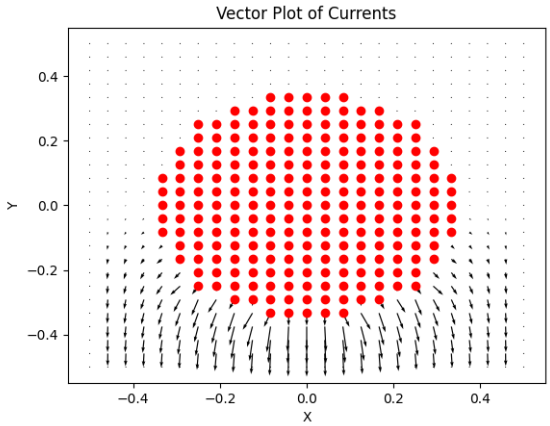


Figure 7: Vector Plot of Currents

Conclusion

Therefore, we have found out the potential using numerical method of Laplace Equations. This method is not very efficient as the rate of decrease in error is very less.