EE2703 : Applied Programming Lab
Assignment 5
The Resistor Problem and Laplace Equation

Bachotti Sai Krishna Shanmukh
EE19B009

March 25, 2021

1 Introduction

In this assignment, we solve for electric potential and current density in a square shaped resistor which is grounded on one side and floating on other three sides. In the middle of the copper plate is a soldered cylindrical wire which maintains the potential at 1V. We shall use the following equations to solve this problem:

The Continuity Equation:

$$\nabla \cdot \vec{j} = -\frac{\partial \rho}{\partial t} \tag{1}$$

Ohms Law:

$$\vec{j} = \sigma \vec{E} \tag{2}$$

Relationship between Electric field and Electric potential:

$$\vec{E} = -\nabla \phi \tag{3}$$

From (1), (2) and (3) we get,

$$\nabla^2 \phi = \frac{1}{\rho} \frac{\partial \rho}{\partial t} \tag{4}$$

For DC currents, the RHS of equation (4) is zero. This implies

$$\nabla^2 \phi = 0 \tag{5}$$

The above equation is often referred to as the Laplace Equation

2 Defining the parameters

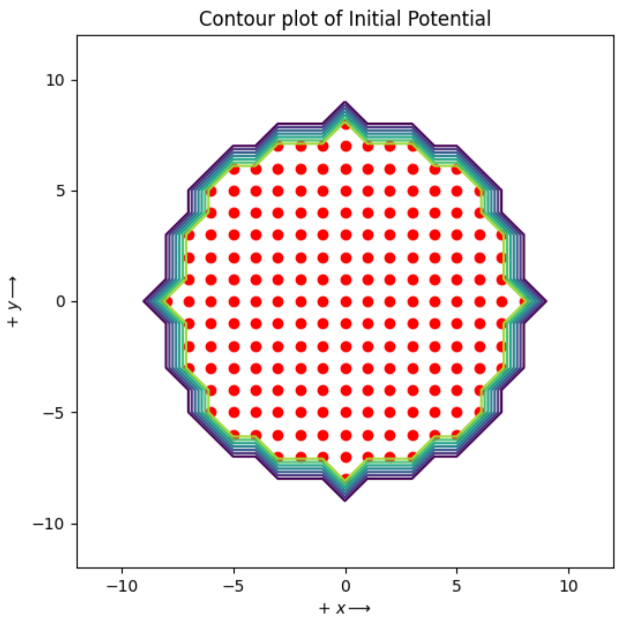
The default parameters chosen are 25x25 grid and radius 8. We perform 1500 iterations on the difference equations to calculate the Electric potential across this grid. Note that the code accepts input parameters of our choice from the command line. If no input parameters are specified it considers the default values.

```
      Nx = 25
      Ny = 25
      radius = 8
      Niter = 1500
```

3 Initialization

We create a 2D array of shape N_y x N_x with entries as zeroes and find the coordinates (indices) which lie within the radius. These coordinates are initialized to a potential of 1. We then plot a contour plot.

```
phi = np.zeros((Ny,Nx))
#No of columns is Nx and No of rows is Ny
x = np.arange(Nx) - (Nx-1)/2
# x contains the discrete positions along X direction (right)
y = np.flip(np.arange(Ny) - (Ny-1)/2)
# y contains the discrete positions along Y direction (top)
X,Y = np.meshgrid(x,y)
ii = np.where(X*X + Y*Y <= radius**2)
# indices /postions of points which lie within the radius
phi[ii] =1 # These set of points always remain at potential = 1
```



4 Performing the Iterations

- We first save the values of previous iteration to a new array before updating the existing array. This is done by

```
    oldphi = phi.copy() # saving a copy
```

- We then update the potential using difference equation derived from the Laplace Equation for our grid. Using equation (5) on a 2D plane with discrete points, we get:

$$\phi_{i,j} = 0.25 * (\phi_{i,j-1} + \phi_{i,j+1} + \phi_{i+1,j} + \phi_{i-1,j}) \tag{6}$$

  Note that we use vectorized code to update the potential rather than using loops to iterate through each row and column. This improves the latency of the code

```
        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])
```

- Using Boundary conditions we can complete our solution for ϕ. We know that one side is grounded i.e., $\phi = 0$ while the remaining are hanging. At these edges, the change in potential along the normal direction is 0. The inner circular area is maintained at 1V.

```
phi[ii] =1  # restoring Potential =1 in the circle
phi[1:-1,0] = phi[1:-1,1]  # Boundary conditions
phi[1:-1,-1] = phi[1:-1,-2] # On hanging
phi[0,1:-1] = phi[1,1:-1] #sides
```

- Calculating Error

```
errors[k] = np.max(np.abs(phi-oldphi))
```

This completes one iteration and the loop repeats. For large number of iterations, the potential matrix settles to a steady value and satisfies the difference equation and boundary conditions too

5 Errors

5.1 Plotting the errors

We will plot the errors on semi-log and log-log plots. We note that the error falls really slowly and this is one of the reasons why this method of solving the Laplace equation is discouraged

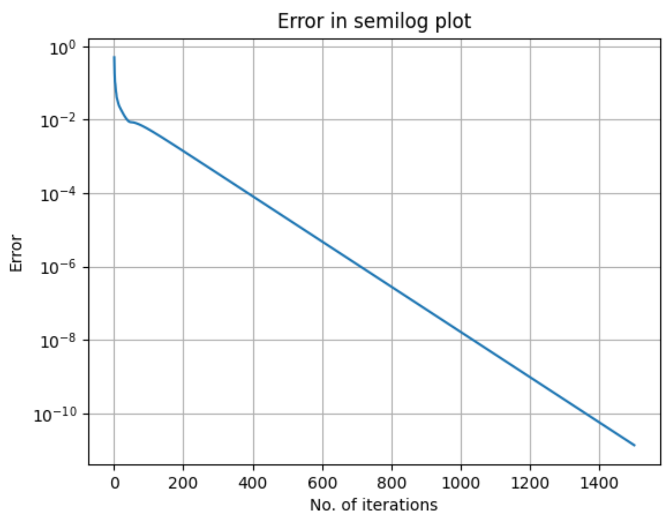


Figure 2: Semilog plot

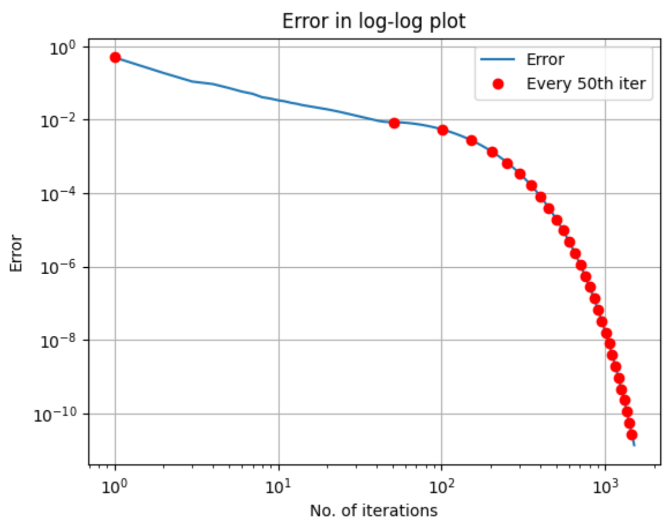


Figure 3: Log Log plot

5.2 Fitting the error

From Figure 2 and 3, we can infer that the error is decaying exponentially for higher iterations. We try to find two fits for the curve, one considers all iterations and their errors, while the second fit is based on errors from 500th iteration.

```
i = np.arange(1,Niter+1)
one_array = np.ones(Niter) # array with all 1s
M = np.c_[one_array,i]     # for all Niter
M_500 = np.c_[one_array[500:],i[500:]] # from 500th iter
v = slg.lstsq(M,np.log(errors))[0] # Least squares
v_500 = slg.lstsq(M_500,np.log(errors[500:]))[0]
error_lstsq = np.exp(np.dot(M,v))
error500_lstsq = np.exp(np.dot(M_500,v_500))
```

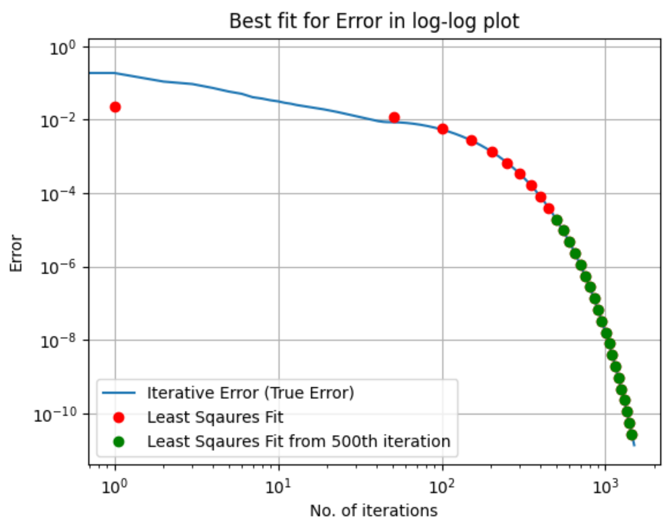


Figure 4: Best Fit using Least Squares. Every 50th iter is plotted

Figure 4 clearly depicts that there is very little difference between the two fits and are close to true error.

5.3 Maximum Possible Error

This method of solving Laplace's Equation is known to be one of the worst available. This is because of the very slow coefficient with which the error reduces.

```
def cum_error(x):
    """
    Input argument x: No. of iterations
    Output is cumulative max possible error
    """
    return -A/B*np.exp(B*(x+0.5))

"""
Log log plot for cumulative error
"""
fig , ax = plt.subplots(num =4)
ax.loglog(i[100::100],cum_error(i[100::100]),'ro')
plt.xlabel(r'No. of iterations')
plt.ylabel(r'Cumulative Error')
plt.title(r'Cumulative Error in log-log plot (Every 100th iter)')
plt.grid()
```

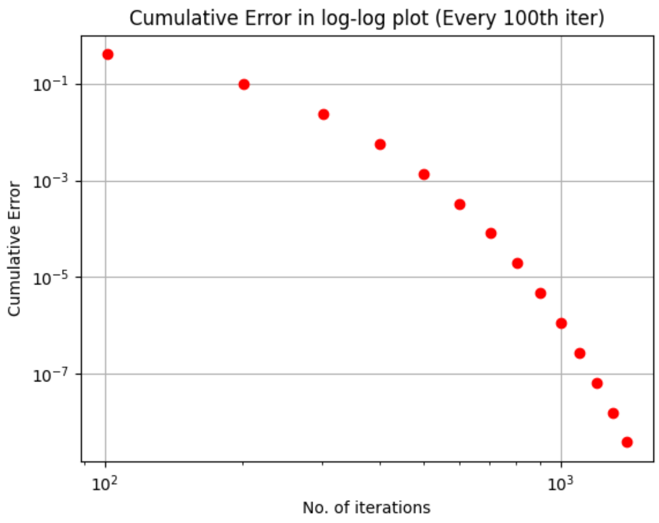


Figure 5: Cumulative Error on Log Log scale

6 Plotting Electric Potential ϕ

The potential which has been computed through difference equation and boundary conditions is now plotted in Figure 6 and 7

```
Contour plot of potential
"""
fig,ax = plt.subplots(figsize=(6,6),num =5)
plt.xlabel(r'+ $x\longrightarrow$ ')
plt.ylabel(r'+ $y\longrightarrow$ ')
plt.title(r'Contour Plot of Potential $\phi$ ')
cs = ax.contour(X,Y,phi)
ax.scatter(ii[1]-(Nx-1)/2,ii[0]-(Ny-1)/2,marker = 'o', color ='r')
ax.clabel(cs, inline =1, fontsize = 9)

"""
```

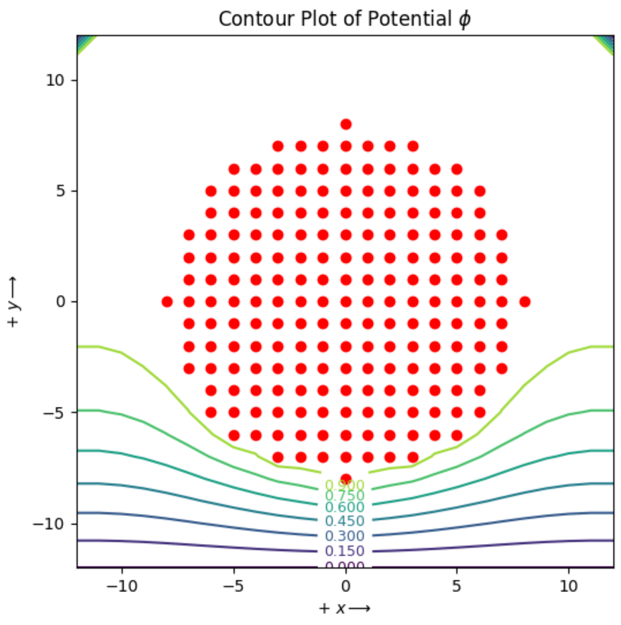


Figure 6: 2D contour plot of Potential

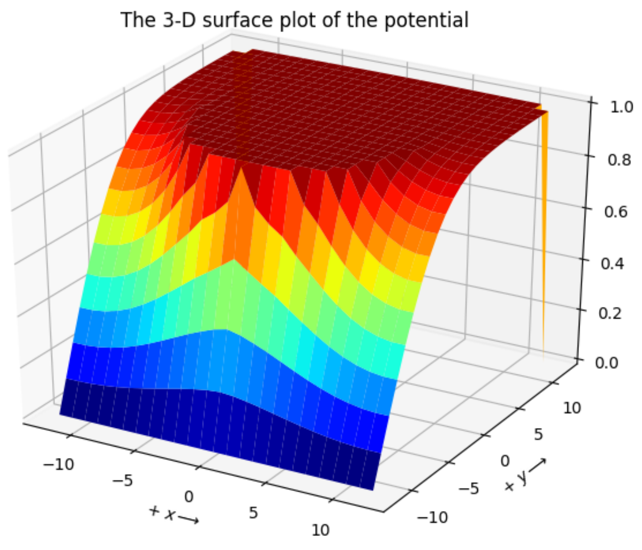


Figure 7: Surface Plot of Electric Potential

```
"""
Surface plot of potential
"""
fig1=plt.figure(6)       # open a new figure
ax=p3.Axes3D(fig1) # Axes3D is the means to do a surface plot
plt.title('The 3-D surface plot of the potential')
plt.xlabel(r'+ $x\longrightarrow$')
plt.ylabel(r'+ $y\longrightarrow$')
surf = ax.plot_surface(X, Y, phi, rstride=1, cstride=1, cmap=plt.cm.jet)
```

7 Finding and Plotting Current Density

Due to availability of potential at discrete points, the gradient equation can be used again as difference equation to solve for J_x and J_y

$$J_{x,ij} = 0.5 * (\phi_{i,j-1} - \phi_{i,j+1}) \tag{7}$$

$$J_{y,ij} = 0.5 * (\phi_{i-1,j} - \phi_{i+1,j}) \tag{8}$$

```
"""
Computing Current density J from the equations
"""
Jx = 0.5*(phi[1:-1,:-2]-phi[1:-1,2:])
# 0.5*(phi(x-1,y) - phi(x+1,y)) in cartesian
Jy = 0.5*(phi[2:,1:-1]-phi[:-2,1:-1])
# 0.5*(phi(x, y-1) - phi(x,y+1)) in cartesian
ax.quiver(X[1:-1,1:-1],Y[1:-1,1:-1],Jx,Jy)
ax.scatter(ii[1]-(Nx-1)/2,ii[0]-(Ny-1)/2,marker = 'o', color ='r')
plt.xlabel(r'+ $x\longrightarrow$')
plt.ylabel(r'+ $y\longrightarrow$')
plt.title('Vector plot of current flow')
```

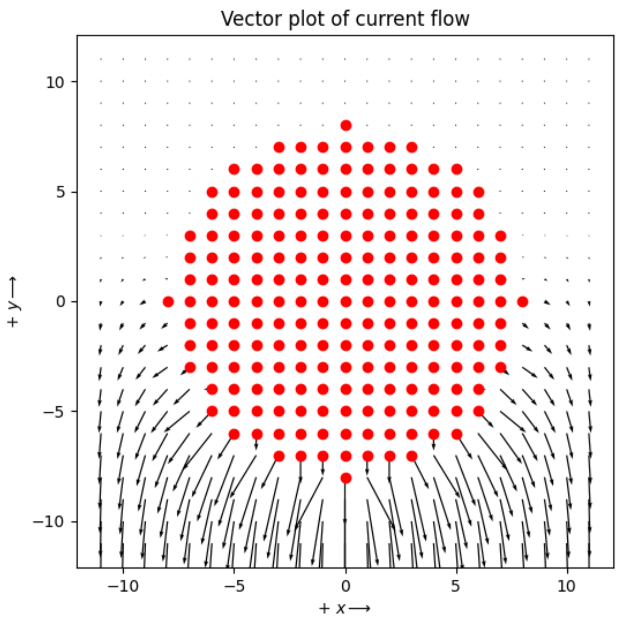


Figure 8: Vector Plot of Current Flow

Conclusion

We have approximated the Laplace equation to a difference equation. Sampling more number of points will lead to a more continuous potential distribution across the surface. This method to solve Laplace equation is known to be one of the worst due to it's low coefficient with which the error reduces w.r.t number of iterations. Also, the flow of current can be solved and the quiver plot gives us a visualization of the path of the current. We can observe that more current flows in the bottom half and very little on the top half. This is because the potential hasn't changed much in the top half whereas there is a larger gradient of potential in the bottom half (Figure 7). We can also make a conclusion that the bottom part (grounded side) gets more heated as more current flows through it.