Assignment 5: The Laplace Equation

Shekhar Diwakar [EE20B123]

March 11, 2022

Abstract

This report will discuss about the solver for the currents in a resistor and discusses about the current's dependency on the shape of the resistor and also discusses which part of the resistor is likely to get hottest.Here we analyse the currents in a square copper plate to which a wire is soldered to the middle of it.It also discuss about how to and stopping condition for the solver after certain iterations,and to model the errors obtained using Least Squares after analysing the actual errors in semilog and loglog plots.And finally we and the currents in the resistor after applying boundary conditions and analyse the vector plot of current ow and conclude which part of resistor will become hot.

1 Introduction

This report will discuss about the solver for the currents in a resistor, the current's dependency on the shape of the resistor and also which part of the resistor is likely to get hottest. Here we analyse the currents in a square copper plate to which a wire is soldered to the middle of it. We find the currents in the resistor after applying boundary conditions and analyse the vector plot of current flow and conclude which part of resistor will become hot.

- 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 sides are floating. The plate is 1 cm by 1 cm in size.
- To solve for currents in resistor, we use following equations and boundary conditions mentioned below:
- Conductivity (Differential form of ohm's law)

$$\vec{J} = \sigma \vec{E} \tag{1}$$

- Electric field is the gradient of the potential

$$\vec{E} = -\nabla\phi \tag{2}$$

- Charge Continuity equation is used to conserve the inflow and outflow charges

$$\nabla.\vec{J} = -\frac{\partial\rho}{\partial t} \tag{3}$$

- Combining the above equations above, we get

$$\nabla.(-\sigma\nabla\phi) = -\frac{\partial\rho}{\partial t} \tag{4}$$

- Assuming that our resistor contains a material of constant conductivity, the equation becomes

$$\nabla^2\phi = \frac{1}{\sigma}\frac{\partial\rho}{\partial t} \tag{5}$$

- For DC currents, the right side is zero, and we obtain

$$\nabla^2\phi = 0 \tag{6}$$

- Here we use a 2-D plate so the Numerical solutions in 2D can be easily transformed into a difference equation. The equation can be written out in

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0 \tag{7}$$

$$\frac{\partial\phi}{\partial x}_{(x_i,y_j)} = \frac{\phi(x_{i+1/2}, y_j) - \phi(x_{i-1/2}, y_j)}{\Delta x} \tag{8}$$

$$\frac{\partial^2\phi}{\partial x^2}_{(x_i,y_j)} = \frac{\phi(x_{i+1}, y_j) - 2\phi(x_i, y_j) + \phi(x_{i-1}, y_j)}{(\Delta x)^2} \tag{9}$$

- Using above equations we get

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4} \tag{10}$$

- Thus, the potential at any point should be the average of its neighbours. This is a very general result and the above calculation is just a special case of it. So the solution process is to take each point and replace the potential by the average of its neighbours. Keep iterating till the solution converges (i.e., the maximum change in elements of ϕ which is denoted by $error_k$ in the code ,where 'k' is the no of iteration, is less than some tolerance).
- At boundaries where the electrode is present, just put the value of potential itself. At boundaries where there is no electrode, the current should be tangential because charge can't leap out of the material into air. Since current is proportional to the Electric Field, what this means is the gradient of ϕ should be tangential. This is implemented by requiring that ϕ should not vary in the normal direction
- At last we solve for currents in the resistor using all these information.

2 Potential Array and Initialization

- Define the Parameters, The parameter values taken for my particular code were $N_x = 25$ and $N_y = 25$ and number of iterations : 1500
- Allocate the potential array as $\phi = 0$ .Note that the array should have N_y rows and N_x columns.
- To find the indices which lie inside the circle of radius 0.35 using meshgrid() by equation :

$$X^2 + Y^2 \leq 0.35^2 \tag{11}$$

- Then assign 1 V to those indices.

The python code snippet is as shown:

```
Nx=25 # size along x
Ny=25 # size along y
radius=8 #radius of central lead
Niter=1700 #number of iterations to perform

phi=np.zeros((Nx,Ny),dtype = float)
x,y=np.linspace(-0.5,0.5,num=Nx,dtype=float),np.linspace(-0.5,0.5,num=Ny,dtype=floa
Y,X=np.meshgrid(y,x,sparse=False)
phi[np.where(X**2+Y**2<(0.35)**2)]=1.0
```

3 Potential array updation and iteration

- Update the potential ϕ according to Equation below using vectorized code

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4} \tag{12}$$

- To apply Boundary Conditions where there is no electrode, the gradient of ϕ should be tangential. This is implemented by Equation given below , basically potential should not vary in the normal direction so we equate the last but row or column to outermost row or column correspondingly when applying boundary conditions for a side of plate,implemented using Vectorized code

$$\frac{\partial \phi}{\partial n} = 0 \tag{13}$$

The python code snippet is as shown:

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

4 The Error Calculations

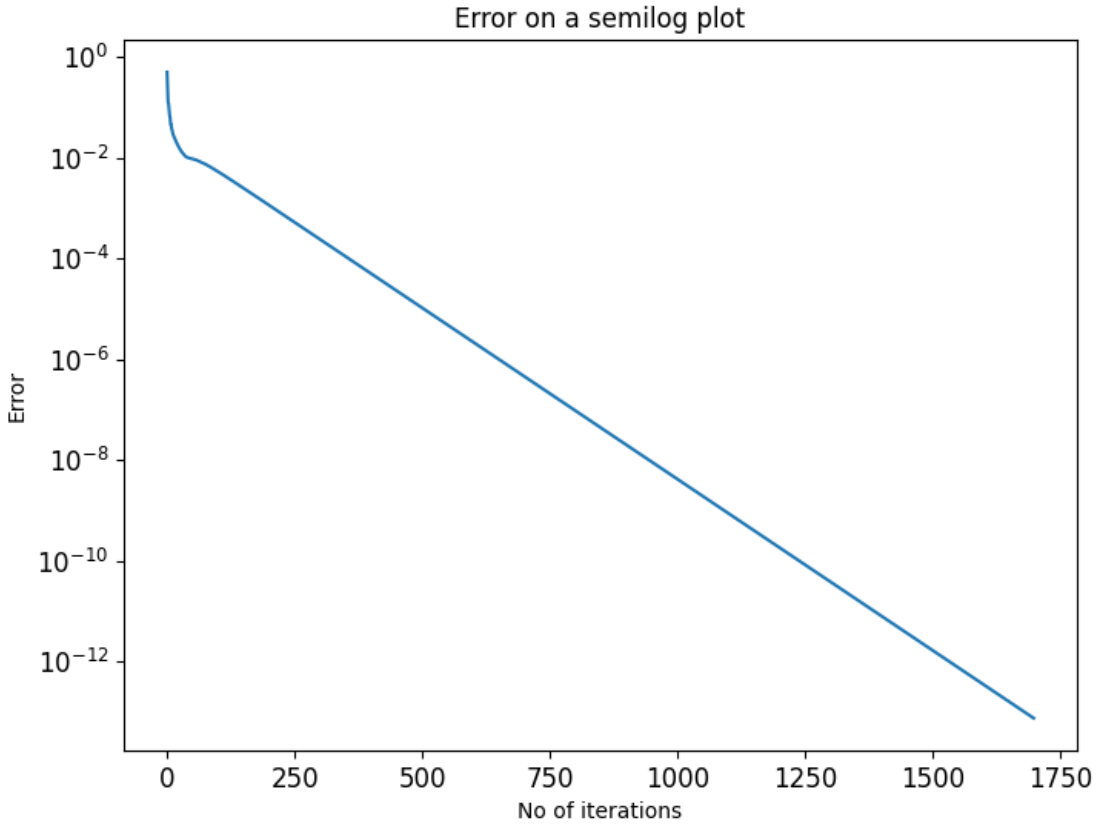
- The error calculated in the last section can be analysed by plotting it against the iteration number.
- This will give us a much more clearer picture of how the the error varies with respect to the iteration number.

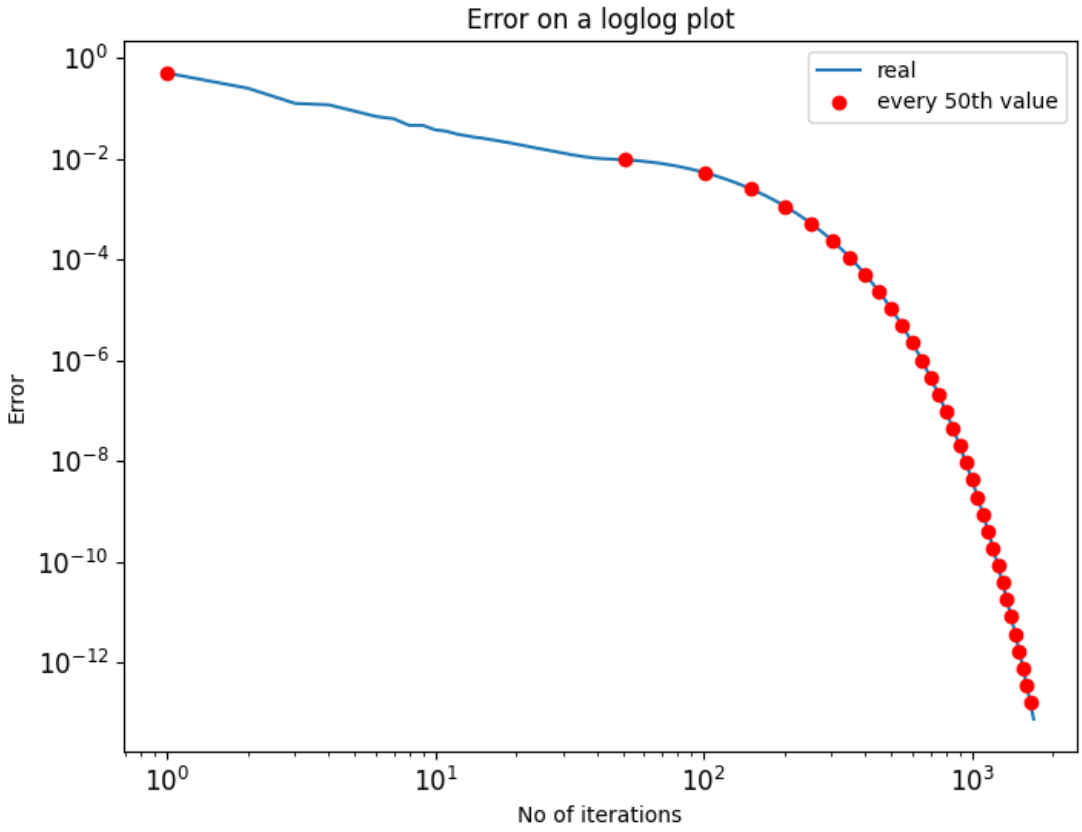
The python code snippet for plotting the graphs are as shown:

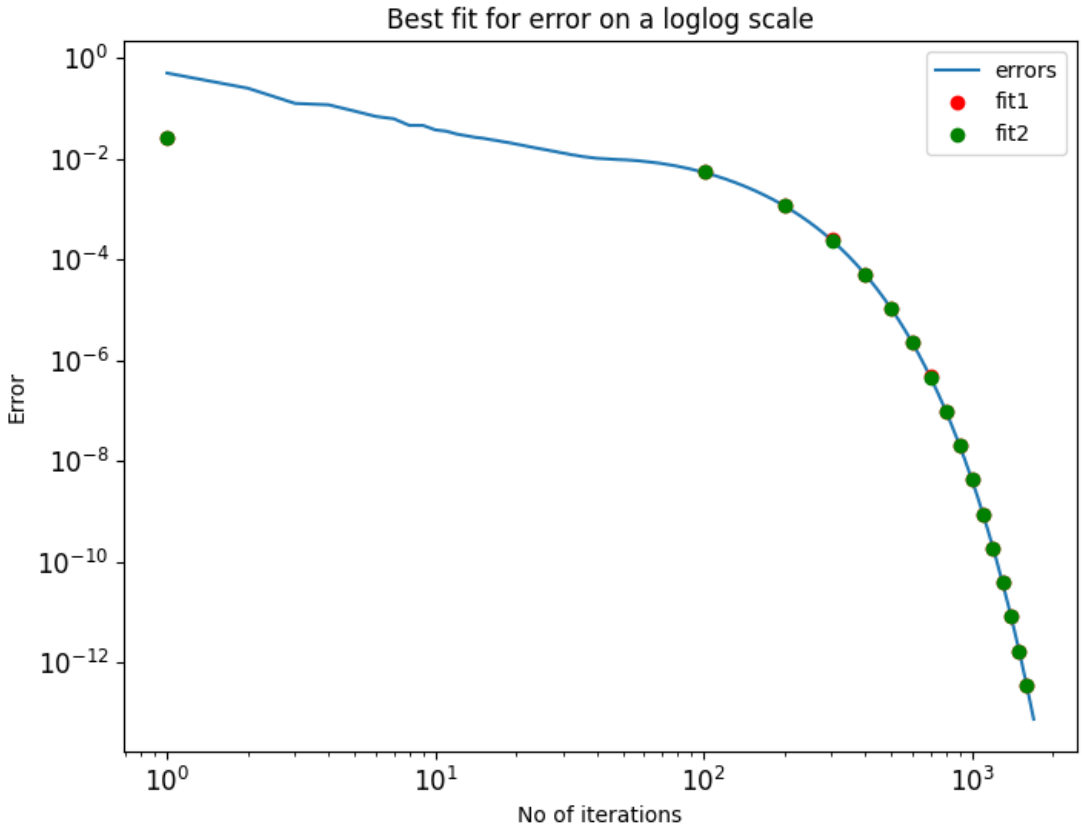
```
plt.figure(4, figsize=(8, 6))
    plt.title("Best fit for error on a loglog scale")
    plt.xlabel("No of iterations")
    plt.ylabel("Error")
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    x = np.asarray(range(Niter))+1
    plt.loglog(x,err)
    plt.loglog(x[::100],np.exp(a+b*np.asarray(range(Niter)))[::100],'ro')
    plt.loglog(x[::100],np.exp(a_+b_*np.asarray(range(Niter)))[::100],'go')
    plt.legend(["errors","fit1","fit2"])
    plt.savefig("suhas/figure4.png")
    plt.show()
    #now semilog
    plt.figure(5, figsize=(8, 6))
    plt.title("Best fit for error on a semilog scale")
    plt.xlabel("No of iterations")
    plt.ylabel("Error")
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.semilogy(x,err)
    plt.semilogy(x[::100],np.exp(a+b*np.asarray(range(Niter)))[::100],'ro')
    plt.semilogy(x[::100],np.exp(a_+b_*np.asarray(range(Niter)))[::100],'go')
    plt.legend(["errors","fit1","fit2"])
    plt.savefig("suhas/figure5.png")
    plt.show()


#plotting Error on semilog
plt.figure(2, figsize=(8, 6))
plt.title("Error on a semilog plot")
plt.xlabel("No of iterations")
plt.ylabel("Error")
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.semilogy(range(Niter),err)
plt.savefig("suhas/figure2.png")
plt.show()

#plotting Error on loglog
plt.figure(3, figsize=(8, 6))
plt.title("Error on a loglog plot")
```

```
plt.xlabel("No of iterations")
plt.ylabel("Error")
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.loglog((np.asarray(range(Niter))+1),err)
plt.loglog((np.asarray(range(Niter))+1)[::50],err[::50],'ro')
plt.legend(["real","every 50th value"])
plt.savefig("suhas/figure3.png")
plt.show()
```

The respective error plots are as shown:




Error on a loglog plot
real
every 50th value
Error
No of iterations
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{0}
10^{1}
10^{2}
10^{3}



Extracting the exponential

- We can find the fit using Least squares for all iterations separately and compare them.
- As we know that error follows Ae^{Bx} at large iterations, we use equation given below to fit the errors using least squares.

$$logy = logA + Bx \tag{14}$$

- We can find the constants of the error function obtained for the two cases using lstsq and compare them.

The python code snippet to extract the exponent is as follows:

```
c_approx_500 =
```

```
lstsq(c_[ones(Niter-500),arange(500,Niter)],log(errors[500:]),rcond=None)
a_500,b_500 = exp(c_approx_500[0][0]),c_approx_500[0][1]

c_approx = lstsq(c_[ones(Niter),arange(Niter)],log(errors),rcond=None)
a, b = exp(c_approx[0][0]), c_approx[0][1]
```

The plots comparing both the errors are as shown:

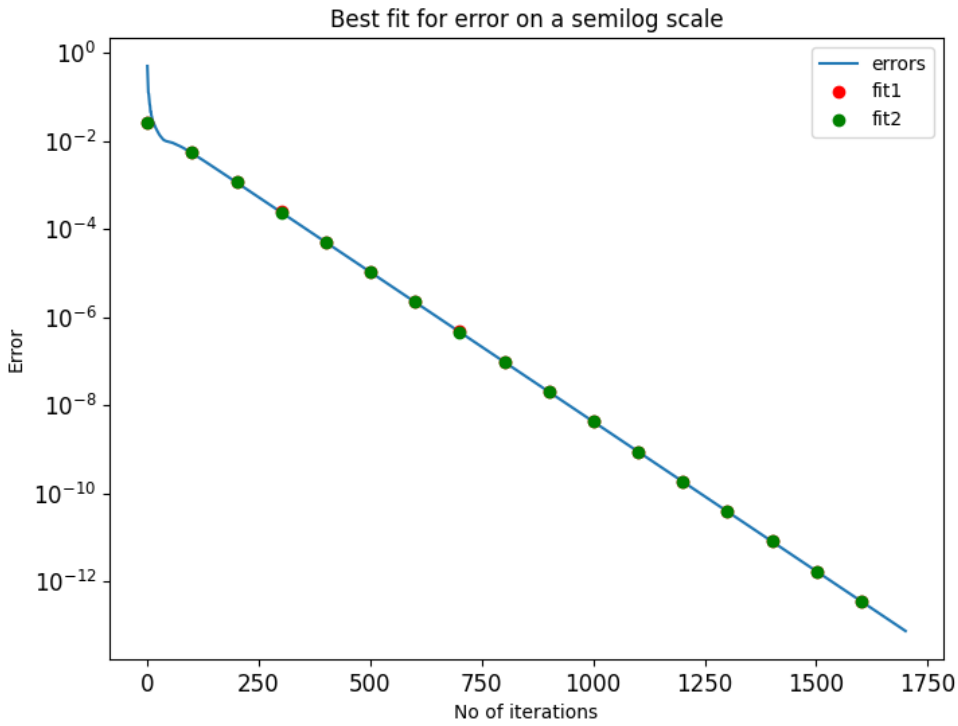


Figure 1: Expected versus actual error (above 500 iterations)

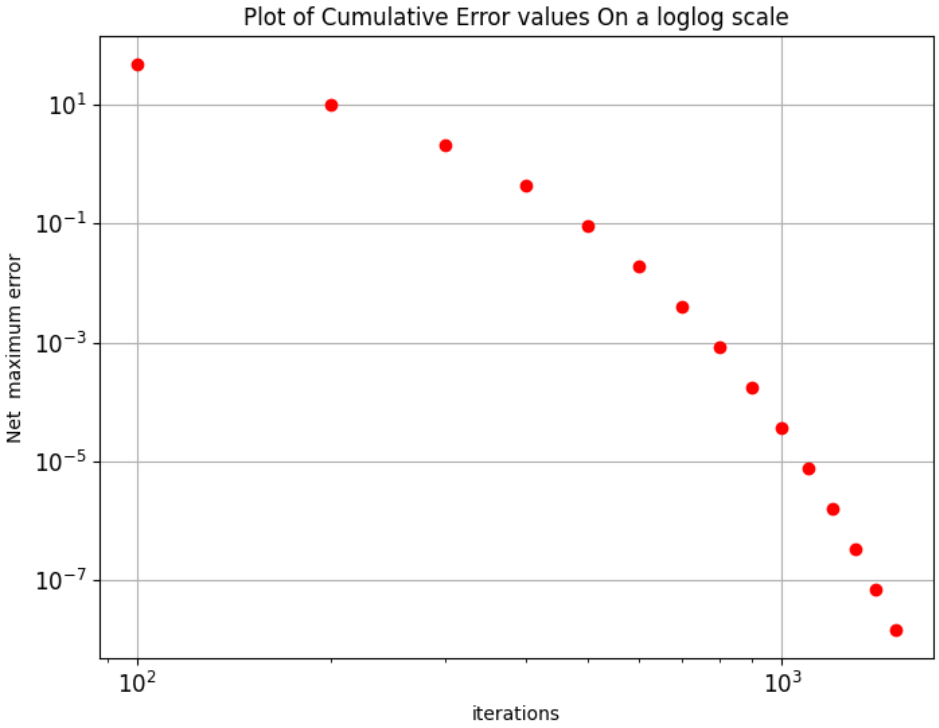


Figure 2: Expected versus actual error

Upon execution of the code, we can observe that

- $A_{500} = 0.02604399$ and $B_{500} = -0.01564807$
- $A = 0.02621557$ and $B = -0.01565526$

5 Surface and Contour Plot of Potential

We can analyse the potential variations by plotting it as a surface and contour plot. The following python code explains it.

```
#plotting 2d contour of final potential
plt.figure(7, figsize=(8, 6))
plt.title("2D Contour plot of potential")
plt.xlabel("X")
plt.ylabel("Y")
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
x_c,y_c=np.where(X**2+Y**2<(0.35)**2)
```

```
plt.plot((x_c-Nx/2)/Nx,(y_c-Ny/2)/Ny,'ro')
plt.contourf(Y,X[::-1],phi)
plt.colorbar()
plt.savefig("suhas/figure7.png")
plt.show()

#plotting 3d contour of final potential
fig1=plt.figure(8)     # 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.xticks(fontsize=12)
plt.yticks(fontsize=12)
surf = ax.plot_surface(Y, X, phi.T, rstride=1, cstride=1, cmap=plt.cm.jet)
plt.savefig("suhas/figure8.png")
plt.show()
```

The surface and contour plots are as shown below:

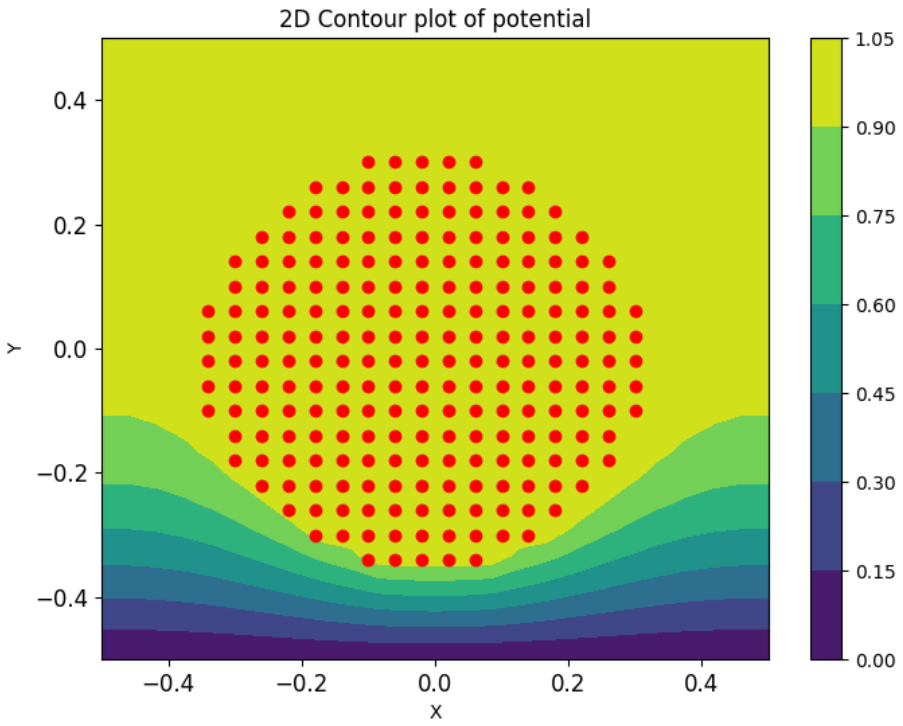


Figure 3: Contour plot of potential

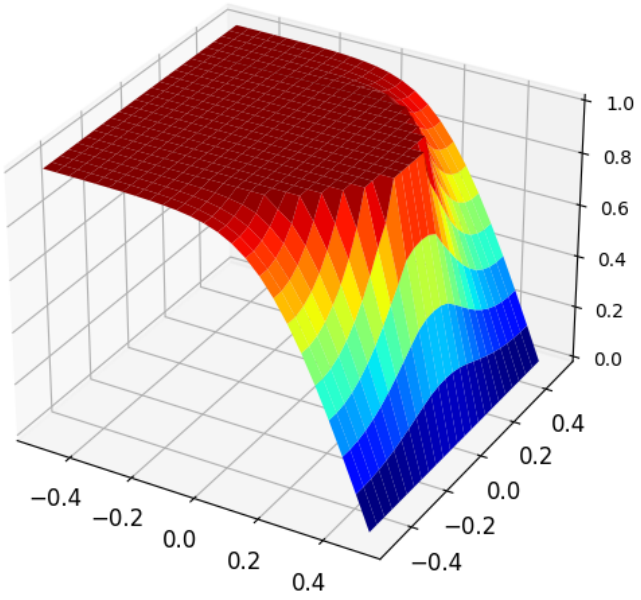


Figure 4: 3-D Surface potential plot of potential

6 Vector Plot of Currents

- We can obtain the currents by computing the gradient.

  The python code snippet for calculating the the current densities and plotting them are as follows:

```
#plotting current density
plt.figure(9, figsize=(8, 6))
plt.title("Vector plot of current flow")
plt.quiver(Y[1:-1,1:-1],-X[1:-1,1:-1],-Jx[:,::-1],-Jy)
x_c,y_c=np.where(X**2+Y**2<(0.35)**2)
plt.plot((x_c-Nx/2)/Nx,(y_c-Ny/2)/Ny,'ro')
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.savefig("suhas/figure9.png")
plt.show()
```

The vector plot of the current flow along with the potential is as shown below:

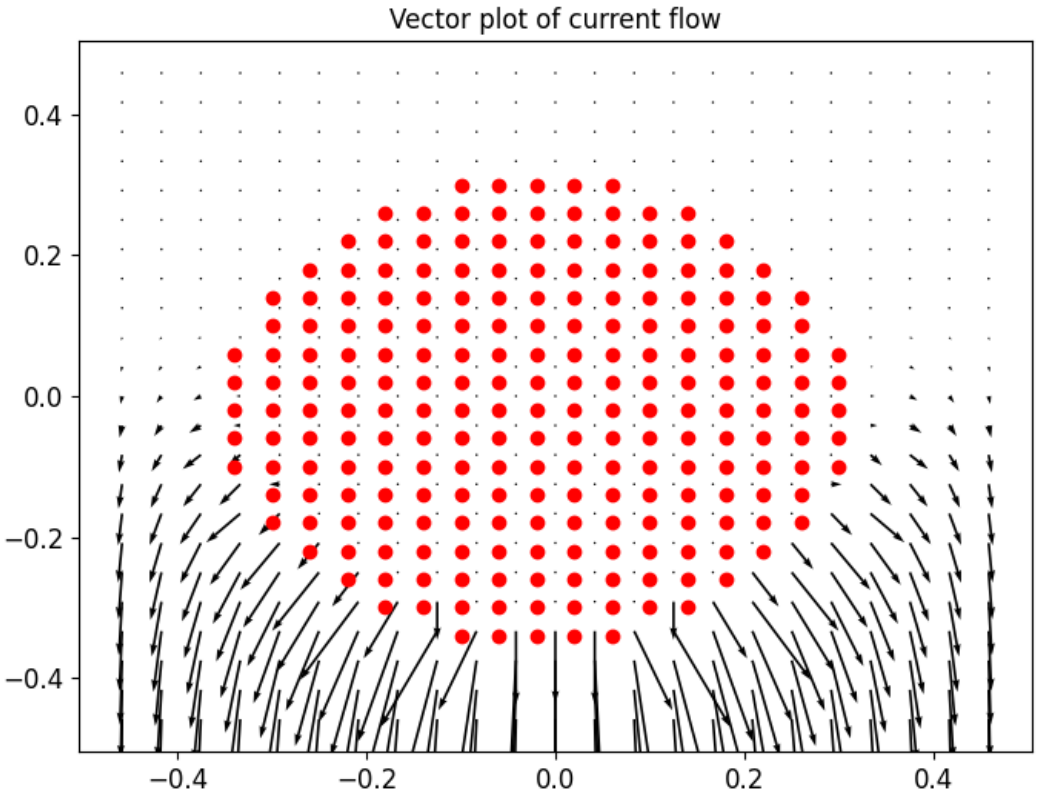


Figure 5: Vector plot of current flow

- So as we noted that the potential gradient was higher in down region of the plate, and we know that Electric field is the gradient of the potential as given below

$$\vec{E} = -\nabla\phi \tag{15}$$

- So $\vec{E}$ is larger where there is potential gradient is high and is inverted since it is negative of the gradient!, So it is higher in down region which is closer to bottom plate which is grounded
- And we know that

$$\vec{J} = \sigma \vec{E} \tag{16}$$

- So $\vec{J}$ is higher and perpendicular to equipotential electrode region i.e "Red dotted region" so the current is larger in down part of the plate and perpendicular to the red dotted electrode region since $I = \vec{J}.\vec{A}$
- This is the reason most of the current flows from electrode to the bottom plate which is grounded because of higher potential gradient.
- And there is almost zero current in upper part of the plate since there is not much potential gradient as we observed from the surface and contour plot of the potential ϕ

7 Conclusion :

• To conclude , Most of the current is in the narrow region at the bottom.So that is what will get strongly heated. Since there is almost no current in the upper region of plate,the bottom part of the plate gets hotter and temperature increases in down region of the plate. And we know that heat generated is from J: E (ohmic loss) so since J and E are higher in the bottom region of the plate, there will more heat generation and temperature rise will be present. So overall we looked the modelling of the currents in resistor in this report ,and we observe that the best method to solve this is to increase Nx and Ny to very high values(100 or 100)and increase the no of iterations too, so that we get accurate answers i.e currents in the resistor. But the tradeo is this method of solving is very slow even though we use vectorized code because the decrease in errors is very slow w.r.t no of iterations.