Assignment 5: The Laplace Equation

Hariharan P EE20B042

March 8, 2022

Brief up:

The tasks of this assignment are the following:

- To solve for currents in a resistor,
- To know which part of the resistor is likely to get hottest,
- To plot graphs to understand the 2-D Laplace equation.

1 Introduction

The current depends on the shape of the resistor and part of the resistor is likely to get hottest. We calculate the currents in the resistor by 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:

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

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

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

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

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

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

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

- Using above equations we get, for a 2-D plate

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4} \tag{7}$$

- The potential at any point is given by the average of its neighbours. 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.

2 Allocating the potential array and initializing:

- Get the parameters from user or the code uses the default parameters. The default parameter values taken are $N_x = 25$ and $N_y = 25$ and number of iterations : 1500
- Allocate the potential array as $\phi = 0$
- To find the indices which lie inside the circle of radius 0.35 using meshgrid() by equation :

$$X^2 + Y^2 \leq 0.35^2 \tag{8}$$

- Then assign 1 V to those indices.

The python code snippet is as shown:

```
#getting inputs - default/user
if(len(sys.argv)==5):
    Nx=int(sys.argv[1])
    Ny=int(sys.argv[2])
    radius=int(sys.argv[3])
    Niter=int(sys.argv[4])
    print("user provided parameters are being used")
else:
```

```
Nx=25
Ny=25
radius=8
Niter=1500
print("Using default values")

#Allocate the potential array and initialize it
x,y=linspace(-0.5,0.5,num=Nx,dtype=float),
linspace(-0.5,0.5,num=Ny,dtype=float)
X,Y = meshgrid(x,-y)
ii=where(X**2+Y**2<(0.35)**2)
phi=zeros((Nx,Ny),dtype = float)
phi[ii]=1.0
```

The plot for the initial potential contour is as shown:

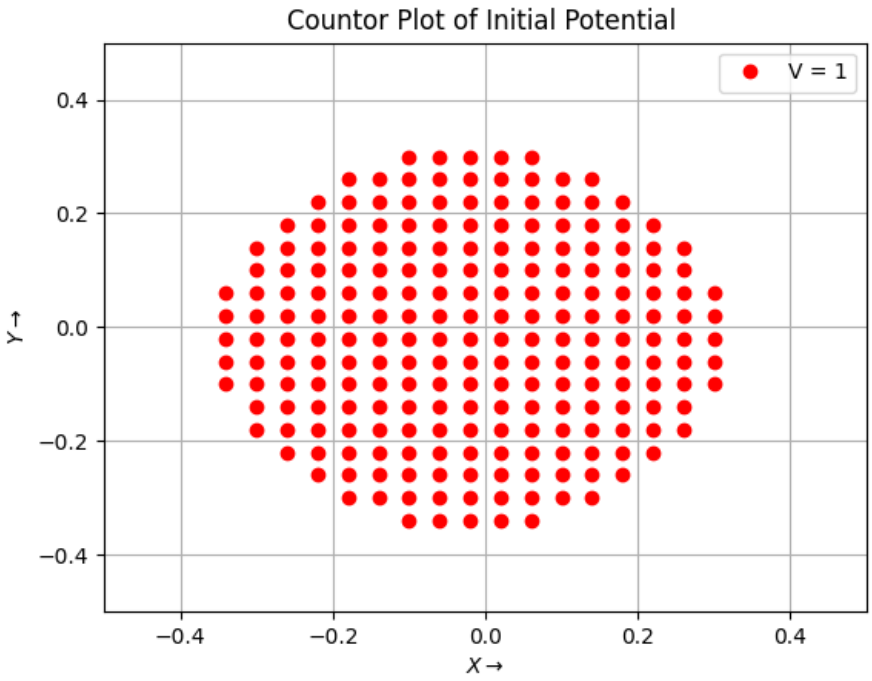


Figure 1: Contour plot of initial potential

3 Iterating and updating the potential array:

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

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4} \tag{9}$$

- To apply Boundary Conditions, the gradient of ϕ should be tangential, where there is no electrode. This is implemented by Equation given below .

$$\frac{\partial \phi}{\partial n} = 0 \tag{10}$$

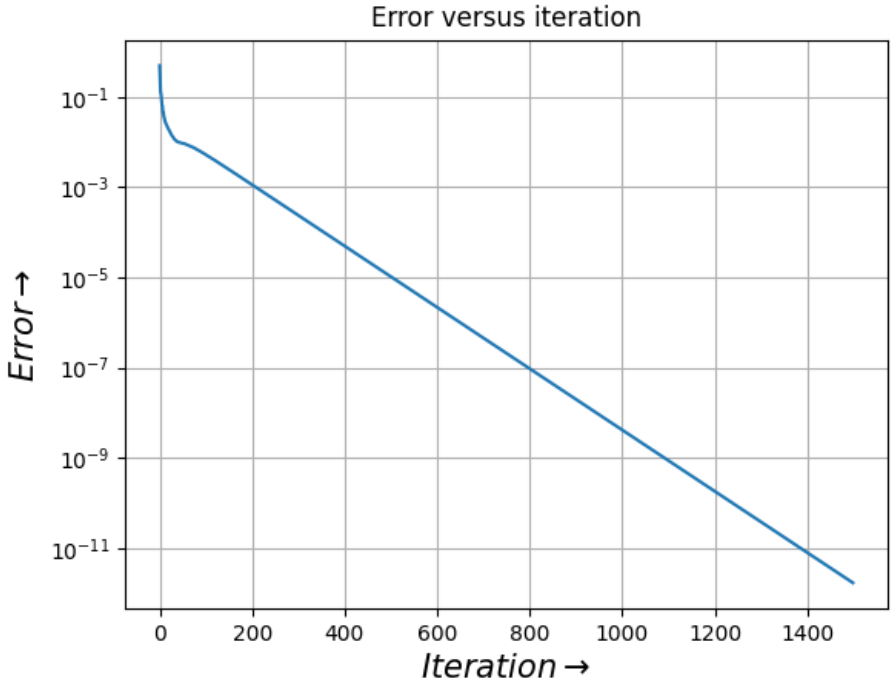
The python code snippet is as shown:

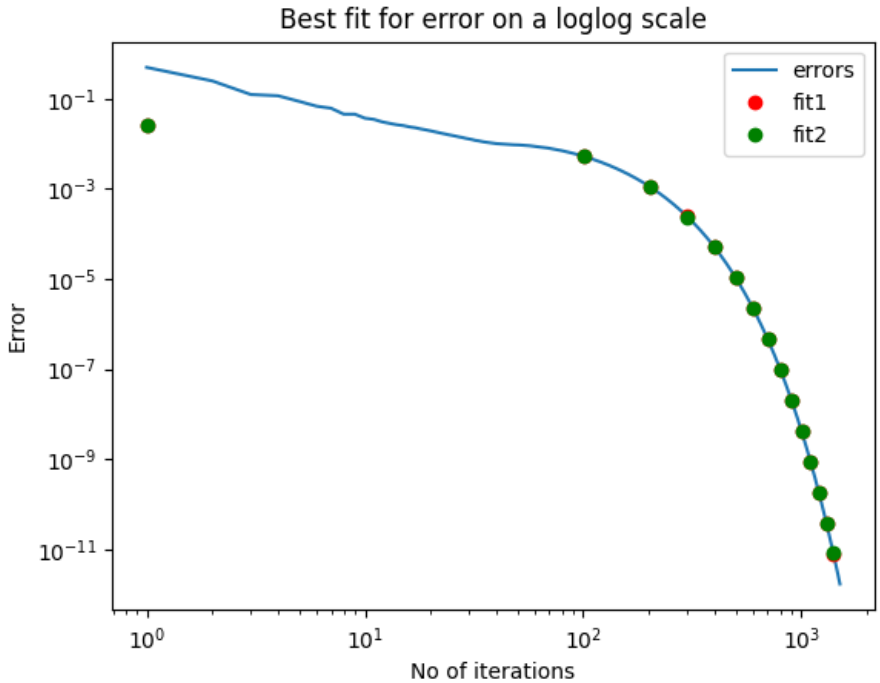
```
errors = empty(Niter)
for k in range(Niter):
        oldphi = phi.copy()
        phi[1:-1,1:-1] = 0.25*(phi[1:-1,0:-2] + phi[1:-1,2:]
    + phi[0:-2,1:-1] + phi[2:,1:-1])
        phi[1:-1,0] = phi[1:-1,1]
        phi[1:-1,-1] = phi[1:-1,-2]
        phi[0,1:-1] = phi[1,1:-1]
        phi[ii] = 1.0
        errors[k]=(abs(phi-oldphi)).max();
```

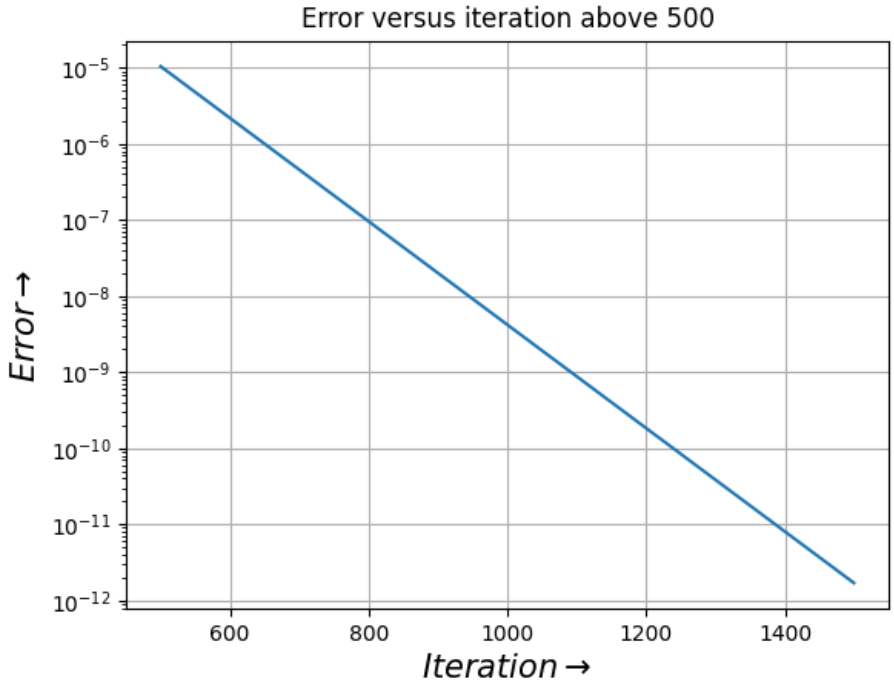
4 Analysing and Plotting errors:

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

The respective error plots are as shown:







Extracting the exponent parts

- We 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.

$$logy = logA + Bx \tag{11}$$

- 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:

```
#exponent part of the error values
c_approx = lstsq(c_[ones(Niter),arange(Niter)]
,log(errors),rcond=None)
a, b = exp(c_approx[0][0]), c_approx[0][1]
print(" A and B : ",a,b)
c_500 = lstsq(c_[ones(Niter-500),arange(500,Niter)]
,log(errors[500:]),rcond=None)
a__,b__ = exp(c_500[0][0]),c_500[0][1]
print("A and B for the iterations after 500 : ",a__,b__)
```

The plots comparing both the errors are as shown:

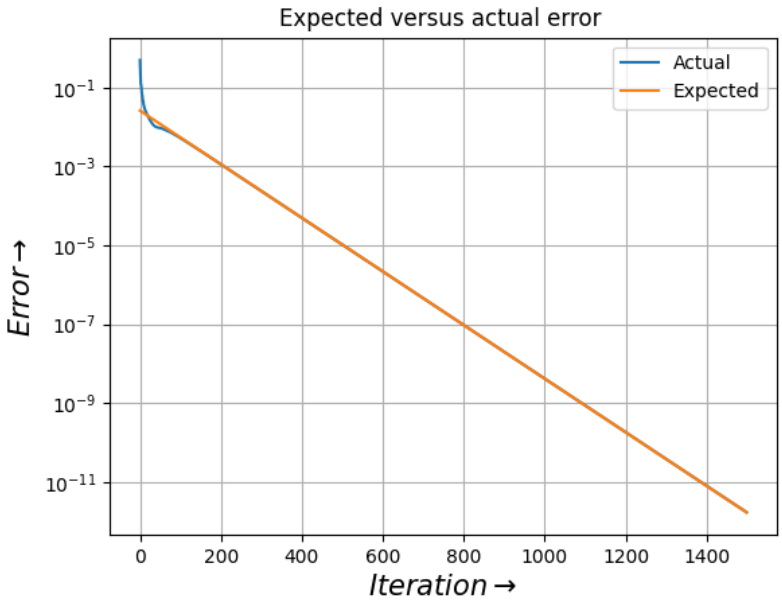


Figure 2: Expected versus actual error

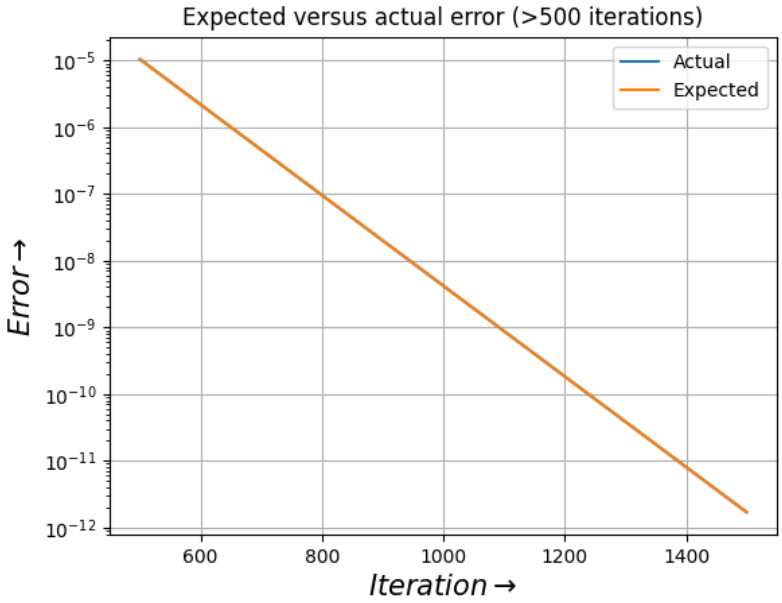


Figure 3: Expected versus actual error (above 500 iterations)

Upon execution of the code, we can observe that

- $A_{500} = 0.026043987460168727$ and $B_{500} = -0.015648069369668782$
- $A = 0.026215571110278363$ and $B = -0.015655264062372592$

Fitting the error

We infer that the error is decaying exponentially for higher iterations from the plots.The tasks is to plot the two fits as well. One considering all the iterations(fit1) and another without considering the first 500 iterations(fit2). There is very little difference between the two fits.

```
def best_fit(y,Niter,lastn=0):
    log_err = log(y)[-lastn:]
    X = vstack([(np.arange(Niter)+1)[-lastn:],ones(log_err.shape)]).T
    log_err = reshape(log_err,(1,log_err.shape[0])).T
    return lstsq(X, log_err,rcond=None)[0]
#computing the Cumulative error
def cummu_error(a,b,Niter):
    return -a/b*exp(b*(Niter+0.5))

b,a = best_fit(errors,Niter)
b_,a_ = best_fit(errors,Niter,500)
```

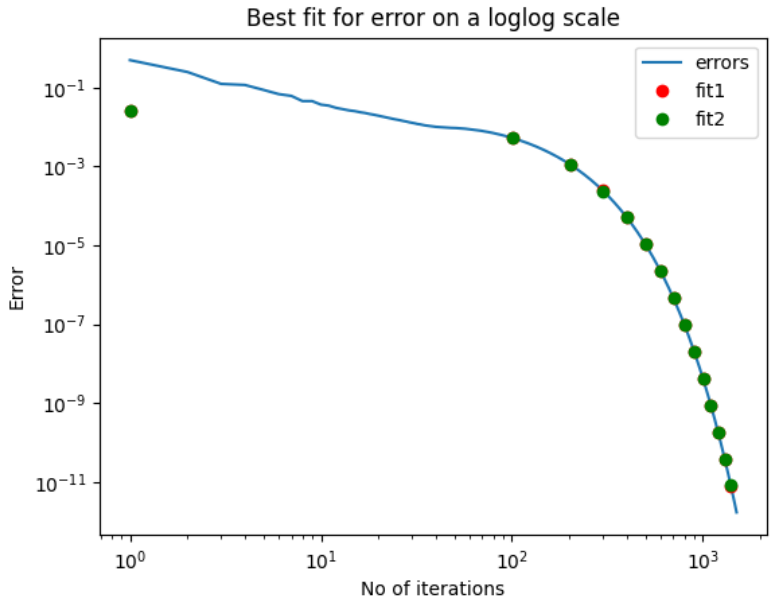


Figure 4: Best Fit of error

Plotting cumulative Error

The maximum error scales as error = Aexp(Bk). 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.

```
iteration=arange(100,Niter+1,100)
figure(3)
grid(True)
title(r'Plot of Cumulative Error values On a loglog scale')
loglog(iteration,abs(cummu_error(a_,b_,iteration)),'ro')
xlabel("iterations")
ylabel("Net  maximum error")
show()
```

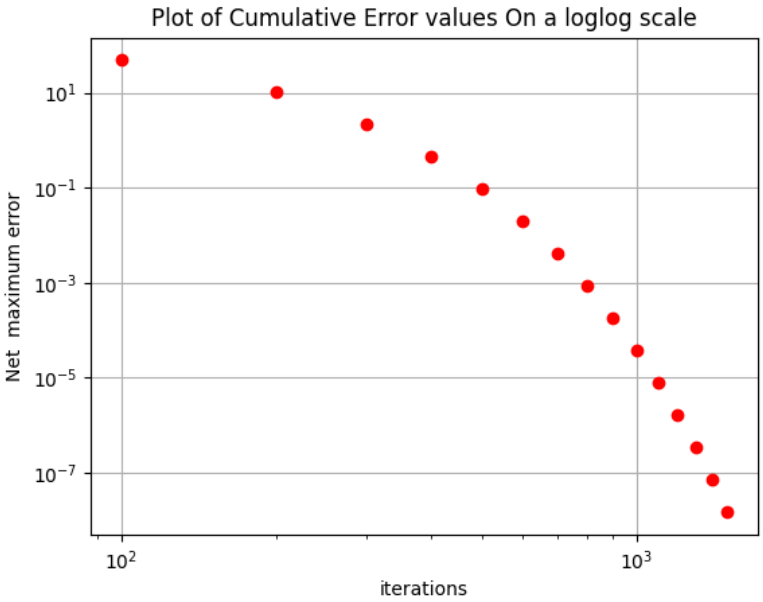


Figure 5: cumulative error values

5 Contour and Surface Plots of Potential:

We analyse the variation of potential by plotting it as a contour and surface plot. The following python code shows that.

```
#the contour of phi
figure(8)
contourf(X,Y,phi)
```

```
plot((ii[0]-Nx/2)/Nx,(ii[1]-Ny/2)/Ny,'ro',label="V = 1")
title("Contour plot of potential")
xlabel(r'$X\rightarrow$')
ylabel(r'$Y\rightarrow$')
colorbar()
grid(True)
legend()

#the surface plots of phi
fig1=figure(9)
ax=p3.Axes3D(fig1)
title("The 3-D surface plot of the potential")
xlabel(r'$X\rightarrow$')
ylabel(r'$Y\rightarrow$')
surf = ax.plot_surface(X, Y, phi, rstride=1, cstride=1, cmap=cm.jet)
fig1.colorbar(surf)
```

The contour and surface plots are as shown below:

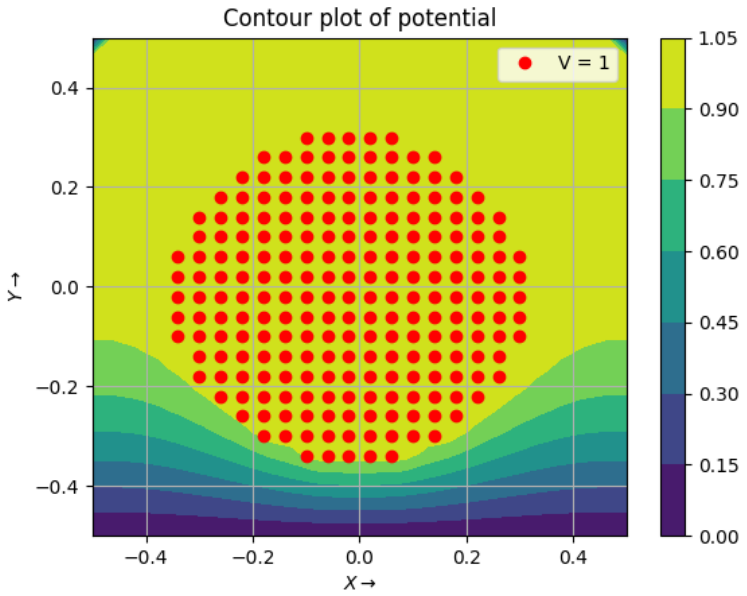


Figure 6: Contour plot of potential

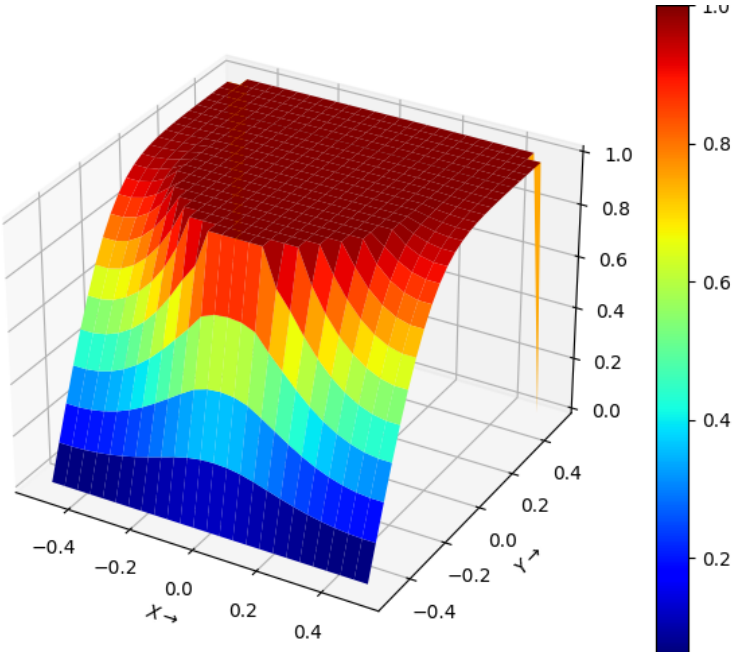


Figure 7: 3-D Surface potential plot of potential

6 Current flow:

- We compute the current flow by computing the gradient.
- The actual value of σ does not matter to the shape of the current profile, so we set it to unity. Our equations are

$$J_x = -\frac{\partial \phi}{\partial x} \tag{12}$$

$$J_y = -\frac{\partial \phi}{\partial y} \tag{13}$$

$$J_{x,ij} = \frac{1}{2}(\phi_{i,j-1} - \phi_{i,j+1}) \tag{14}$$

$$J_{y,ij} = \frac{1}{2}(\phi_{i-1,j} - \phi_{i+1,j}) \tag{15}$$

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

```
# The current densities are calculated
Jx = zeros((Ny, Nx))
Jy = zeros((Ny, Nx))
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])
```

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

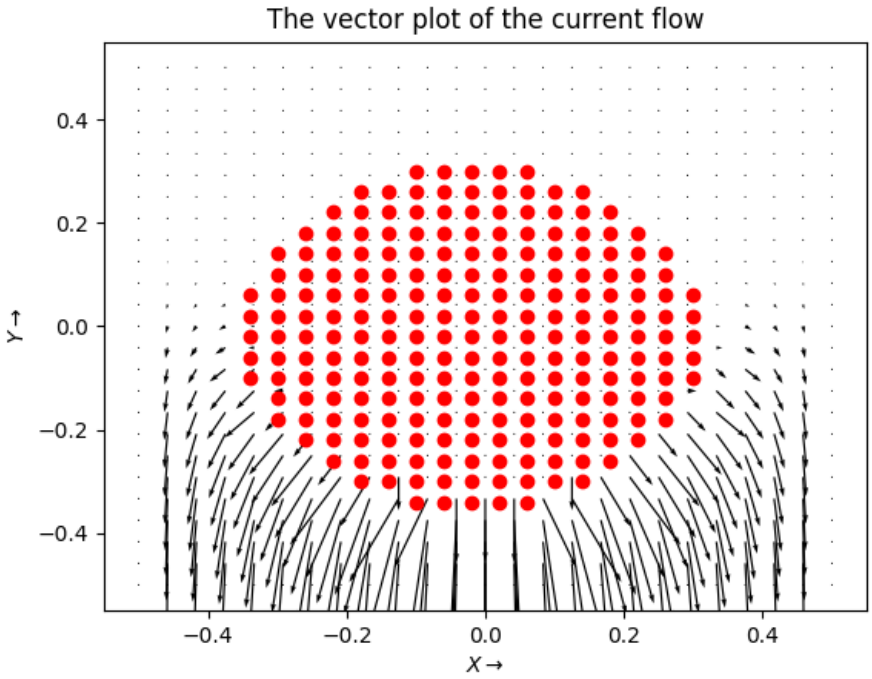


Figure 8: Vector plot of current flow

- As we see , the potential gradient was higher in down region of the plate, and Electric field is the gradient of the potential
- 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 equi-potential electrode region so the current is larger in down part of the plate and perpendicular to the red dotted electrode region since $I = \vec{J}.\vec{A}$
- Thus most of the current flows from electrode to the bottom plate which is grounded because of higher potential gradient.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 :

- Most of the current is at the bottom. So this region will get strongly heated.
- Heat is generated from $\vec{J}.\vec{E}$ (ohmic loss) , and $\vec{J}$ and $\vec{E}$ are higher in the bottom region of the plate, there will be increase in temperature.
- Thus we observe that the best method to solve this is to increase N_x and N_y to very high values (100 or ≥ 100) and increase the number of iterations, so that we could get almost precise currents in the resistor.
- 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.