EE2703 : Applied Programming Lab
Assignment 5
Laplace equations

Keerthana Rachuri
EE20B102

March 11, 2022

1 AIM:

The aim of this assignment is to: 1. Find the potential on each point on the Resistor through Laplace's equation by numerical methods. 2. Find the current current profile through the Resistor. 3. Find the temperature profile of the Resistor due to Joule's heating

2 DEFINING THE PARAMETERS:

We initialise the parameters with their default values. The user can modify them with commandline arguments if he wishes to.

```
    if len(sys.argv)==5:
    try:
        Nx = int(sys.argv[1])
        Ny = int(sys.argv[2])
        radius = float(sys.argv[3])
        Niter = int(sys.argv[4])
    except ValueError:
        print("Input arguments must be integers. Radius can be float too")
        exit()
```

3 INITIALISATION:

We create a 2D array of shape Ny x Nx 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))
x = np.arange(Nx) - (Nx-1)/2
y = np.flip(np.arange(Ny) - (Ny-1)/2)
X,Y = np.meshgrid(x,y)
ii = np.where(X*X + Y*Y <= radius**2)

phi[ii] =1
```

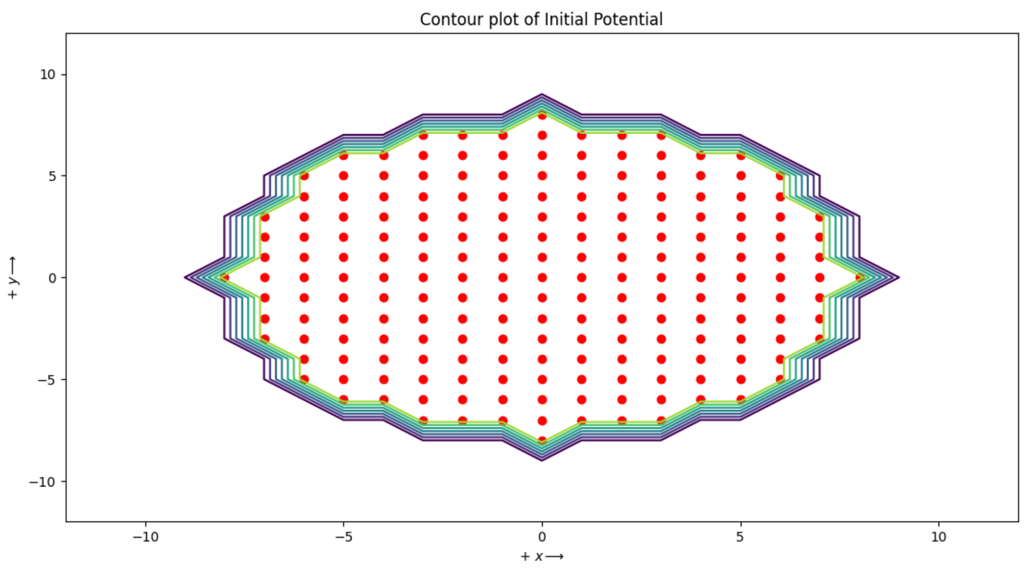

Figure 1.1: Electric potential initial values

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()
```

• 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: i,j = 0.25 (i,j1 + i,j+1 + i+1,j + i1,j)

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 . We know that one side is grounded i.e., = 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
    phi[1:-1,0] = phi[1:-1,1]
    phi[1:-1,-1] = phi[1:-1,-2]
    phi[0,1:-1] = phi[1,1:-1]
```

• Calculating Error

```python
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 discourage

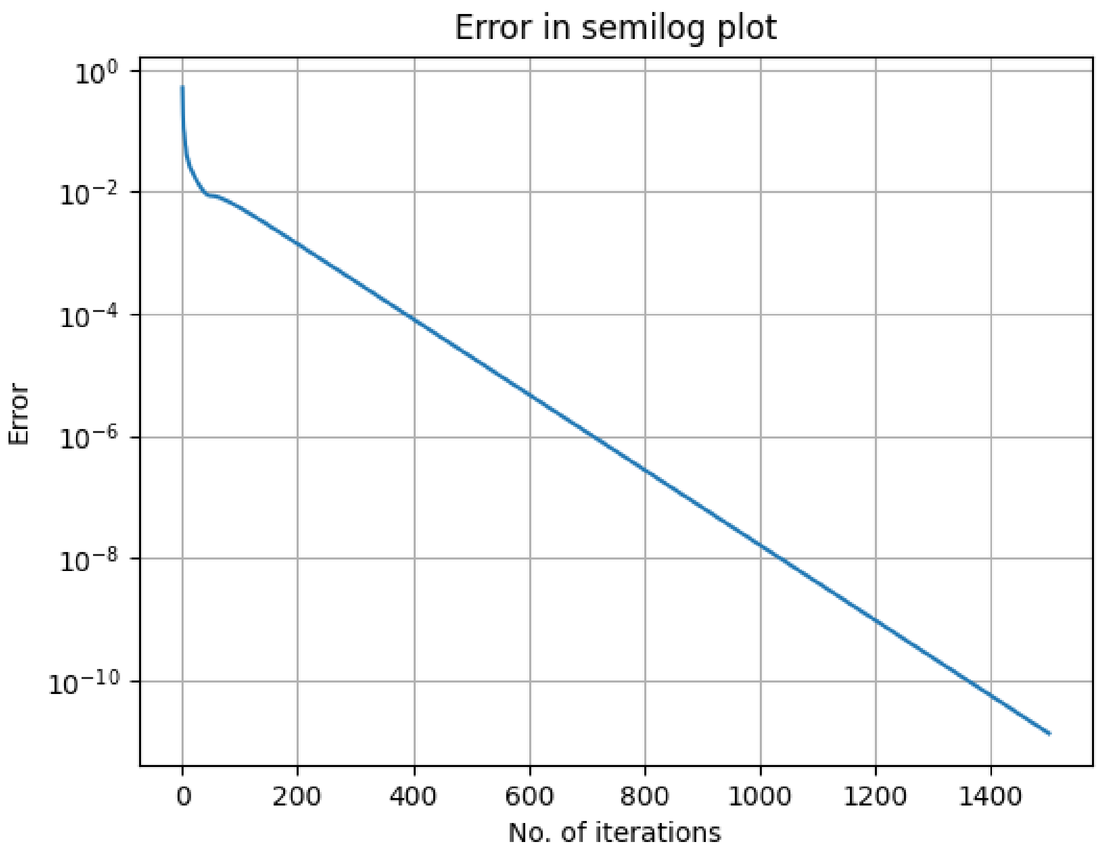

Figure 1.2: semilog plot

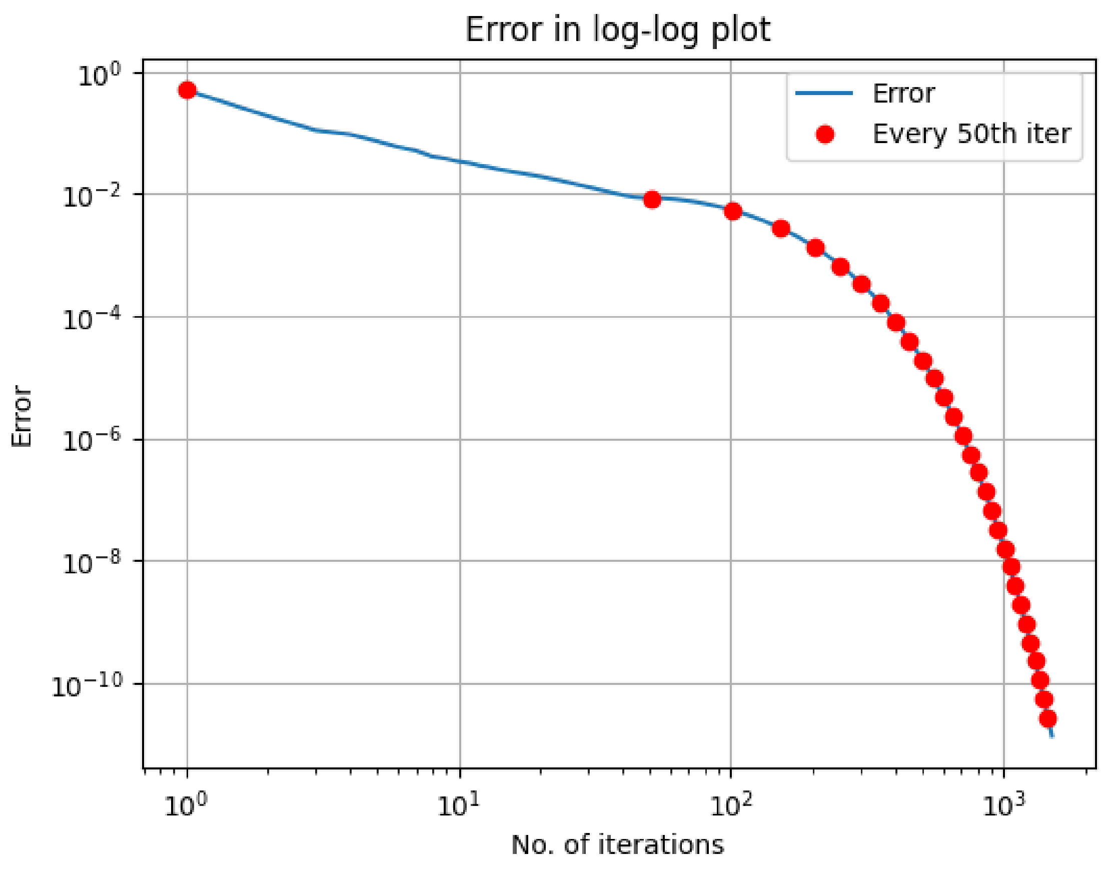

Figure 1.3: loglog plot

5.2 Fitting the error:

rom Figure 1.3 and 1.4, 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 500 th iteration.

```
i = np.arange(1,Niter+1)
one_array = np.ones(Niter)
M = np.c_[one_array,i]
M_500 = np.c_[one_array[500:],i[500:]]
v = slg.lstsq(M,np.log(errors))[0]
v_500 = slg.lstsq(M_500,np.log(errors[500:]))[0]
```

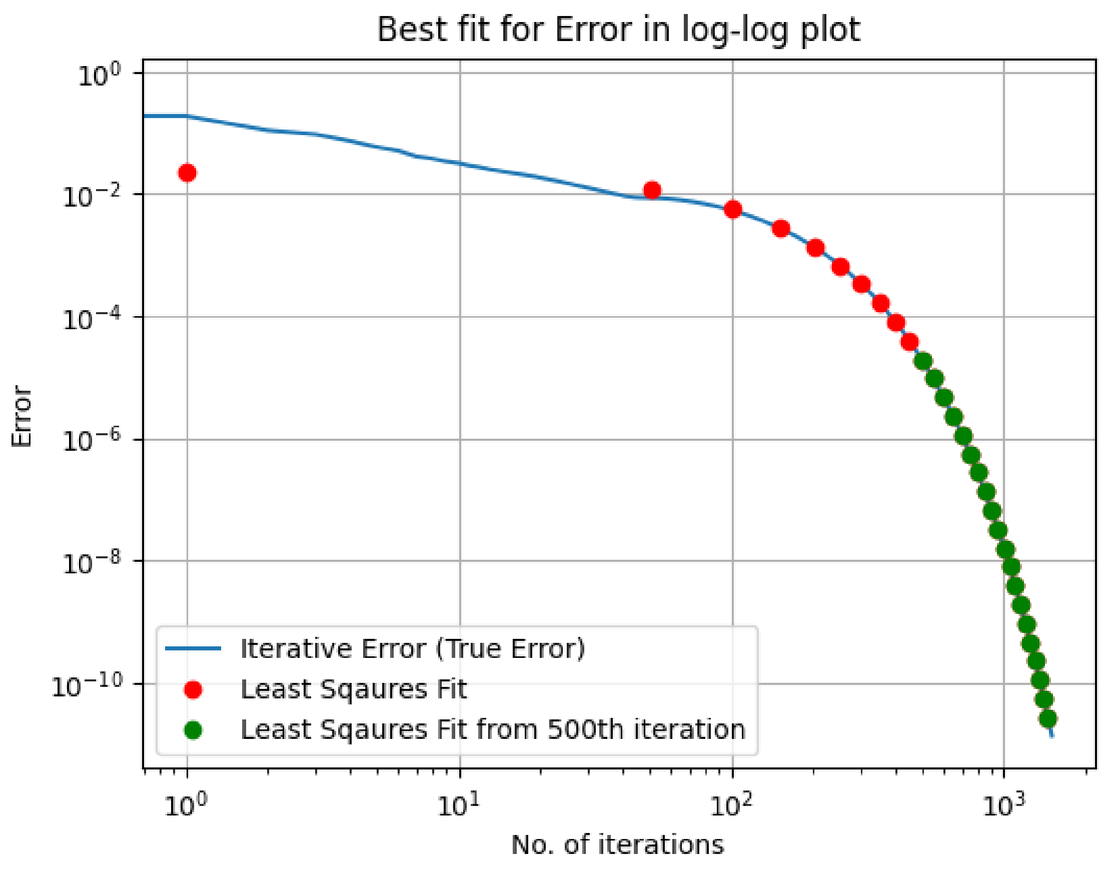

Figure 1.4: no of iterations

Figure 1.5 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.

```python
def cum_error(x):
     return -A/B*np.exp(B*(x+0.5))
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()
```

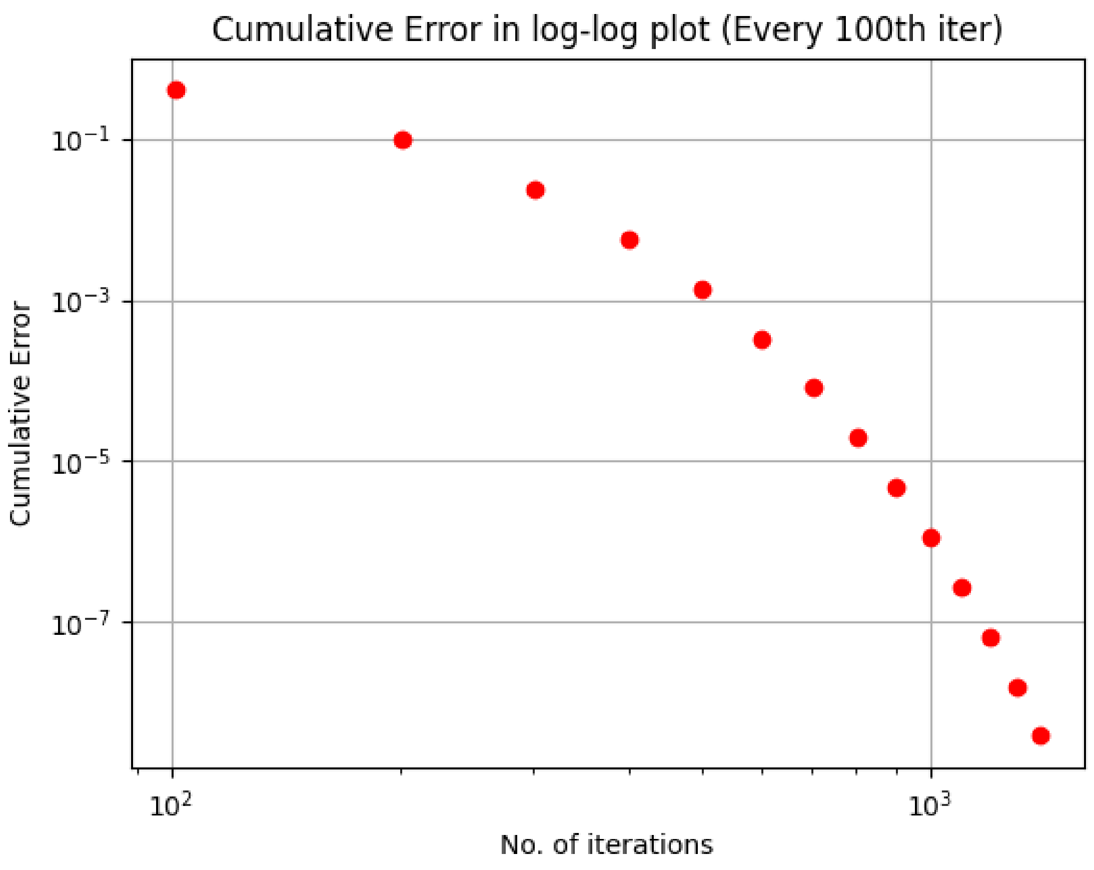

Figure 1.5: cumulative error on log scale

6 Plotting Electric Potential :

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

```
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)
```

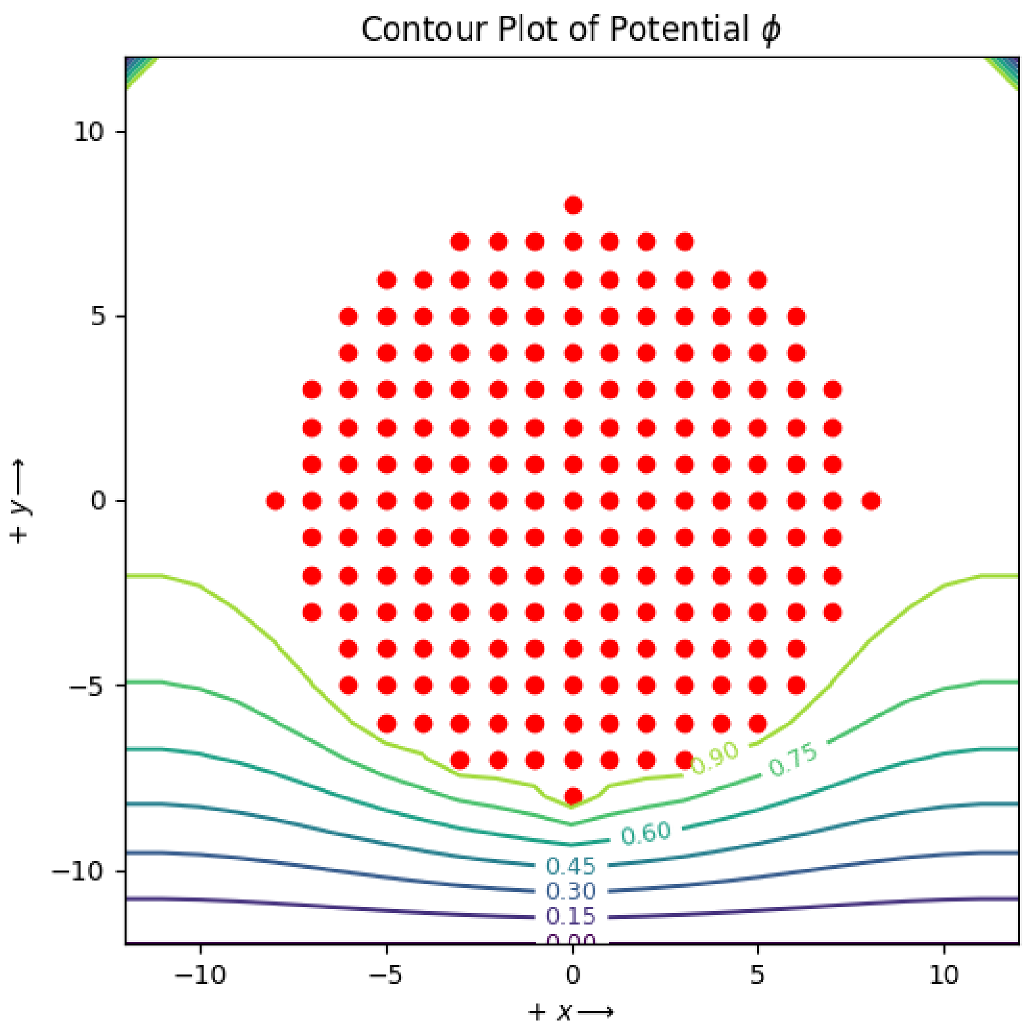

Figure 1.6: 2D contour plot of Potential

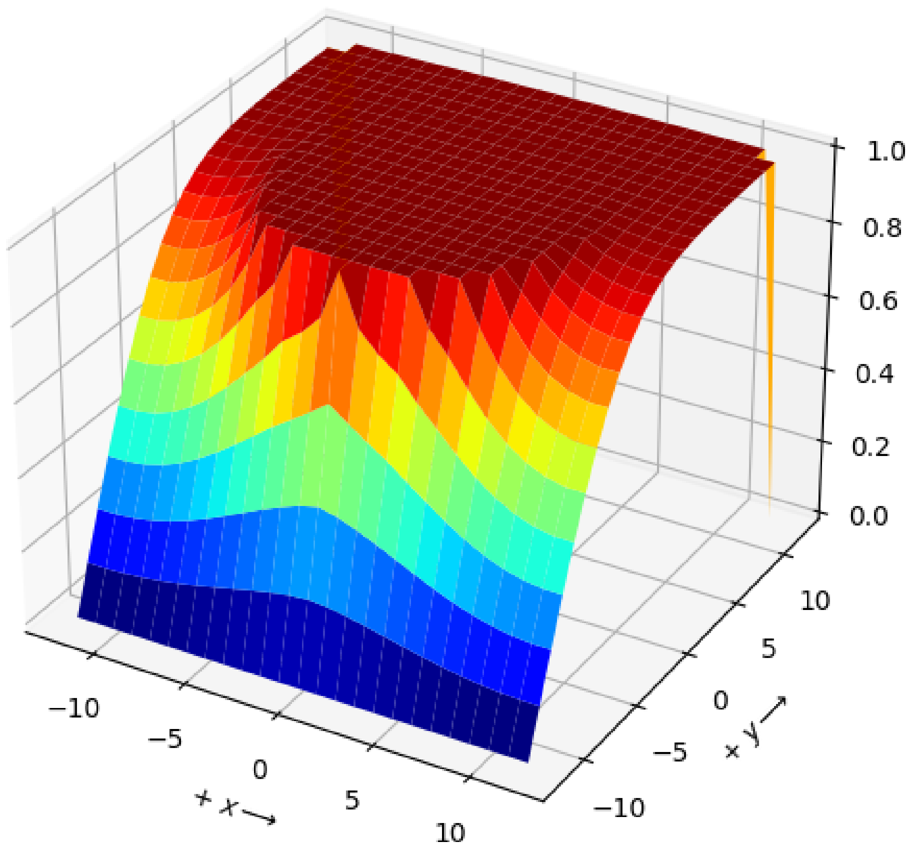

Figure 1.7: Surface Plot of Electric Potential

```python

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 Jx and Jy Jx,ij = 0.5 (i,j1 i,j+1) Jy,ij = 0.5 (i1,j i+1,j)

```python

Jx = 0.5*(phi[1:-1,:-2]-phi[1:-1,2:])
Jy = 0.5*(phi[2:,1:-1]-phi[:-2,1:-1])
```

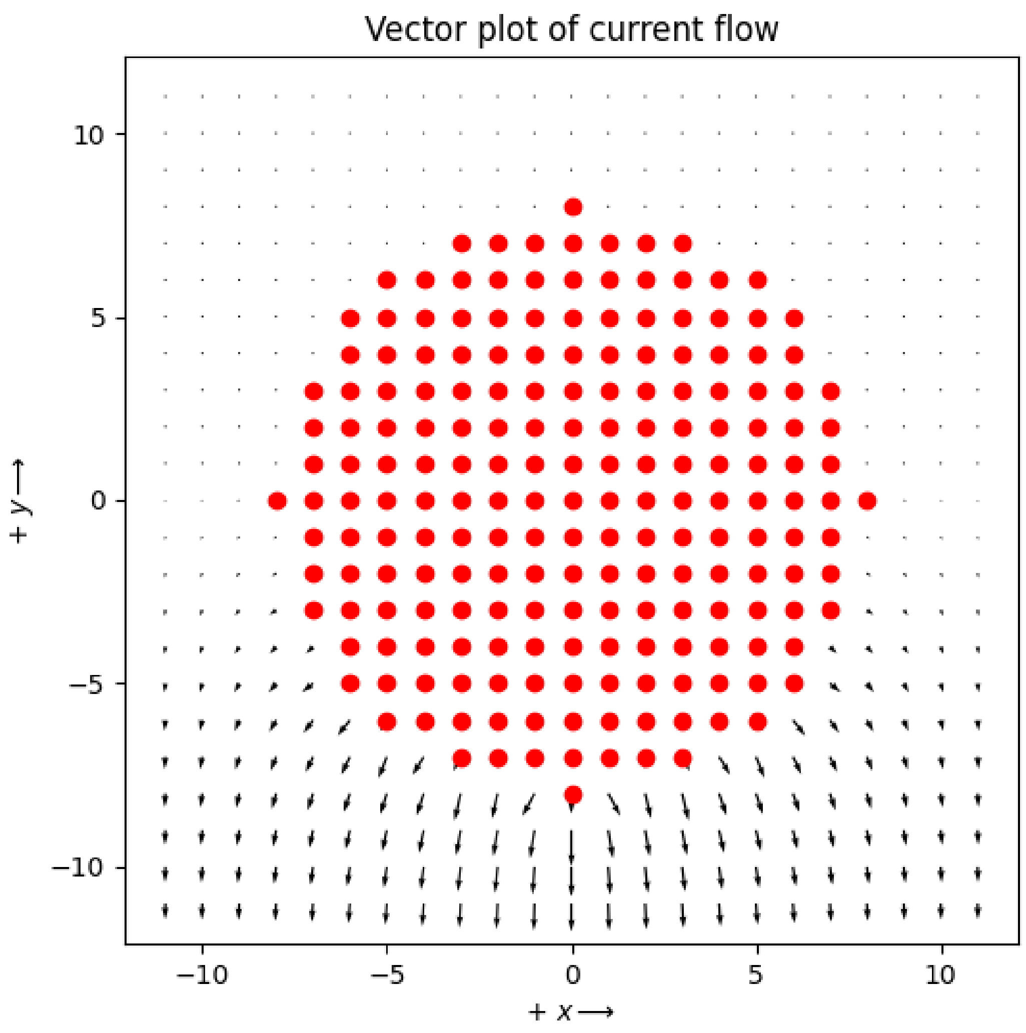

Figure 1.8: Vector Plot of Current Flow

8 Conclusion:

We have computed the current density at every point in the plate. The current density is maximum at the bottom of the plate, and hence the potential gradient is maximum there too.