EE2703: Applied Programming Lab Assignment 5 Resistor Problem using Laplace Equation

Kaushik Ravibaskar
EE20B057

March 11, 2022

Contents

1 Aim

The assignment mainly deals with the following points:

- To solve the **laplacian** equation using the technique of **averaging of neighboring points** using python **sub-arrays**.
- To analyse the **error** in each iteration and also find the **cumulative error** corresponding to net iteration number.
- To plot **2-D and 3-D contours** for potential pattern and also plotting the flow of current in a vectorized manner, along with plotting and analysing the error data.

2 Introduction

In this assignment, we will talk about the following two aspects and perform corresponding python operations.

2.1 The Resistor Problem

A wire is soldered to the middle of a copper plate and its voltage is held at $1V$. One side of the plate is grounded, while the remaining are open to space. The plate is 1cm by 1cm in size. As a result, current flows. The current at each point can be described by a current density $\vec{j}$. This current density is related to the local Electric field by the conductivity:

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

Now the Electric field is the gradient of the potential,

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

and continuity of charge yields,

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

Combining these equations, we obtain,

$$\nabla \cdot (-\sigma \nabla \phi) = -\frac{\partial \rho}{\partial t}$$

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

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

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

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

2.2 Numerical Solutions in 2-D

Laplace's equation is easily transformed into a difference equation. The equation can be written out in 2-D Cartesian coordinates as,

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

Assuming ϕ is available at points (x_i, y_j), we can write,

$$\frac{\partial phi}{\partial x} = \frac{\phi(x_{i+0.5}, y_j) - \phi(x_{i-0.5}, y_j)}{\Delta x}$$

and,

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

Combining this with the corresponding equation for the y derivatives, we obtain,

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

Thus, if the solution holds, 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.

2.3 The Solution Process

Thus, the process is obvious now. We will first make the potential in a circle of radius 8 units as 1V. Then we will use the above mentioned averaging process to determine the potential distribution. We will keep on iterating till the solution converges, or the error becomes very small. For the **boundary conditions**, we have to take care of the following points:

- For electrode boundary, set it up as 1V as it must be maintained at that voltage.
- For the outer boundary, the bottom edge will be set to 0V, rest all will be made equal to the previous point potentials, as current can't flow through the edge into the air!

3 Explaining the Code and Plots

3.1 Importing the libraries

Firstly, we will import all the important libraries necessary for the program.

```
import math as ma
import sys
from turtle import title
import scipy.linalg as sp
import numpy as np
import pylab as py
import mpl_toolkits.mplot3d.axes3d as p3
```

3.2 Defining the parameters

We will define some necessary parameters like:

- The number of rows in the potential matrix
- The number of columns in the potential matrix
- The raduis of the electrode
- Number of iterations

This can be either taken from the user or if not provided, will be given default values.

```
if(len(sys.argv) == 5):
    Nx = float(sys.argv[1])
    Ny = float(sys.argv[2])
    radius = float(sys.argv[3])
    Niter = float(sys.argv[4])
    print('Using user provided parameters for the program.')
else:
    Nx = 25
    Ny = 25
    radius = 8
    Niter = 1500
    print('Using default parameters for the program.')
```

3.3 Creating the Potential Array ϕ

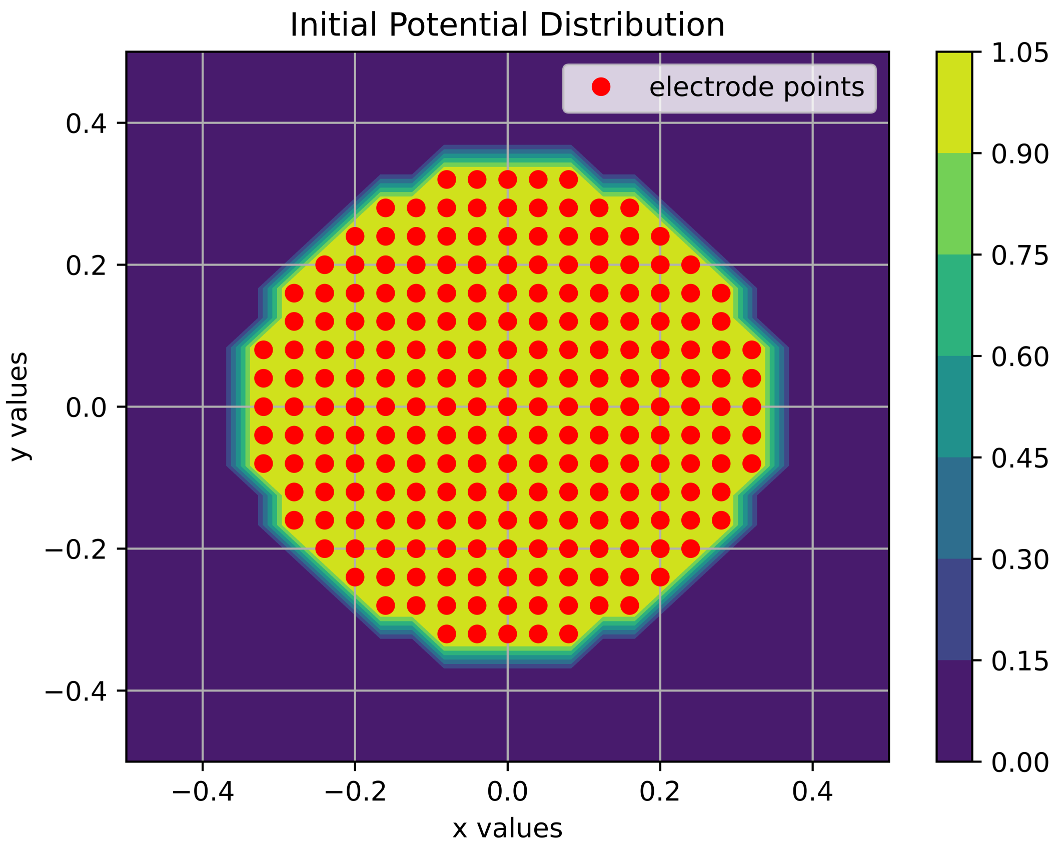
We will now create the potential array ϕ (of dimension $N_x * N_y$ which will hold the potential distribution in the plate. Firstly, we will initialize all the points in it to 0V. Then we will make the circular portion of radius 8 at its center as 1V. They symbolize the electrode nodes attached to the plate.

```
phi = py.zeros((Ny, Nx))
x = py.linspace(-0.5, 0.5, Nx)
y = py.linspace(-0.5, 0.5, Ny)
```

```
Y, X = py.meshgrid(y, x)
ii = py.where(X*X + Y*Y <= (0.35)*(0.35))
phi[ii] = 1.0
```

3.4 Plotting the initial contour of ϕ

The contour plot of the initial potential distribution in the plate looks as follows:



3.5 Performing the Laplace Operation

The Laplace operation can be laid out as follows:

- Firstly, we will create a copy of the old ϕ matrix values and store it in another matrix. This will be used to compute the error in each iteration.

- Then we will perform the Laplace algebra by making potential of the inner points excluding the boundaries as the average of the four neighboring points.
- Then we will take care of boundary conditions, one for electrode boundary and other for edge boundaries, the explanation as mentioned in the introduction.
- Finally we will compute the error in the iteration by finding the maximum difference between the old and the new ϕ matrix.

The code to perform the operation is as follows:

```
error = py.zeros(Niter)

for index 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, Nx-1] = phi[1:-1, Nx-2]
    phi[0, 0:] = phi[1, 0:]
    phi[ii] = 1.0
    error[index] = abs(phi-oldphi).max()
```

3.6 Analysing Real and 'Best fit' Errors

It is found that the error in the data is of exponential nature explicitly for larger iterations. It is of the form,

$$y = Ae^{bx}$$

Taking log on both sides, we have,

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

. Hence we will find a linear model using the **least square** algorithm to get a best fit for A and B as per the given error data values. We will plot them together with the true error values, both in semilog and loglog fashion. The code for the program is as follows:

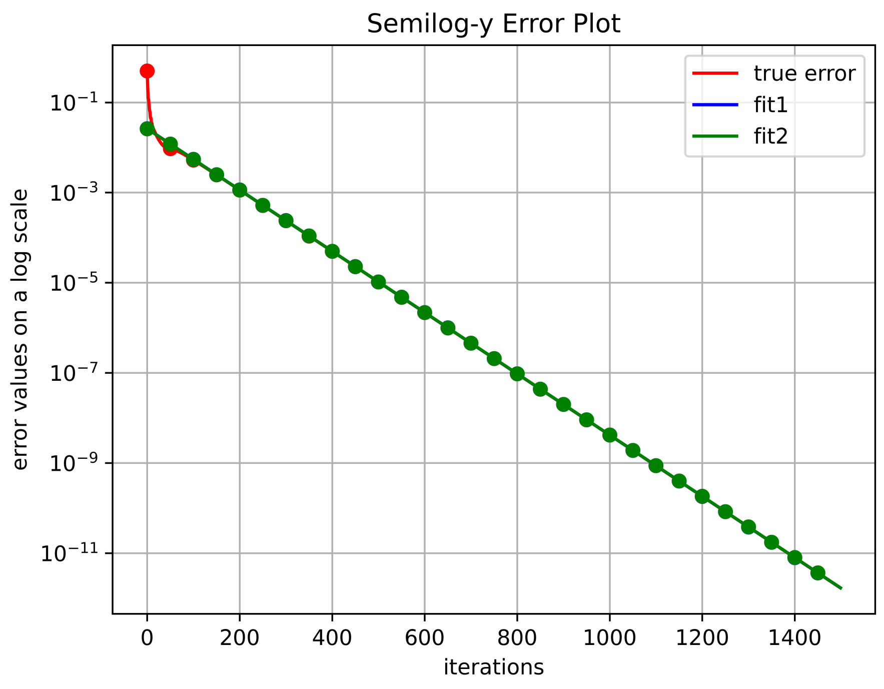
```
M_all = py.column_stack((py.ones(Niter), py.array(range(Niter))))
M_5h = py.column_stack((py.ones(Niter)[500:], py.array(range(Niter))[500:]))

fit1_data = fitfinder(M_all, py.log(error))
fit2_data = fitfinder(M_5h, py.log(error[500:]))
```

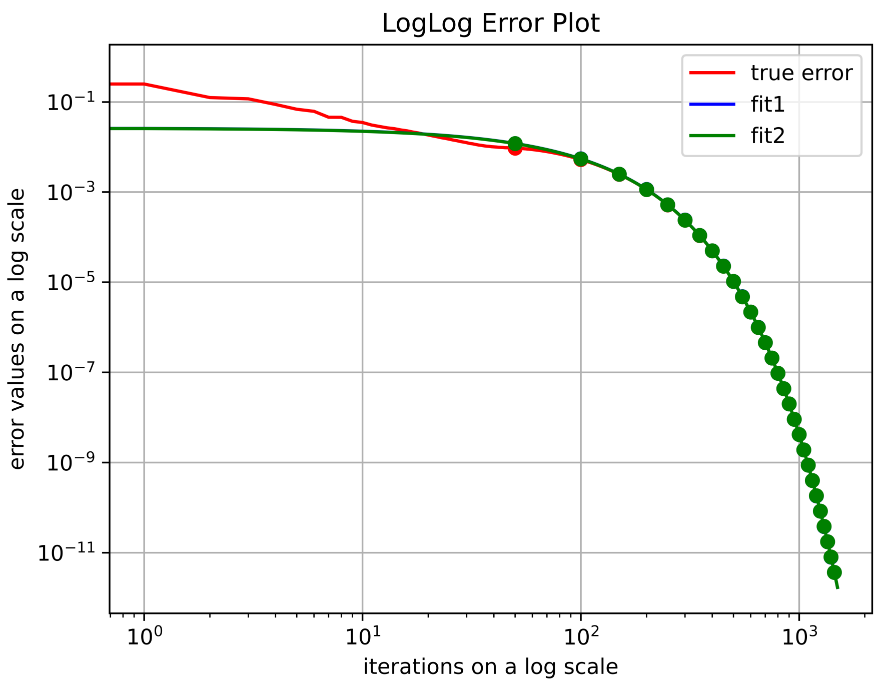
```
fit1_final = py.dot(M_all, fit1_data)
fit2_final = py.dot(M_all, fit2_data)
```

where fitfinder is a function defined as follows to compute the least square parameters,

```
def fitfinder(a, b):
    return sp.lstsq(a, b)[0]
```

The plot looks as follows:





3.7 Analysing Cumulative Error

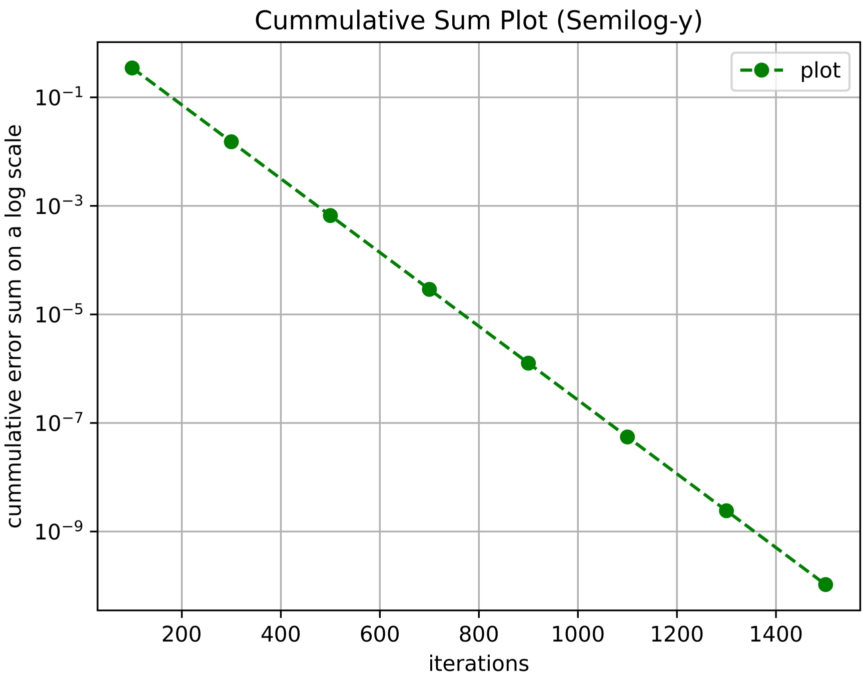
Now the maximum error scales as,

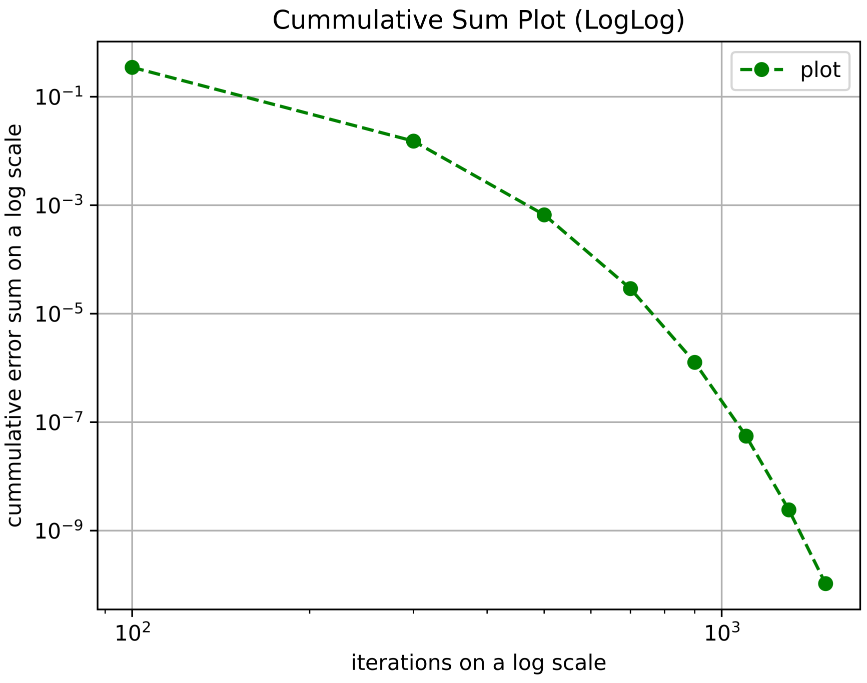
$$error = Ae^{Bk}$$

where k is the iteration number. Now if we sum this error from $k = N + 1$ to $k = \infty$, we get the following expression for the cumulative error,

$$Error = (\frac{-A}{B})\exp(B(N + 0.5))$$

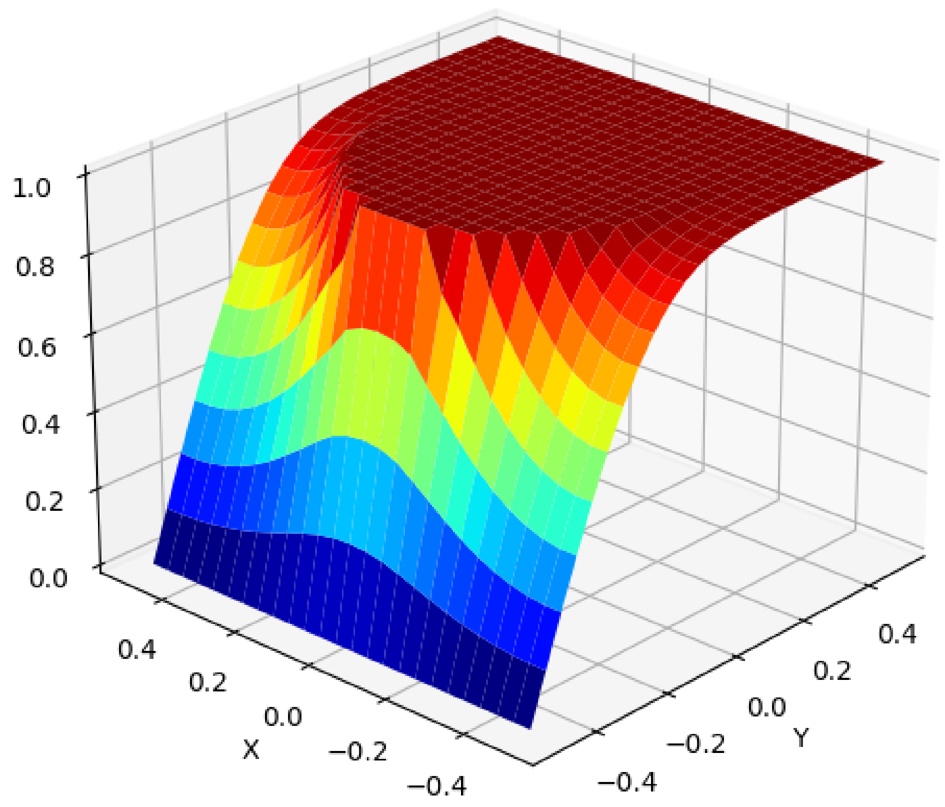
. It basically means that, if we start from smaller N, we will get a larger cumulative error as the individual error, however small, adds up. The cumulative sum plot in both scales looks as follows:


Cummulative Sum Plot (Semilog-y)
plot
cummulative error sum on a log scale
iterations
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
200
400
600
800
1000
1200
1400

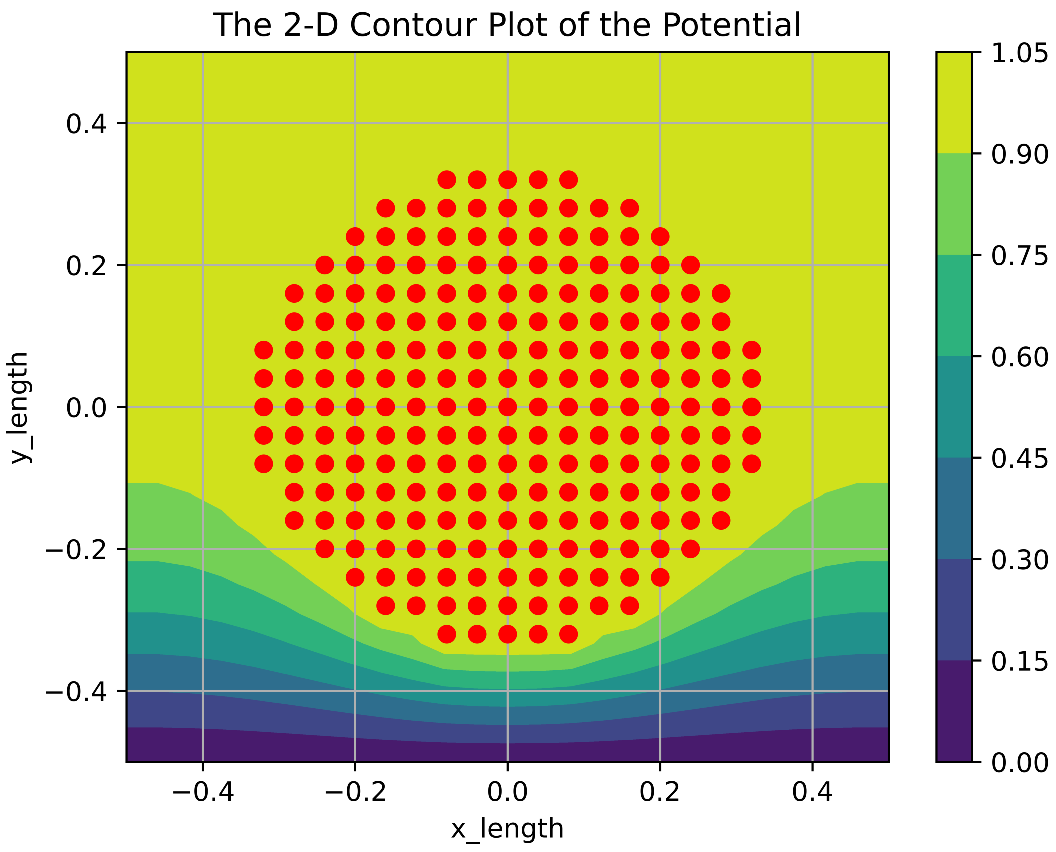

Cummulative Sum Plot (LogLog)
plot
cummulative error sum on a log scale
iterations on a log scale
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
10^2
10^3

3.8 Plotting 3-D and 2-D Potential Pattern

The 3-D surface plot of the potential pattern on the plate is as follows:



The 2-D contour plot for the same is as follows:



3.9 Vector Plot of Currents

We know that current densities in x and y directions can be written as,

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

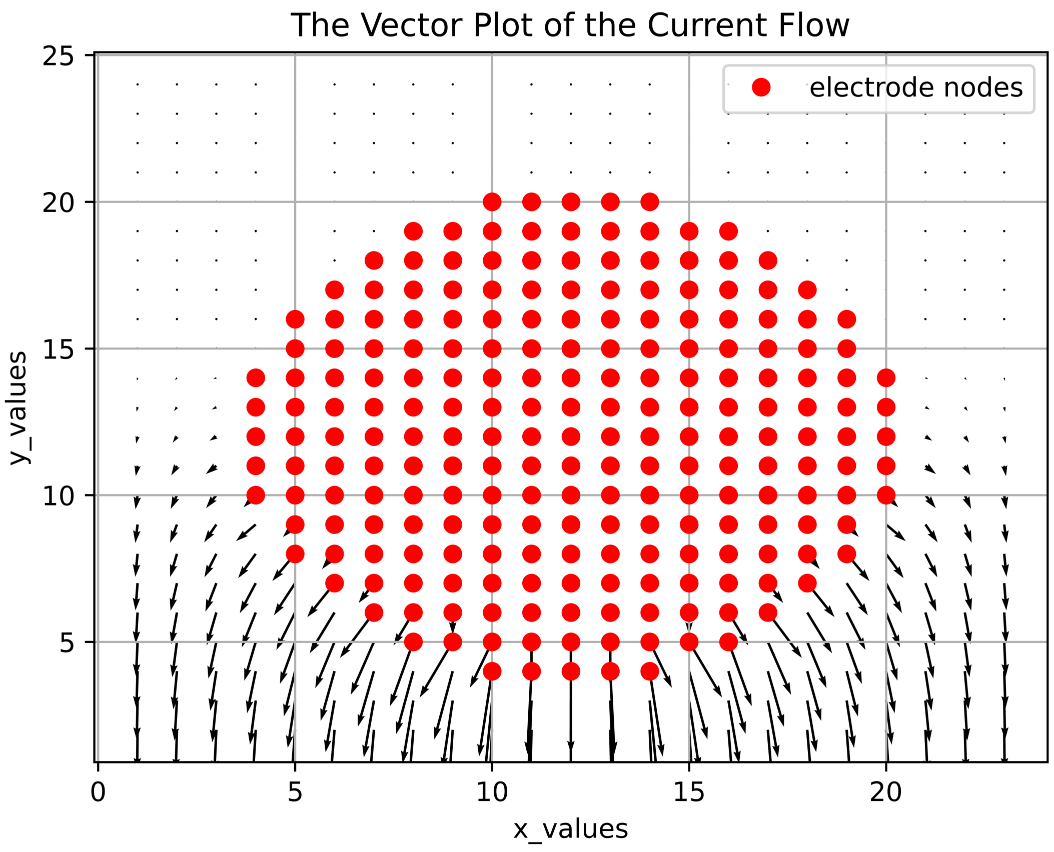
$$j_y = -\frac{\partial \phi}{\partial y}$$

which numerically translates to,

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

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

Hence we will create two arrays J_x and J_y and use **quiver** function to make vector plots for current. The plot looks as follows:



4 Conclusion

We can make the following conclusions:

- The exponential error function is valid only for iterations greater than 500 as can be seen from the best fit plots.
- This method of solving Laplace equation is considered the worst method because of the slow coefficient with which the error reduces.
- We can conclude that as the density of current vectors is very high at the bottom of the plate, the temperature will be high there as

$$P_{heat} \propto j^2$$