# Assignment 5: The Resistor Problem

Saurav Sachin Kale, EE19B141

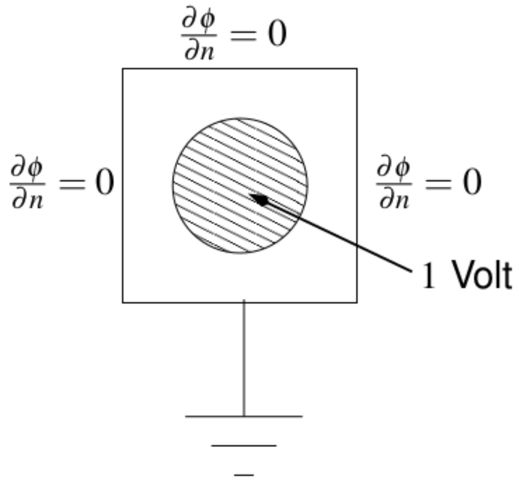
March 26, 2021

## 1 Aim

- To solve and visualize the potential distribution, current density field, and temperature profile of a thin 1cm x 1cm area copper plate with an electrode of given radius kept at 1V attached to the plate at its center. The bottom edge of this plate is grounded.
- Learn and implement one way of solving the Laplace Equation
- Assess the feasibility and efficiency of this method of solving Laplace Equation

## 2 The Resistor Problem and its solution

The diagram below illustrates a thin 1cm x 1cm area copper plate with an electrode of given radius kept at 1V attached to the plate at its center.


$\frac{\partial\phi}{\partial n} = 0$
$\frac{\partial\phi}{\partial n} = 0$
$\frac{\partial\phi}{\partial n} = 0$
1 Volt

The following equations must hold from our knowledge of electromagnetism:

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

(Conductivity)

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

(Electric field is gradient of potential)

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

(Continuity Equation)

Combining these three, we get the following equation:

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

Assuming constant conductivity we obtain:

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

And for DC currents $\frac{\partial\rho}{\partial t} = 0$, therefore:

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

For our case, we are dealing with 2D distribution, so we can write $\nabla^2\phi$ as:

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

## 3 Numerical Solution to Laplace equation

We now have this equation to solve in order to calculate our potential distribution:

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0 \tag{1}$$

We split the copper plate into discrete sections so the value of $\phi$ is available at points $(x_i, y_j)$ we can write:

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

$$\frac{\partial^2\phi}{\partial y^2} = \frac{\phi(x_i, y_{j+1}) + \phi(x_i, y_{j-1}) - 2\phi(x_i, y_j)}{(\Delta y)^2}$$

Replacing these in the equation (1) we get:

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4} \tag{2}$$

Iterating equation (2) repeatedly over an initial distribution should give us the steady state values of $\phi$.

The boundary conditions are these:

- $\phi = 1V$ at the electrode junction
- $\phi = 0V$ at the bottom edge, where the plate has been grounded

We shall ensure these are satisfied in every iteration.
We have thus found one way to solve the Laplace Equation.

## 4 Procedure

### 4.1 Defining the parameters

- $N_x, N_y$: This represents the number of horizontal indices. Each discrete step is assumed to be of length 1 unit. The conversion between these units and the real life measurements is left to the user. A user may set a value of $(25, 25)$ in order to divide a 1cm x 1cm plate into 25 x 25 discrete blocks. If the user wishes more fine divisions, $(100, 100)$ or larger may be used. Setting $N_X \neq N_y$ creates a rectangular plate. Therefore by tweaking these parameters, a user can adjust any dimension rectangular plate to the desired level of detail. By default this is taken as $(25, 25)$.

- $R$: Radius of the electrode in units. By default this is taken as 8 units.
- $N_{iter}$: The number of iterations for calculating the solution to the laplace equation. By default, $N_{iter} = 1500$.

These parameters take their default values unless specified while calling the program as command line arguments. For the below analysis, default values have been used.

## 4.2 Calculation of $\phi$ and the error

Initially, $\phi$ is initialized to being 1V where the electrode is in contact, that is in a circle of radius $R$ centered at the center of the plate. The red dots represent the electrode junction

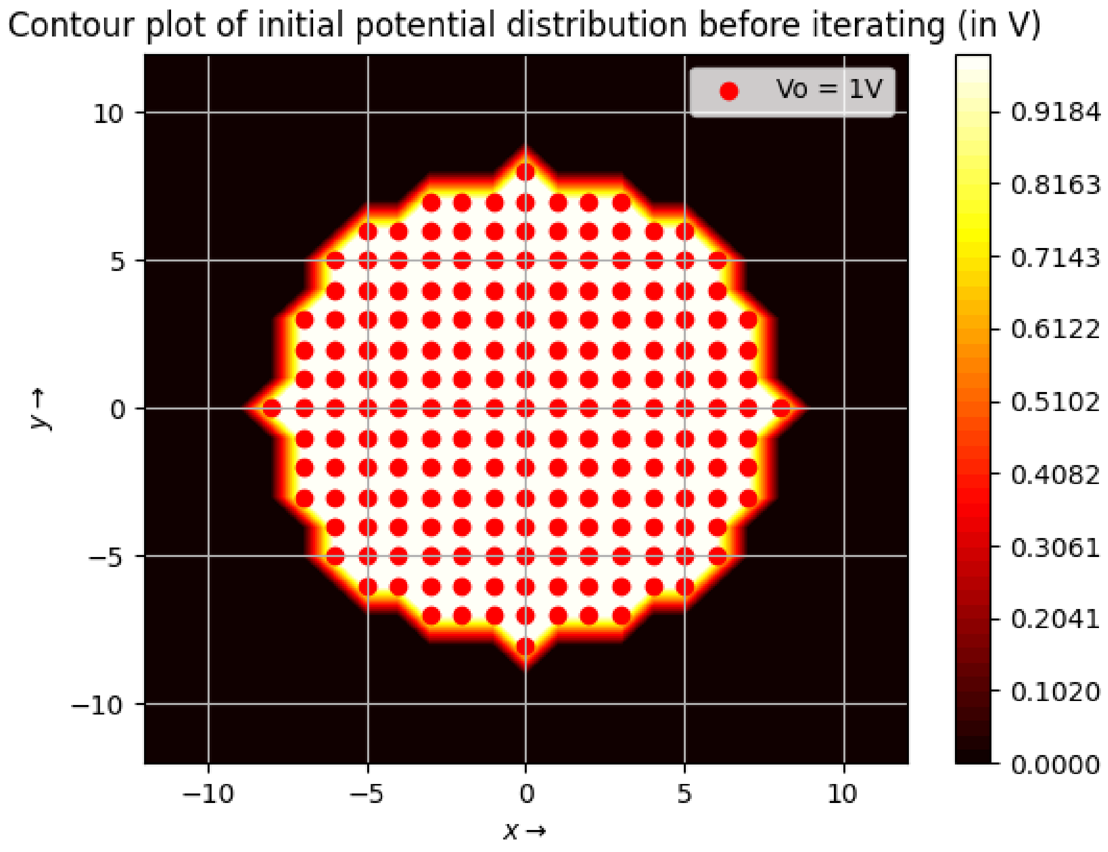


Figure 1: Contour plot of the initial value of $\phi$

We then obtain the steady state solution, and the error per iteration using the following algorithm:

**Algorithm 1** Solve-Laplace

```
    do Niter times:

        #save phi
        oldphi = phi.copy()
        #update phi
        for each phi[i, j] in phi:

            phi[i, j] = (oldphi[i + 1, j] + oldphi[i - 1, j] + oldphi[i, j + 1] + oldphi[i, j - 1])/4
        #boundary conditions
        phi[:, 0] = phi[:, 1]
        phi[:, Nx - 1] = phi[:, Nx - 2]
        phi[0, :] = phi[1, :]
        phi[Ny - 1, :] = 0
        phi[electrode_area] = 1
        errors.append(abs(phi-oldphi).max())
```

The boundary conditions are implemented by ensuring that at every iteration, the electrode contact area is at 1V, and the bottom row is at 0V.
We now make use of vectorized python notation in order to make our code more efficient. We can do the update phi part of the code with a single line instead of a loop as follows:

```
phi[1:-1, 1:-1] = 0.25*(oldphi[2:, 1:-1] + oldphi[:-2, 1:-1] + oldphi
    [1:-1, 2:] + oldphi[1:-1, :-2])
```

And our boundary conditions are as follows:

```
#left edge
phi[1:-1,0]=phi[1:-1,1]
#right edge
phi[1:-1,-1]=phi[1:-1,-2]
#bottom edge
phi[0, 1:-1]=0
#top edge
phi[-1, 1:-1]=phi[-2, 1:-1]
# 4 corners get updated with the average of their neighboring values
phi[0, 0] = 0.5*(phi[0, 1] + phi[1, 0])
phi[0, -1] = 0.5*(phi[0, -2] + phi[1, -1])
phi[-1, 0] = 0.5*(phi[-1, 1] + phi[-2, 0])
phi[-1, -1] = 0.5*(phi[-1, -2] + phi[-2, -1])
#restore the electrode portion
phi[ii]=1.0
#ii is the electrode area
```

After implementing this, we obtain the following plots:

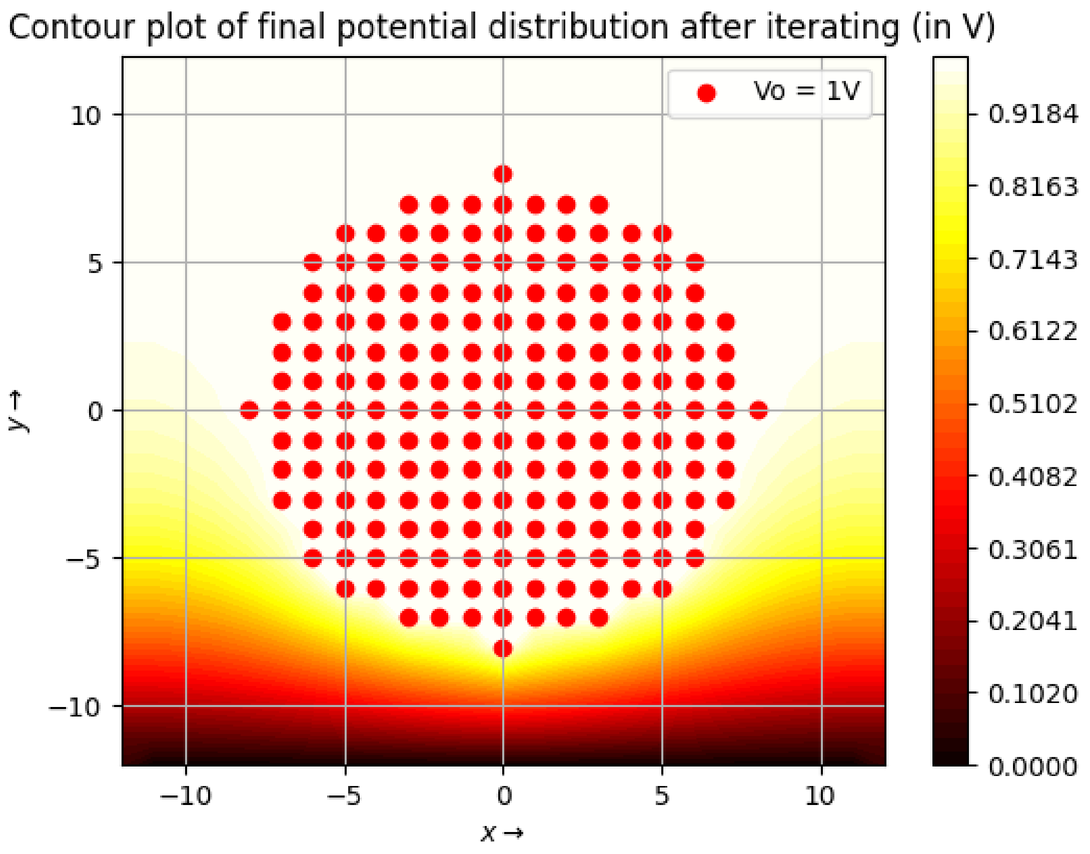


Figure 2: Contour plot of the steady state value of $\phi$ (in V)calculated using Laplace equation

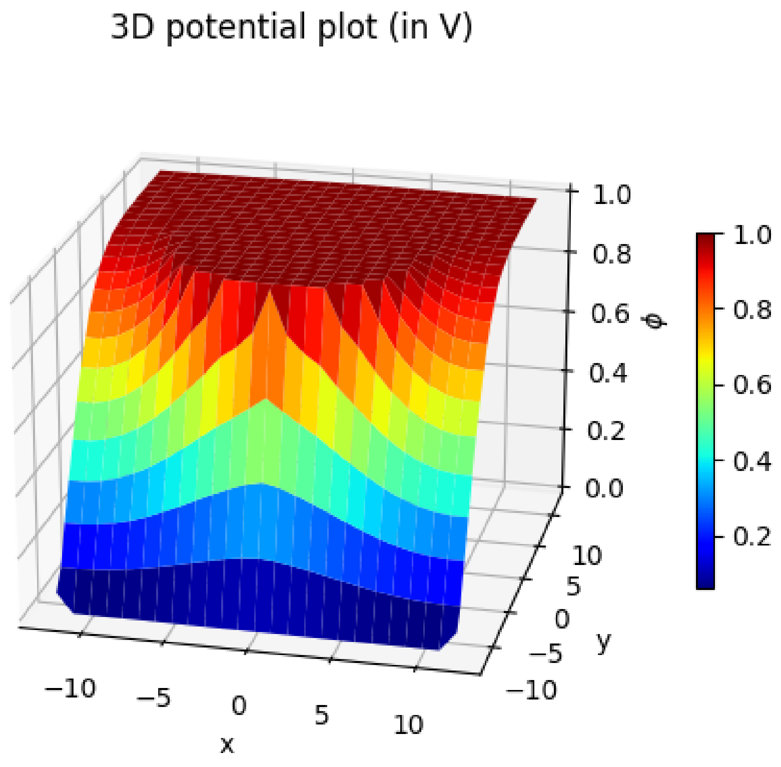


Figure 3: 3D surface plot of the steady state value of $\phi$ (in V) calculated using Laplace equation

Now we have also calculated errors as a function of number of iterations. We now attempt to extract a fit for the semilogy plot of error vs. number of iterations like so:

$$y = Ae^{Bx}$$

$$logy = logA + Bx$$

From our previous experience in curve fitting (Assignment 3), we will use the method of least squares to compute the values of $logA$ and $B$. Our data is every $50^{th}$ datapoint of this graph. We represent this problem in matrix form like so:

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ ... & ... \\ 1 & x_n \end{pmatrix} \begin{pmatrix} logA \\ B \end{pmatrix} = \begin{pmatrix} logy_1 \\ logy_2 \\ logy_3 \\ ... \\ logy_n \end{pmatrix}$$

$$M \cdot p = G$$

And we obtain $p$ using SciPy's lstsq() function:

```
M = np.c_[np.ones(len(x_datapoints)), x_datapoints]
p = scipy.linalg.lstsq(M, np.log(Y))
```

For all the datapoints over all iterations, we get the fit:

$$p = \begin{pmatrix} -3.74206593 \\ -0.0141953 \end{pmatrix}$$

For the datapoints beyond 500 iterations, we get the fit:

$$p = \begin{pmatrix} -3.73856391 \\ -0.01419784 \end{pmatrix}$$

We now plot both of these fits along with the semilog plot:

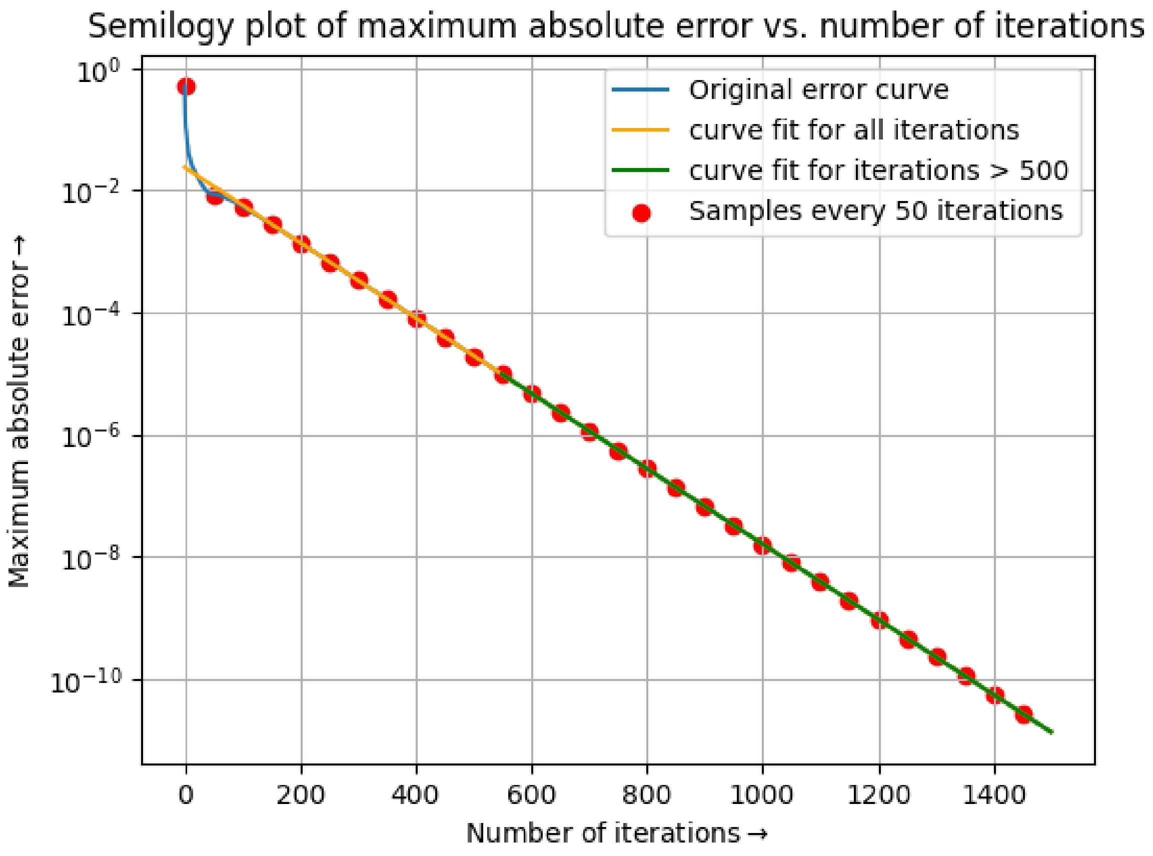


Figure 4: Semilogy plot of errors vs. number of iterations, with fitted curves.

Plotting a loglog plot reveals that the curve is fairly linear till about 500 iterations, but then it becomes exponential.

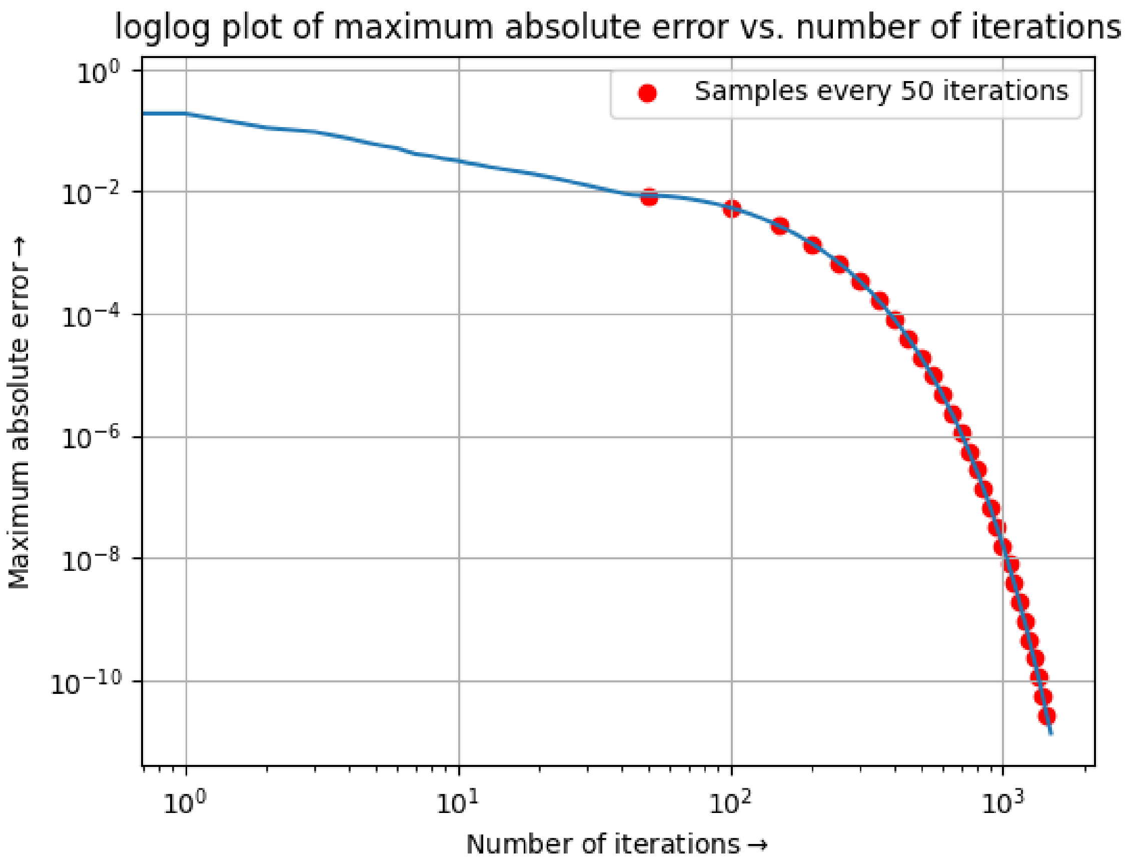


Figure 5: Loglog plot of errors vs. number of iterations

### 4.3 Assessing the performance of the Solve-Laplace Algorithm

We now analyse the feasibility and performance of this algorithm by plotting the maximum possible error obtained in the final plot as a function of $N_{iter}$.
We notice that the error varies as:

$$y = Ae^{Bx}$$

$$Error = \Sigma_{k=N+1}^{\infty} error_k$$

An upper bound for this estimate is:

$$Error < \Sigma_{k=N+1}^{\infty} Ae^{Bk}$$

since we are taking the absolute value sum.
We can approximate this as continuum and therefore the summation tends to an integral:

$$Error < \int_{N+0.5}^{\infty} Ae^{Bk} dk$$

$$Error < -\frac{A}{B}e^{B(N+0.5)} < \frac{A}{B}e^{B(N+0.5)}$$

We take the following fit values:

$$p = \begin{pmatrix} -3.74206593 \\ -0.0141953 \end{pmatrix}$$

Varying $N$ in this expression, we obtain the following plot:

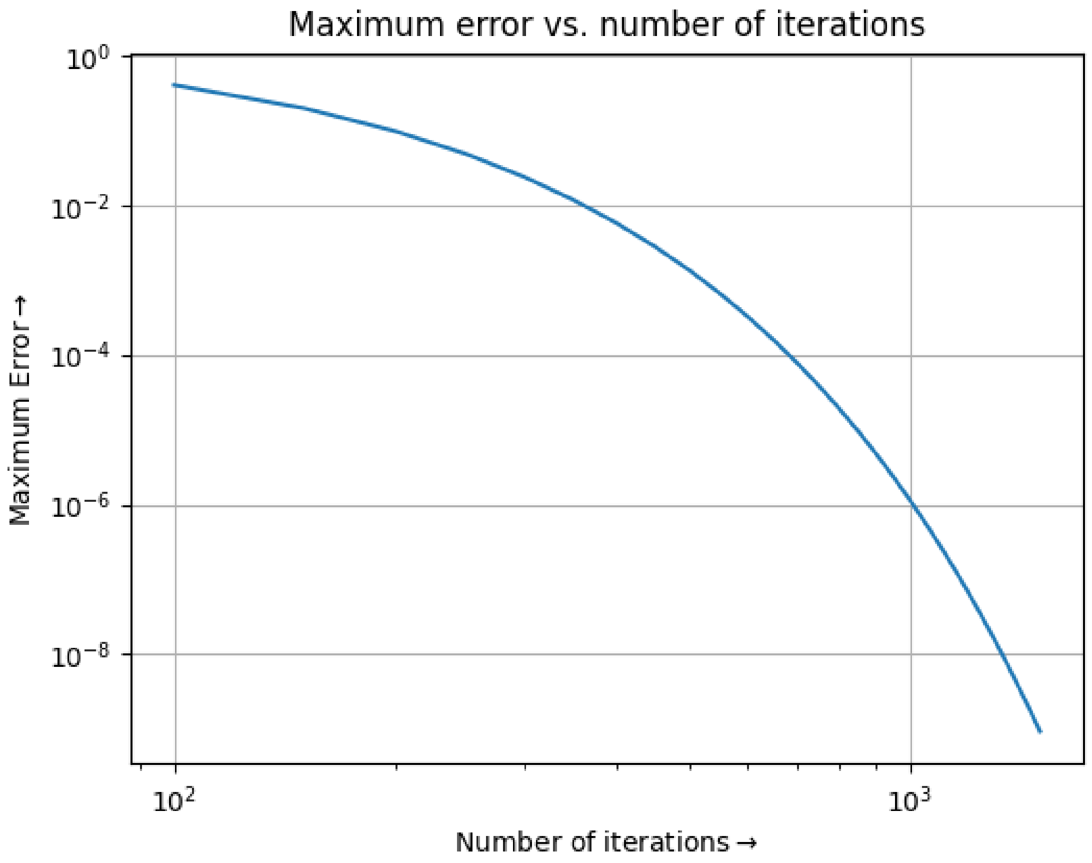


Figure 6: Maximum cumulative error vs. $N_{iter}$

This algorithm is one of the worst ways to solve a Laplace equation, because the time constant is quite high. This means the error values take a lot of iterations to converge to a value close to zero.

### 4.4 Current Density $\vec{J}$

We know:

$$\vec{J} = -\sigma \nabla \phi$$

Here we assume $\sigma = 1$, and after simplification we obtain the following:

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

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

Converting to difference equations we get:

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

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

The python code for calculating these values is:

```
Jx = 0.5*(phi[1:-1,0:-2]-phi[1:-1,2:])
Jy = 0.5*(-phi[2:, 1:-1]+phi[0:-2,1:-1])
```

We plot $\vec{J}$ using the quiver function:

```
plt.quiver(Y[1:-1, 1:-1], X[1:-1, 1:-1], Jy, Jx, scale=4, label="$\\vec{J}
    $")
```

The following is the result.

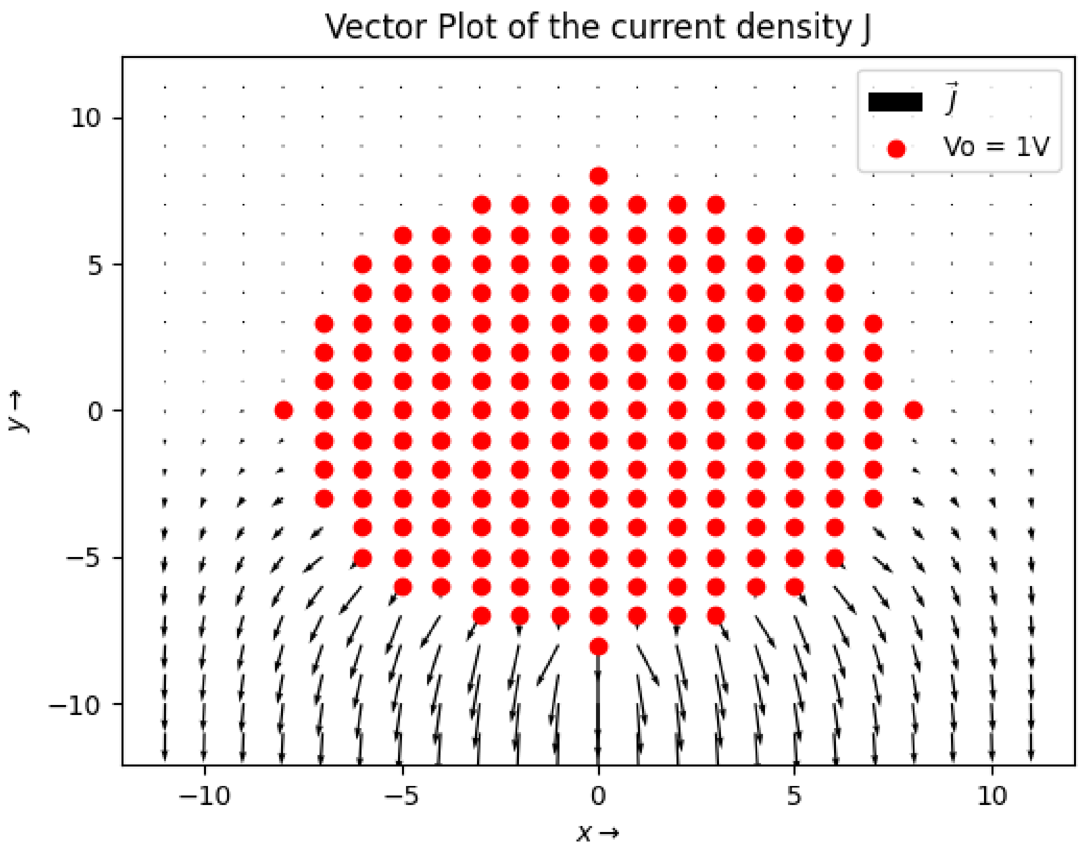


Figure 7: Plot of the vector field $\vec{J}$

Most of the current seems to flow through the bottom half of the plate. Hardly any current flows from the top part of the wire. This is because as we saw in the potential contour plot, the value of $\phi$ is almost constant and equal to 1V in the top part. This means $\vec{J} = -\sigma\nabla\phi \approx 0$. Hence hardly any current flows from the top part of the wire.

## 4.5 Temperature distribution across the plate

We now attempt to calculate and visualize the temperature distribution across the copper plate. We do this by writing the heat equation:

$$-\nabla \cdot (\kappa \nabla T) = \frac{1}{\sigma}|\vec{J}|^2$$

Assuming constant $\kappa, \sigma$ we get:

$$\nabla^2 T = -\frac{1}{\sigma\kappa}|\vec{J}|^2$$

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = -\frac{1}{\sigma\kappa}|\vec{J}|^2$$

Converting to difference equations, we get the following:

$$\frac{T_{i+1,j} + T_{i-1,j} - 2T_{i,j}}{(\Delta x)^2} + \frac{T_{i,j+1} + T_{i,j-1} - 2T_{i,j}}{(\Delta y)^2} = -\frac{1}{\sigma\kappa}|\vec{J}|^2$$

We assumed in Section 4.1 that $\Delta x = \Delta y = \Delta n = 1 unit$. Rearranging, we obtain:

$$T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1} - 4T_{i,j} = -\frac{1}{\sigma\kappa}|\vec{J}|^2$$

$$T_{i,j} = \frac{T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1}}{4} + \frac{1}{4\sigma\kappa}|\vec{J}|^2 \tag{3}$$

Since $\kappa, \sigma$ are scaling constants, we ignore their effect for now by setting $\kappa\sigma = 1$.
The general profile of the temperature plot will not be affected by this. We initialize the entire plate at room temperature (300K).
Our boundary conditions are

- Electrode contact area remains at 300K
- Bottom edge (grounded edge) remains at 300K

Applying a slightly modified version of Solve-Laplace:

**Algorithm 2** Solve-Temp

```
do Niter times:

    #save temp
    oldtemp = temp.copy()
    #update temp
    for each temp[i, j] in temp:

        temp[i, j] = (oldtemp[i + 1, j] + oldtemp[i - 1, j] + oldtemp[i, j + 1] + oldtemp[i, j - 1]
        + Jx*Jx + Jy*Jy)/4
    #boundary conditions
    temp[:, 0] = temp[:, 1]
    temp[:, Nx - 1] = temp[:, Nx - 2]
    temp[0, :] = temp[1, :]
    temp[Ny - 1, :] = 300
    temp[electrode_area] = 300
```

We now obtain the following plot:

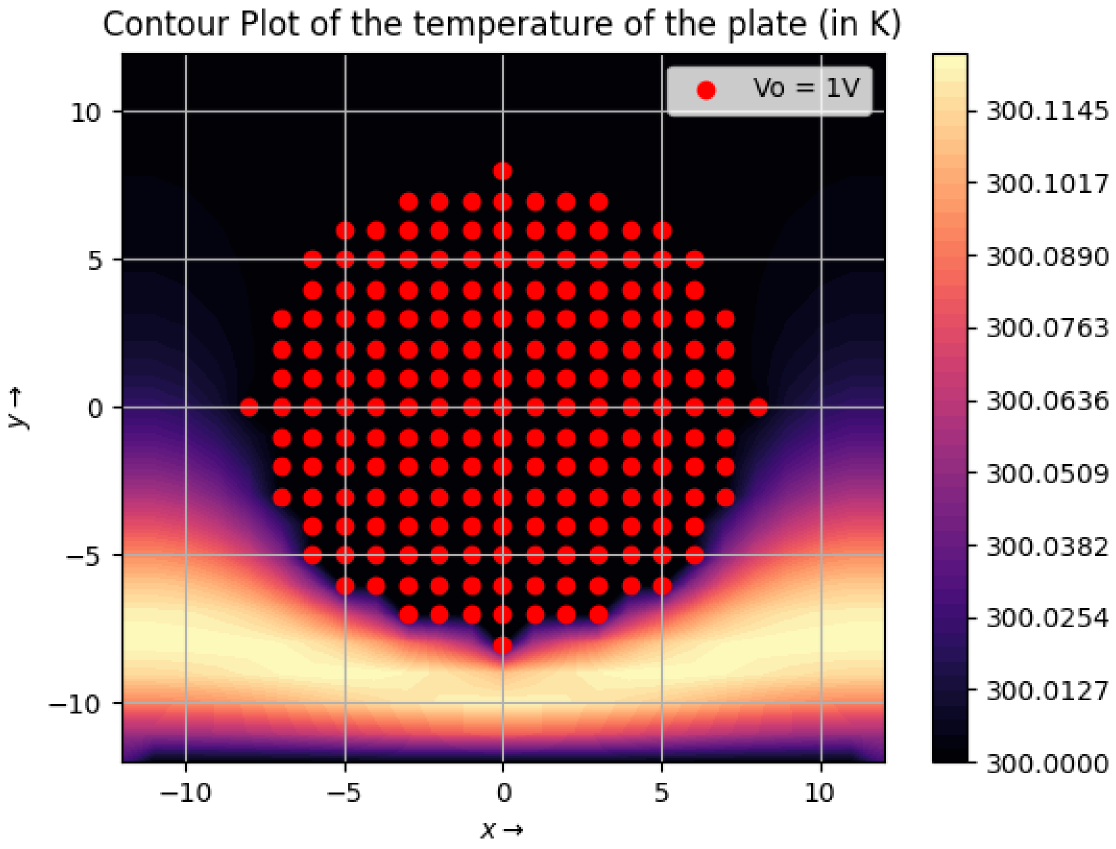


Figure 8: Contour plot of the temperature distribution across the copper plate (in K)

As expected, the areas with high current density are hotter.

## 5 Conclusions

- A copper square plate with cylinderical electrode was modelled and its potential distribution, current density and temperature distribution was calculated and visualized.
- Most of the current propagates in the lower part of the plate, since the potential gradient is near zero at the upper part of the plate.
- For the same reason, the temperatures are higher in the lower part of the plate.
- The Solve-Laplace algorithm was analysed, and is one of the most inefficient ways of solving the Laplace equation.