Assignment No. 5: Report

-Akash Reddy A, EE17B001

March 7, 2019

1 Overview

Assignment 5 deals with a Resistor Problem.

Our resistor is a square plate of area 1 cm $\times$ 1 cm, and to the centre of this resistor is attached a cylindrical electrode of radius 0.35 cm. One side of the square is grounded while the rest are left open. Our aim is to analyse the potential and current situation within the resistor in the following ways:

- Create a matrix for electric potential, and use the Laplace Equation to solve for the situation and obtain the values for the potential at all the points with some minor error.
- The Laplace Equation uses iterations during numerical implementation, and the magnitude of error changes from one iteration to the next. The evolution of these errors with iterations is captured and plotted.
- A 3D surface plot of the potential is made.
- A contour plot of the potential is made.
- A vector plot of the currents in the resistor plate is obtained.

2 The Assignment

2.1 Relevant Theory

The solution to the situation is obtained using the **Laplace Equation**, which is obtained from the Continuity Equation:

$$\nabla \cdot \vec{J} = -\frac{\partial \rho}{\partial t}$$
$$\nabla \cdot (-\sigma \nabla \phi) = -\frac{\partial \rho}{\partial t}$$
$$\nabla^2 \phi = \frac{1}{\sigma}\frac{\partial \rho}{\partial t}$$

At the steady state for DC currents, we have $\frac{\partial \rho}{\partial t} = 0$, as there is no time-variance of charge density at any point in the resistor. Therefore,

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

This is the Laplace Equation.

Writing out the Laplace Equation in Cartesian coordinates for this case, we have:

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

We express the differentials in the following numerical form for computation:

$$\left.\frac{\partial \phi}{\partial x}\right|_{(x_i,y_j)} = \frac{\phi(x_{i+\frac{1}{2}}, y_j) - \phi(x_{i-\frac{1}{2}}, y_j)}{\Delta x}$$
$$\left.\frac{\partial \phi}{\partial y}\right|_{(x_i,y_j)} = \frac{\phi(x_i, y_{j+\frac{1}{2}}) - \phi(x_i, y_{j-\frac{1}{2}})}{\Delta y}$$

We also have:

$$\left.\frac{\partial^2 \phi}{\partial x^2}\right|_{(x_i,y_j)} = \frac{\phi(x_{i+1}, y_j) - 2\phi(x_i, y_j) + \phi(x_{i-1}, y_j)}{(\Delta x)^2}$$
$$\left.\frac{\partial^2 \phi}{\partial y^2}\right|_{(x_i,y_j)} = \frac{\phi(x_i, y_{j+1}) - 2\phi(x_i, y_j) + \phi(x_i, y_{j-1})}{(\Delta y)^2}$$

Using the four above equations, we ultimately obtain:

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

i.e, **the potential at a point is equal to the average of the potentials at the four points around it.**

As a result, we set all the values of known potentials (at the electrode and at the earthed boundary) to be fixed at their values, apply boundary condition $\frac{\partial \phi}{\partial x}_{normal} = 0$ at the open boundaries (as the current *has* to be tangential at the boundaries) and apply (1) at all points, starting with a ϕ matrix of all zeroes. This is repeatedly done until the potentials start converging. Once a narrow threshold of error is reached, the iterations are stopped.

The solution for the potentials ϕ at all points has been obtained.

Once we have this solution, it becomes straightforward to plot relevant contour plots, and also to evaluate the current vectors at each point and make a vector plot.

2.2 Tackling the Assignment Step-by-Step

The program begins by importing the libraries we need through the following code:

```
from pylab import *
import mpl_toolkits.mplot3d.axes3d as p3
import sys
import numpy as np
import scipy.linalg as la
```

We use `pylab` to draw 2-D plots, `mpl_toolkits.mplot3d.axes3d` to plot 3-D surface plots, `sys` to accept commandline arguments, `numpy` to deal with datatypes such as arrays and perform operations on arrays, and `scipy.linalg` for using linear algebra operations (`lstsq` in this case).

Accepting User Inputs

Next, it moves on to accepting user inputs as arguments from the commandline using `sys.argv` as follows:

```
Nx=int(sys.argv[1])
Ny=int(sys.argv[2])
radius=float(sys.argv[3])
Niter=int(sys.argv[4])
```

The arguments that we accept are:

- `Nx`, the number of columns of data points we divide our resistor plate into.
- `Ny`, the number of rows of data points we divide our resistor plate into.
- `radius`, the radius of the cylindrical electrode we are attaching to the centre of the resistor, in cm.
- `Niter`, the number of iterations we perform on our potential array in order to update the values.

Initialising the Potential Matrix and Placing the Electrode

Now, we create the potential array with Nx columns and Ny rows, and initialise it to zero with

```
phi=zeros((Ny,Nx))
```

after which we create the x-axis and y-axis values arrays. We take care to make sure that x=0 and y=0 are at the centre of the respective arrays, so that we can place the electrode and make V = 1 at all points that are within the radius from the centre of the plate. Subsequently, a meshgrid of these two arrays is created.

```
x=np.arange(float(-(Nx-1)/2),float((Nx+1)/2),1)
y=np.arange(float(-(Ny-1)/2),float((Ny+1)/2),1)
X,Y=meshgrid(x,y)
```

We then make the potential at all the points in direct contact with the electrode equal to 1 using:

```
ii=where((X/Nx)**2+(Y/Ny)**2<=(radius*radius))
phi[ii]=1.0
```

Here, we divide `X` and `Y` by `Nx` and `Ny` respectively so that we can compare it with `radius`. `X` and `Y` are presently in the units of number of data points, and not cm. In order to compare it with `radius`, we first convert the values to cm, and then make $V = 1$ at the points which satisfy $X^2 + Y^2 \leq radius^2$.

We then make a contour plot of these initial potentials (the potentials we have before applying our solution):

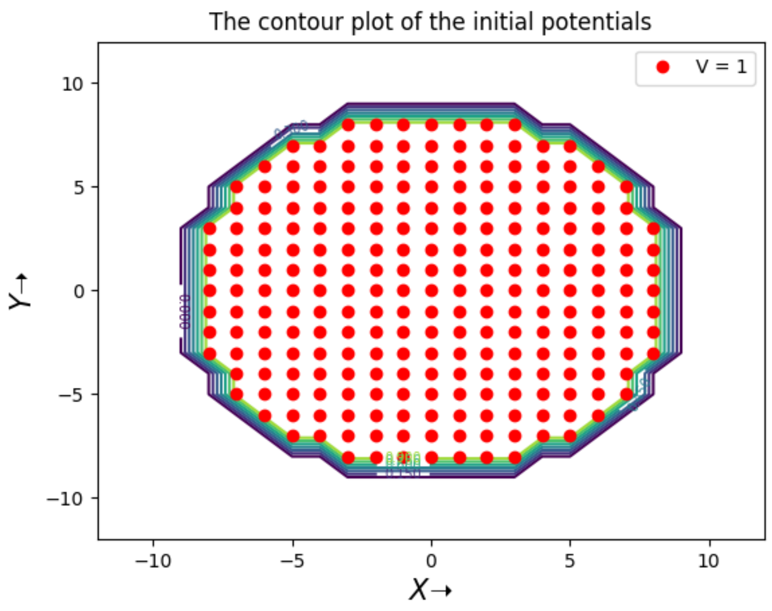

Figure 1: Contour plot of the initial potentials

In the above plot, the points where the potential is equal to exactly 1 (the points in contact with the electrode) are marked with red circles.

Updating the Potential through Iterations and Recording and Plotting the Evolving Errors

```
errors=zeros((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()
```

`errors` is the array we use to store the value of error at each iteration, hence its length is equal to `Niter`. We initialise the values of all errors to 0 first and then update them one by one at each iteration.

`oldphi` contains a copy of `phi` from which we calculate the error for each iteration.

We now update the values of potential at each point to be the average of the potentials at its four closest neighbours. **This is the implementation of the Laplace Equation that we arrived at earlier.**

The boundary condition $\frac{\partial \phi}{\partial x}_{normal} = 0$ is implemented by making sure that the values of potential at the points of the *penultimate* row/column at the top, left and right sides, are equal to the corresponding *ultimate* row/column.

It is of importance to note that Python's arrays help us work in a much smoother manner with such matrices, over the same implementation in C which would require for loops and updating the array values one by one.

Plotting the Evolution of the Errors

The errors are plotted against the number of iterations on a **semilog** plot. We observe that this plot is linear with decreasing slope. We highlight every 50^{th} point on this curve for visibility of individual data points.

Along with the actual errors, we also plot the best fit curves for the errors, by assuming the function for the errors to be of the form:

$$y = Ae^{Bx}$$

Taking log on both sides, we get:

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

which is a linear plot on the *semilog*(y) scale.

Therefore, we assume the best-fit curve to be a linear curve on the *semilog*(y) scale and calculate the values of A and B that best fit the curve. The equation that we obtained above:

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

gives:

$$\frac{\log y}{B} - \frac{\log A}{B} = x$$

When this is applied to all data points that we have, we get a series of equations which can be reduced to the following matrix equation:

$$\begin{bmatrix} \log y_1 & -1 \\ \log y_2 & -1 \\ \vdots & \vdots \\ \log y_{Niter} & -1 \end{bmatrix} \cdot \begin{pmatrix} \frac{1}{B} \\ \frac{\log A}{B} \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{Niter} \end{pmatrix}$$

or

$$M \cdot x = b$$

We implement this in the following code:

```
C1=np.log(errors)
C2=-1*ones((len(errors)))
M=c_[C1,C2]
b=xaxis_N
x=la.lstsq(M,b)[0]
x[0]=1/x[0]
x[1]=x[1]*x[0]
fit1=np.exp(x[1]+x[0]*b)

C1=np.log(errors[500:])
C2=-1*ones((len(errors[500:])))
M=c_[C1,C2]
```

```
b2=xaxis_N[500:]
x=la.lstsq(M,b2)[0]
x[0]=1/x[0]
x[1]=x[1]*x[0]
fit2=np.exp(x[1]+x[0]*b)
```

This equation is solved twice: once for `Niter` in the range $\{0, 1, 2, ..., 1500\}$ and once more for `Niter` beginning from 500, i.e, $\{500, 501, 501, ..., 1500\}$. **We use `lstsq` to solve the equation as the method of obtaining the best-fit coefficients.**

Upon solving the equation, we get the values of $\frac{1}{B}$ and $\frac{\log A}{B}$. The lines

```
x[0]=1/x[0]
x[1]=x[1]*x[0]
```

in the code convert these to B and A respectively.

As a result, we get two best-fit curves: one for the errors in the `Niter` range $\{0, 1, 2, ..., 1500\}$ and one more for the errors in the `Niter` range $\{500, 501, 501, ..., 1500\}$. These two curves are labelled `fit1` and `fit2` and plotted on the same *semilog*(y) plot.

We find that these curves align almost perfectly, as seen below.

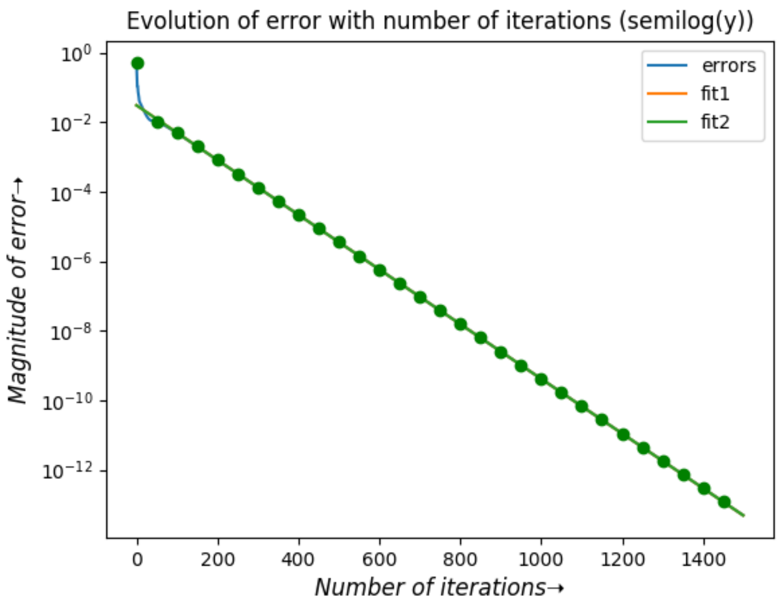

Figure 2: Evolution of errors with number of iterations on the *semilog*(y) scale

In the above figure, the blue, green and orange lines all almost overlap.

When we zoom into the semilog graph, we clearly see the three different lines as below:

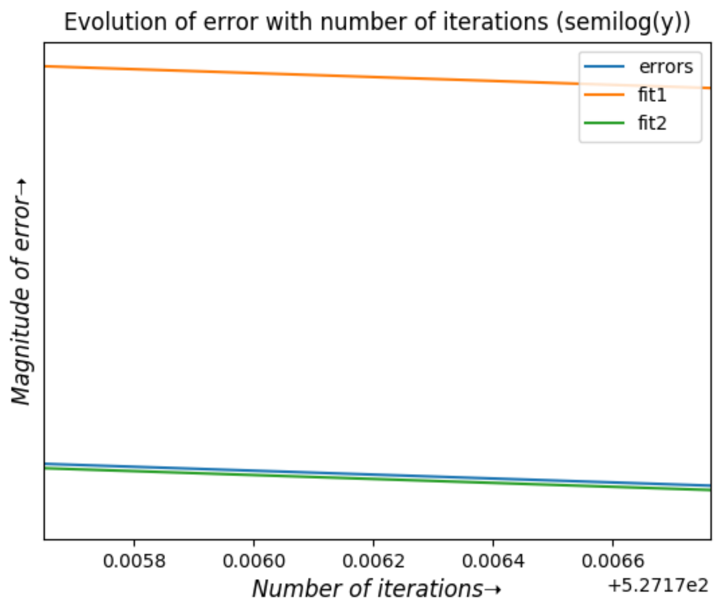

Figure 3: Evolution of errors with number of iterations on the *semilog*(y) scale: ZOOMED IN

Here, it is visible that a linear graph is obtained in the *semilog*(y) plot.

However, in the *loglog* plot, we see that the curve we obtain decreases almost linearly until `Niter` ≈ 500, and starts decreasing exponentially after that. This holds for all three curves. We observe this in the plot below.

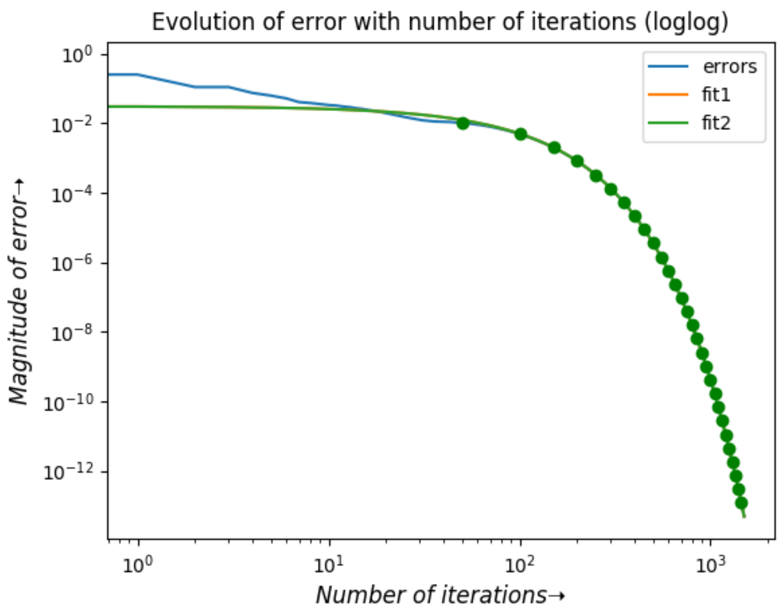

Figure 4: Evolution of errors with number of iterations on the *loglog* scale

Stopping Condition

We see that the errors reduce very slowly, which makes this method cumbersome and very non-ideal.

For example, after **600** iterations, we see that the error is still somewhat significant (5.93×10^{-7}) and after **800** iterations, it has still not decreased much (1.59×10^{-8}).

In any case, we can set a threshold of acceptable per-iteration error at which we wish to stop, and include a block of code along the lines of:

```
if errors[k]<=threshold:
    break
```

to stop the code from doing further iterations once we have reached acceptable potential values. We can also use `scipy.integrate.quad` and perform:

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

where $y = Ae^{Bx}$ is our best fit curve of the per-iteration errors, to obtain the *total* error with respect to the final value of potential. Now, we can set a threshold for this *total* error and `break` once we reach the threshold.

Surface Plot of the Potential

```
fig1=figure(4)
ax=p3.Axes3D(fig1)
title('The 3-D surface plot of the potential')
surf = ax.plot_surface(X,Y, phi[-1::-1,:], rstride=1, cstride=1, cmap=cm.jet)
ax.set_xlabel('$X$', size=20)
ax.set_ylabel('$Y$', size=20)
ax.set_zlabel('$\phi$', size=20)
show()
```

The above code is used to generate our 3-D surface plot, and the plot generated looks like this:

In the code, we use `ax.set_xlabel` instead of simply `xlabel`, because the former is the syntax for axis naming in the case of 3-D surface plots. We plot `phi[-1::-1,:]` instead of just `phi`, because in our (i, j) system of indexing, **increasing** i corresponds to **decreasing** y along the y-axis. Therefore, we flip the rows of the `phi` array in order to correctly plot the 3-D surface plot.

The `ax.plot_surface` function does the actual job of plotting. In our `ax.plot_surface` function call, the parameter

- `rstride` refers to **array row stride (step size)**
- `cstride` refers to **array column stride (step size)**
- `cmap` refers to **colormap for the surface patches**

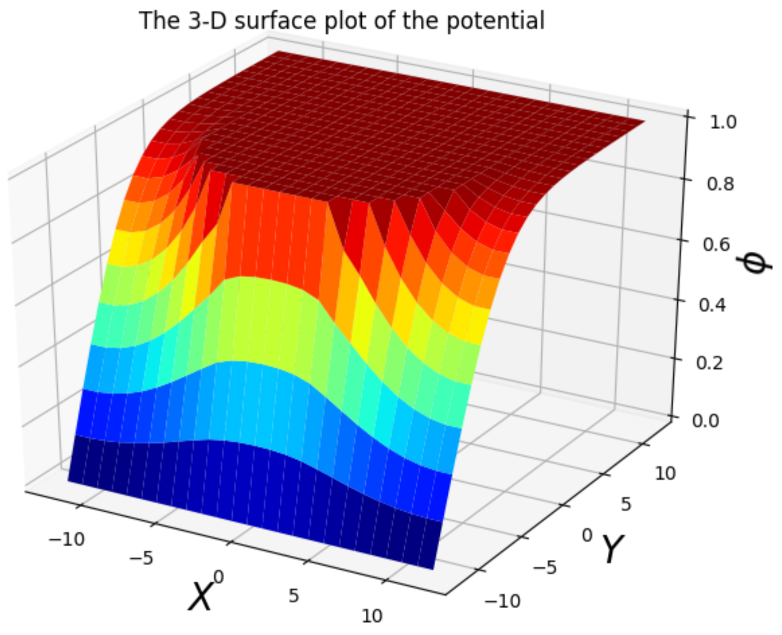

Figure 5: The 3-D surface plot of the potential

Contour Plot of the Potential

```
cp=contour(X,Y,phi[-1::-1,:])
clabel(cp, cp.levels[0:7], inline=True, fontsize=7)
plot(ii[1]-(Nx-1)/2,ii[0]-(Ny-1)/2,'ro',label='V = 1')
xlabel('$X\u279d$', size=15)
ylabel('$Y\u279d$', size=15)
title('The contour plot of the updated potentials')
legend(loc='upper right')
show()
```

The above code is used to generate the contour plot of the updated potentials on the resistor plate. The $V = 1$ points in contact with the electrode are kept at a constant potential, and once again marked with red circles. The plot looks like this:

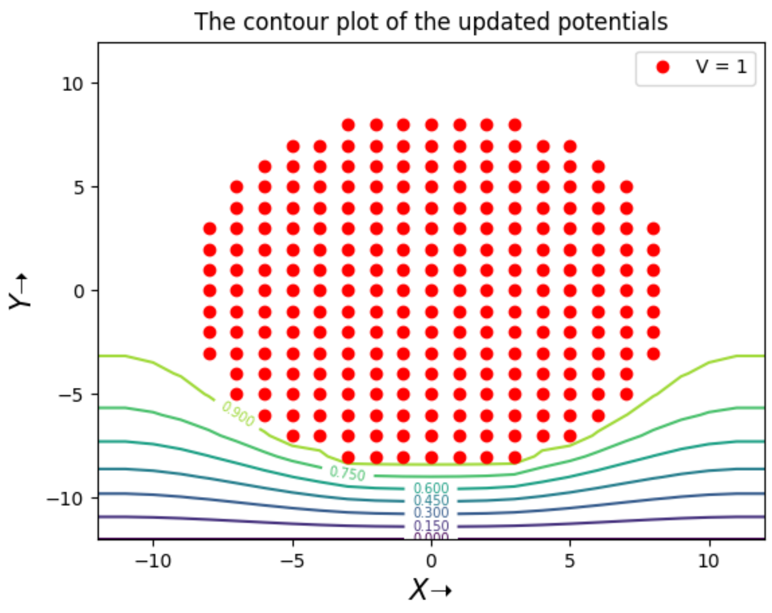

Figure 6: The contour plot of the updated potentials

Once again, we plot the contours for `phi[-1::-1,:]` instead of just `phi` for the same reason as in the plotting of the 3-D surface plot.

Vector Plot of Currents

In order to plot the currents at each point as a vector plot, firstly we evaluate the current values. For this, we create matrices `Jx` and `Jy` to contain the current values at each point, and first initialise it to 0 with:

```
Jx=zeros((Ny,Nx))
Jy=zeros((Ny,Nx))
```

In order to obtain the **shape** of the current profile, the actual value of σ does not matter. Therefore we use the following equations assuming $\sigma = 1$:

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

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

for the current values. Numerically, these equations come out to be:

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

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

which are then implemented in code as:

```
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])
```

Ultimately, we plot the vector plot using the following code:

```
quiver(X,Y,Jx[-1::-1,:],Jy[-1::-1,:], scale=5)
plot(ii[1]-(Nx-1)/2,ii[0]-(Ny-1)/2,'ro', label='V = 1')
xlabel('$X\u279d$', size=15)
ylabel('$Y\u279d$', size=15)
title('The vector plot of the current flow')
show()
```

and the plot we obtain looks like this:

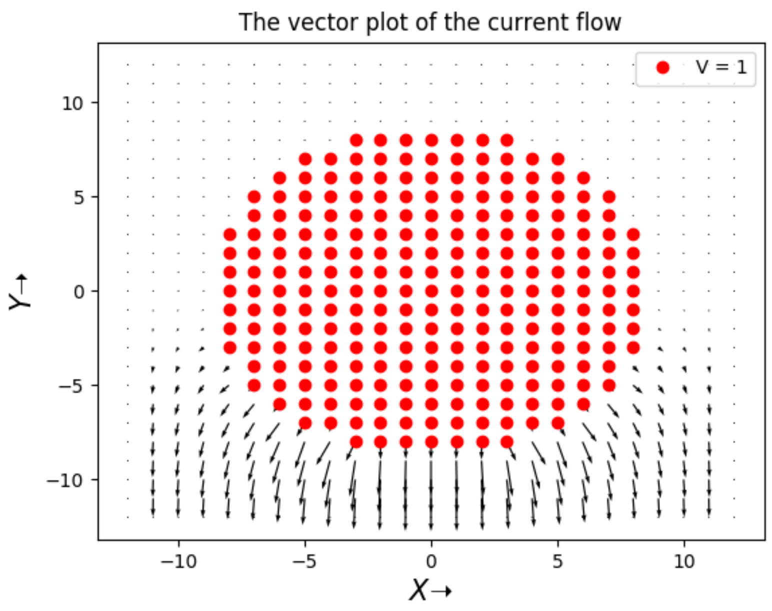

Figure 7: The vector plot of the current flow

Hardly any current flows out of the top part of the wire.

This is because the only the bottom side of the resistor is earthed/fixed at 0 potential. The other sides are left open. Now, current always flows from higher potential to lower potential. When we fix the electrode at $V = 1$ and the bottom edge at $V = 0$, we are maintaining a relatively large potential difference between these two regions. That is why most of the current flows through this region.

Our implementation of the Laplace Equation has kind of an averaging effect in the regions between points which have fixed values. In this case, the bottom half of the resistor lies between $V = 1$ and $V = 0$, and so the values of the potentials vary progressively and rapidly from 1 to 0. However in the top half, there is no fixed region of potential between which to average, therefore all points reach $V \approx 1$ (the approximate equality holds because each point still belongs to a path between $V = 1$ and $V = 0$, therefore tiny currents still flow at all these top-half points). Therefore, according to the Ohm's Law:

$$I = \frac{V}{R}$$

and the current in the top half is very small because of very small potential differences between points in the top half. If we didn't fix $V = 0$ at the bottom edge, then even the bottom half would get the same values of potential (≈ 1) as the top half, and even the bottom half would have the same tiny insignificant currents.

Calculating and Plotting Heat Generated and Temperature

Heat

Presently, we are not interested in the exact value of heating. We simply wish to observe the general trend of heat generated on the plate due to the currents, and the corresponding temperature rises across the plate. We use our previous assumption that the conductivity of the plate material $\sigma = 1$.

The ohmic loss $(\vec{J} \cdot \vec{E})$ in the plate at any point is represented by:

$$\begin{aligned} Q &= \frac{|J|^2}{\sigma} \\ &= |J|^2 \end{aligned}$$

where J is the current value and σ is the conductivity.

We implement this in code using:

```
Q=Jx**2+Jy**2
Q[ii]=0
```

where the last line is used to implement the condition that there is no heat generated at the electrode.

The contour plot of the heat generated is shown below:

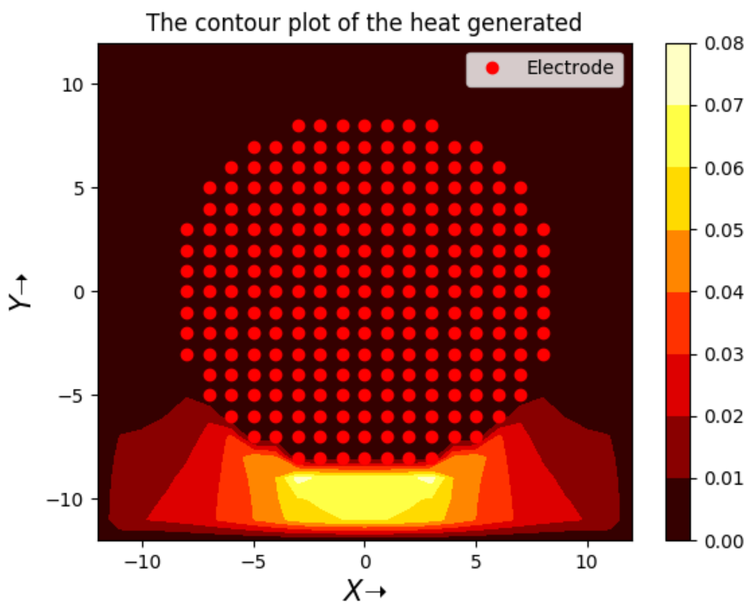

Figure 8: The contour plot of the heat generated

Temperature

The relation between temperature T and ohmic loss Q at a point is given by:

$$\nabla \cdot (-\kappa \nabla T) = Q$$

where κ is the thermal conductivity of the material.

Therefore, we get:

$$-\kappa\nabla^2 T = Q$$
$$-\kappa(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}) = Q$$
$$Q + \kappa\frac{\partial^2 T}{\partial x^2} + \kappa\frac{\partial^2 T}{\partial y^2} = 0$$

Once again, similar to what we did for the potential ϕ, we have the numerical forms of the differentials:

$$\left.\frac{\partial^2 T}{\partial x^2}\right|_{(x_i,y_j)} = \frac{T(x_{i+1},y_j) - 2T(x_i,y_j) + T(x_{i-1},y_j)}{(\Delta x)^2}$$
$$\left.\frac{\partial^2 T}{\partial y^2}\right|_{(x_i,y_j)} = \frac{T(x_i,y_{j+1}) - 2T(x_i,y_j) + T(x_i,y_{j-1})}{(\Delta y)^2}$$

Substituting these values, we get:

$$Q + \kappa\frac{T(x_{i+1},y_j) - 2T(x_i,y_j) + T(x_{i-1},y_j)}{(\Delta x)^2} + \kappa\frac{T(x_i,y_{j+1}) - 2T(x_i,y_j) + T(x_i,y_{j-1})}{(\Delta y)^2} = 0$$

$$\implies T_{i,j} = \frac{\frac{Q(\Delta x)^2(\Delta y)^2}{\kappa} + T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1}}{4}$$

For the sake of this problem, we assume that $\kappa = 0.01$.

We have more information regarding this scenario: the boundary condition $\frac{\partial T}{\partial n}_{normal} = 0$ at the open boundaries, and $T = 300K$ (room temperature) at the electrode and at the ground.

All of the above is implemented in code similar to the potential ϕ calculation, as follows:

```
T=zeros((Ny,Nx))
T[ii]=300
T[-1]=300
deltax=1/Nx
deltay=1/Ny
for k in range(Niter):
    T[1:-1,1:-1]=0.25*(T[1:-1,0:-2]+T[1:-1,2:]+T[0:-2, 1:-1]+T[2:,1:-1]
    +(Q[1:-1,1:-1]*(deltax**2)*(deltay**2))/0.01)
    T[1:-1,0]=T[1:-1,1]
    T[1:-1,-1]=T[1:-1,-2]
    T[0,:]=T[1,:]
    T[ii]=300
    T[-1]=300
```

and the contour plot is made. The contour plot of the temperature is shown below:

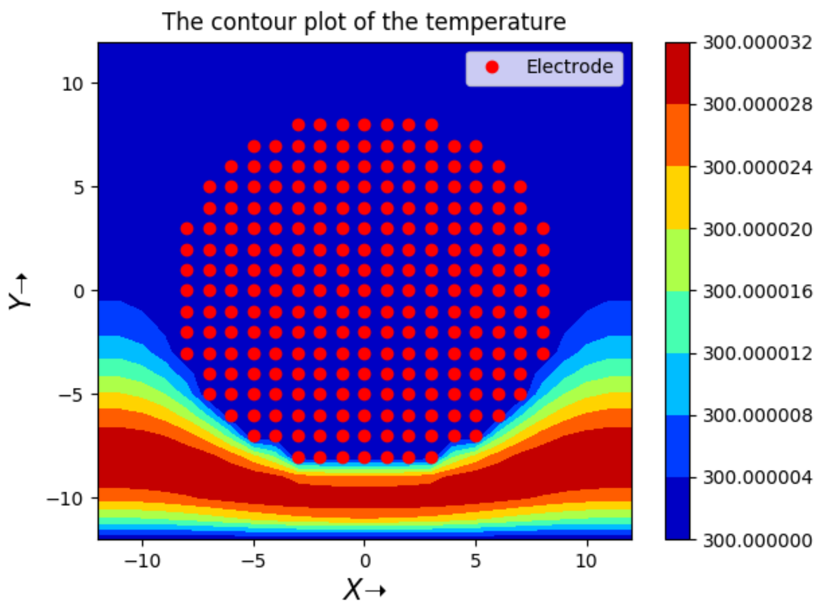

Figure 9: The contour plot of the temperature

3 Conclusion

This Assignment has helped me learn the following:

- Solving a resistor problem for a resistor of finite dimensions (square) using the Laplace Equation as a special case of the Continuity Equation.
- Applying the Laplace Equation numerically, and understanding that the errors decrease very slowly with every iteration. **Therefore, we learn that this method is bad and cumbersome, and requires a large number of iterations before we arrive at a suitable stopping condition.**
- Plotting 3-D surface plots and vector plots for a matrix quantity (the potential ϕ).
- Dealing with the heat generated and temperature calculations in a resistor problem.

****