Indian Institute of Technology, Madras
Department of Electrical Engineering
Applied Programming Lab

Lab Report
Assignment 5

Laplace Equation

Nithin Uppalapati
EE18B035

01-03-2020

Contents

1 Abstract **1**

2 Introduction **1**

3 Theory **2**

3.1 Calculating ϕ for the Conductor: . . . 3

3.2 Computing the Errors in each Iteration: . . . 4

3.3 Computing the Current Density: . . . 4

3.4 Least Squares Approach to fit the Errors to the Exponential Form: . . . 5

3.4.1 Fitting the Entire Vector of Errors: . . . 5

3.4.2 Fitting the Errors for the number of iterations exceeding 500: . . . 5

3.4.3 Least Squares Method . . . 5

3.5 Plots of Functions and Coefficients: . . . 7

3.5.1 Semilog plot of Errors for all iterations along with the Estimated Functions: . . . 7

3.5.2 Loglog plot of Errors for all iterations along with the Estimated Functions: . . . 8

3.5.3 The 3-D surface plot of the Potential - ϕ: . . . 9

3.5.4 Contour plot of ϕ_{ij} : . . . 10

3.5.5 Vector plot of Current Density: . . . 11

4 Conclusions **12**

4.1 On Current Density: . . . 12

1. Abstract

Aim :
The aim of this assignment is to solve for the currents and equipotential lines in the resistor. And the conductor, fro which we are solving the currents is of square shape of dimensions $1cm$x$1cm$.

2. Introduction

Assignment 5 is based on:

- Calculating the Poisson's equation by discretization of the differential equation.
- Analysing the error with every iteration, and fitting the error to a exponential function of the form: Ae^{Bx}, where x is the number of iteration performed.
- Plotting graphs using the pylab library for both of the potential contours, errors in loglog scale and semilogy scale, and $3D$ plot of the potential on the conductor.
- And plotting the current density vectors with the quiver command.

Implementation :

- So, in order to find the current density in the conductor, we solve the Laplace's equation under proper boundary conditions.
- And use lstsq method for fitting the errors to the function.
- Asserting proper boundary conditions for each iteration.

3. Theory

Electrodynamic Equations for a conducting conductor :
Let assume that the conductivity of a conductor is σ.
Then the Ohm's law can be represented as:

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

Now the Electric field is the gradient of the potential,

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

and continuity of charge yields,

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

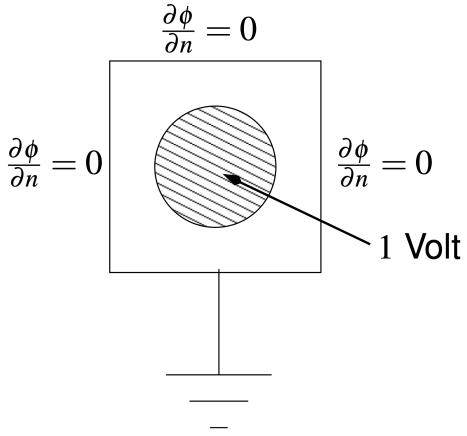
Combining these equations we obtain and assuming that our resistor contains a material of constant conductivity, the equation becomes

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

As we are dealing the DC supply, and as at the steady state the charge density doesn't vary with time, thus

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

which is the Poisson's equation.



3.1 Calculating ϕ for the Conductor:

Laplace's equation is easily transformed into a difference equation.
The difference equation can be given as:

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

and this can be easily implemented in python without using nested for loop as shown below. This is possible to perform by using vectorized for loops.
Initializing the array of phi:

$$\phi = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{3.1}$$

```
Niter=1500    ### no. of iterations to be performed
phi = array([zeros(Ny)]*Nx)
for i in range(0,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])
errors[i]=(abs(phi-oldphi)).max() ##calc. of errors with each iteration
```

where,
ii corresponds to the points in contact with the positive terminal of the supply in contact with the conductor and it is given using the *where*() function as follows:

```
ii=where((X*X+Y*Y)<=(radius**2))
```

Specifying the resolution of the required potential on the conductor and the number of iterations to be performed is given through the input from the user. If it's not given, the defaults values we assume for calculating the potential is given as follows:

```
Nx=25; ## size along x
Ny=25; ## size along y
radius=8;  ## radius of central lead
Niter=1500; ## number of iterations to perform
```

And finally we have to assert the boundary conditions for each iteration :

```
phi[ii]=1.0
phi[-1]=0
phi[1:-1,0]=phi[1:-1,1]
phi[1:-1,-1]=phi[1:-1,-2]
phi[0]=phi[1]
```

3.2 Computing the Errors in each Iteration:

The coefficients are stored in the fashion,

$$errors = \begin{bmatrix} e_0 & e_1 & e_2 & e_3 & \cdots & e_{1498} & e_{1499} \end{bmatrix}$$

Where, e_i is the error while iterating the loop for i^{th} time.
And this calculated as follows:

```
errors=zeros(Niter)
errors[i]=(abs(phi-oldphi)).max()   ##is executed in the for loop
```

3.3 Computing the Current Density:

The Current Density $\vec{j}$, is defined as follows:

$$\vec{j_x} = \sigma.(-\frac{\partial \phi}{\partial x})$$

As we are to plot the current density vectors, so we drop the σ term as it contributes to the magnitude of vectors only.
And this is numerically translated as follows:

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

And this is implemented in python as follows:

```
j_x = array([zeros(Ny)]*Nx)
j_y = array([zeros(Ny)]*Nx)

j_x[:,1:-1]=0.5*(phi[:,0:-2]-phi[:,2:])
j_y[1:-1,:]=0.5*(-phi[0:-2,:]+phi[2:,:]) #calculating currrent density
j_y[-1]=(-phi[-2])
```

3.4 Least Squares Approach to fit the Errors to the Exponential Form:

So, as the errors are stored in the array named - *errors*

We want to fit the errors to the form:

$$e = Ae^{Bx}$$

Where, x is the number of iterations, and e is the error. Thus, we need to calculate the A and B constants for the acquired errors.

The above equation can be modified to the form:

$$log(e) = log(A) + Bx$$

Thus, we need tho fing the values of log(A) and B by giving the $log(errors)$ to the *lstsq* funciton.

As the coefficient of $log(A)$ is 1, and the coefficient of B is e, so we create the $2D$ arrays as follows:

```
 samples=array([zeros(2)]*Niter)
samples[:,0]=1
iter=arange(1,Niter+1,1)
samples[:,1]=iter
```

3.4.1 Fitting the Entire Vector of Errors:

```
err_coeff=lstsq( samples , log(errors) ,rcond=-1)[0]
A=exp(err_coeff[0])
B=err_coeff[1]
```

3.4.2 Fitting the Errors for the number of iterations exceeding 500:

```
err_coeff_500=lstsq( samples[500:] , log(errors[500:]) ,rcond=-1)[0]
A_500=exp(err_coeff[0])
B_500=err_coeff_500[1]
```

3.4.3 Least Squares Method

The method of least squares is a standard approach in regression analysis to approximate the solution of overdetermined systems (sets of equations in which there are more equations than unknowns) by minimizing the sum of the squares of the residuals made in the results of every

single equation. The most important application is in data fitting. The best fit in the least-squares sense minimizes the sum of squared residuals (a residual being: the difference between an observed value, and the fitted value provided by a model).

So, as the coefficients are varied to minimize the error, we differentiate the error wrt. each coefficient and equate it to zero, in order to obtain the vector of optimal coefficients ($p_i's$).

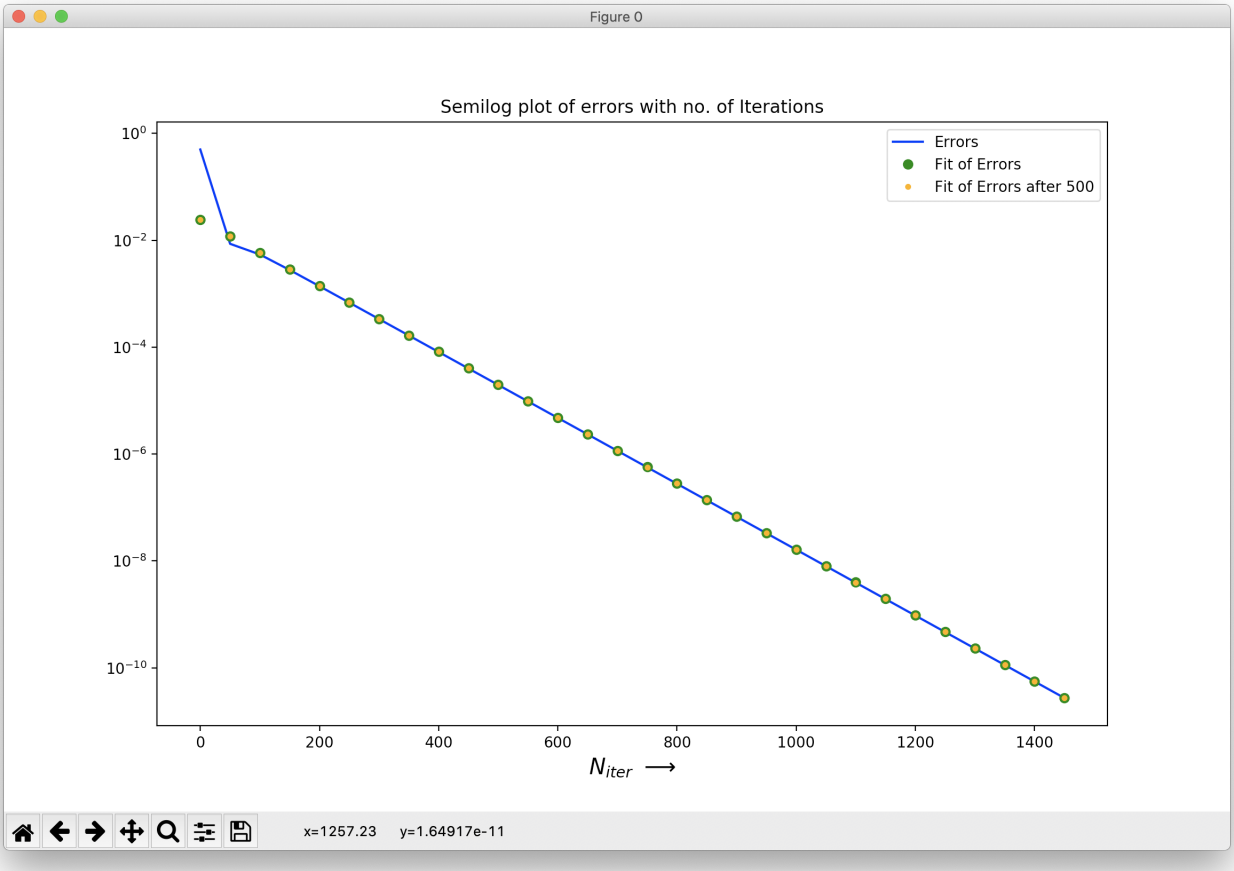
And in python, in scipy.linalg library, there is a dedicated function, named $lstsq()$ [leastsqares], which accepts the arguments as : $lstsq(F, x)$

3.5 Plots of Functions and Coefficients:

```
n=arange(0,Niter,50)
```

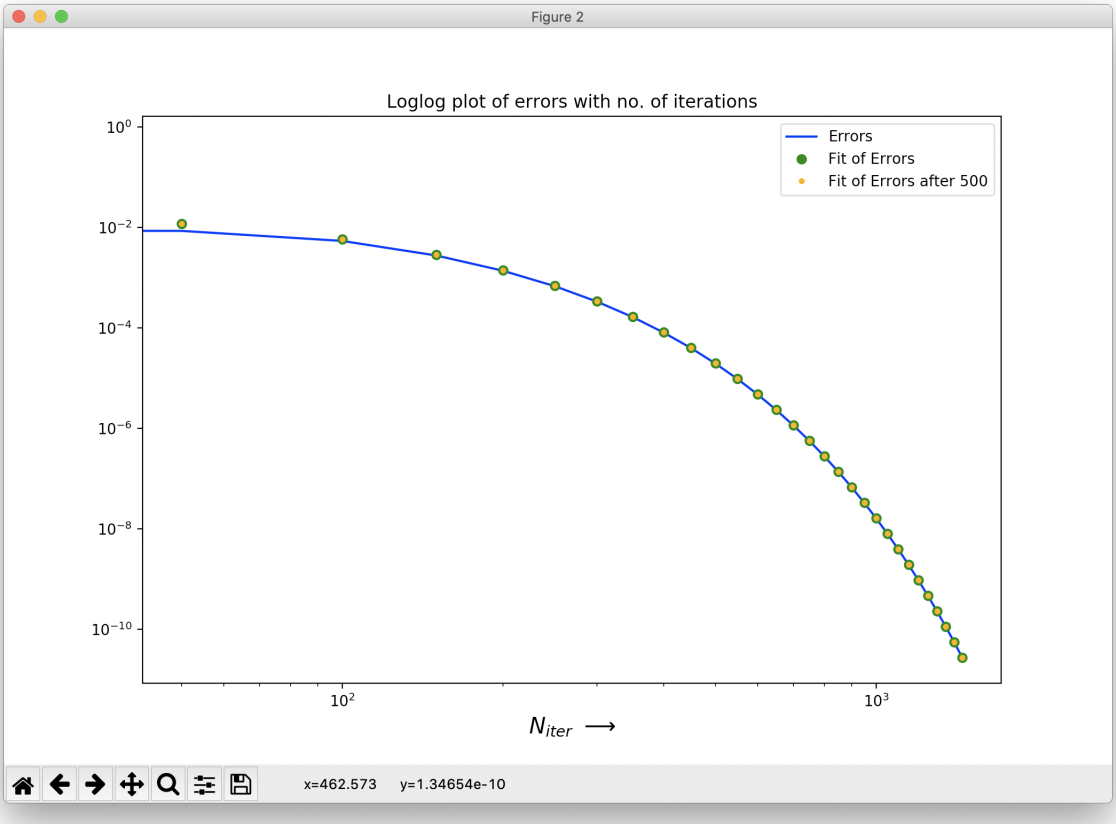
3.5.1 Semilog plot of Errors for all iterations along with the Estimated Functions:

```
title('Semilog plot of errors with no. of Iterations')
semilogy(n,errors[::50],color='blue',label= 'Errors')
semilogy(n,e(n,A,B),color='green',label= 'Fit of Errors')
xlabel(r'$N_{iter}$' ' ' '$\longrightarrow$',size=15)
semilogy(n,e(n,A_500,B_500),color='orange',label= 'Fit of Errors after 500')
legend(loc="upper right")
```



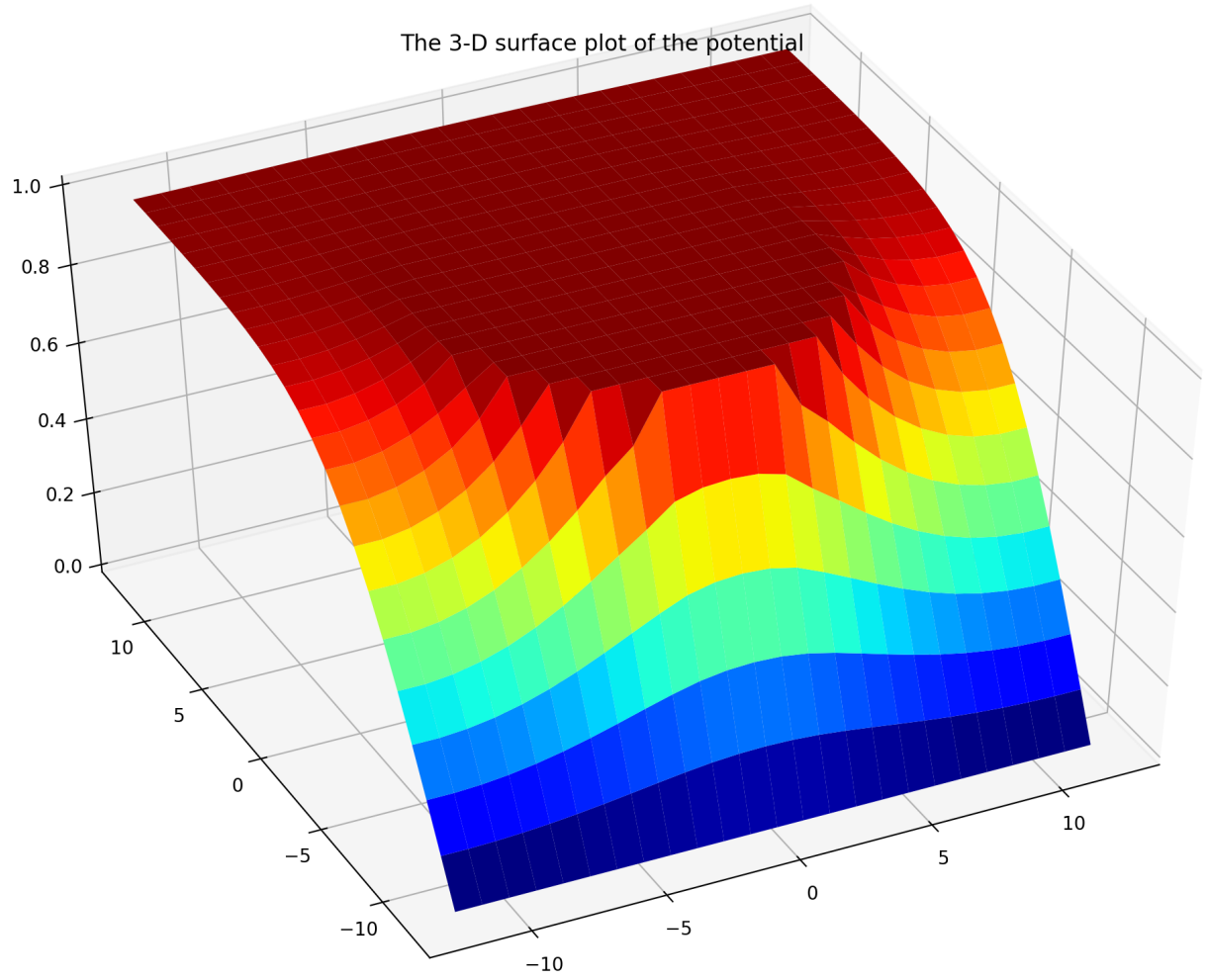
3.5.2 Loglog plot of Errors for all iterations along with the Estimated Functions:

```
title('Loglog plot of errors with no. of iterations')
loglog(n,errors[n],color='blue',label= 'Errors')
loglog(n,e(n,A,B),color='green',label= 'Fit of Errors')
xlabel(r'$N_{iter}$' ' ' '$\longrightarrow$',size=15)
loglog(n,e(n,A_500,B_500),color='orange',label= 'Fit of Errors after 500')
legend(loc="upper right")
```



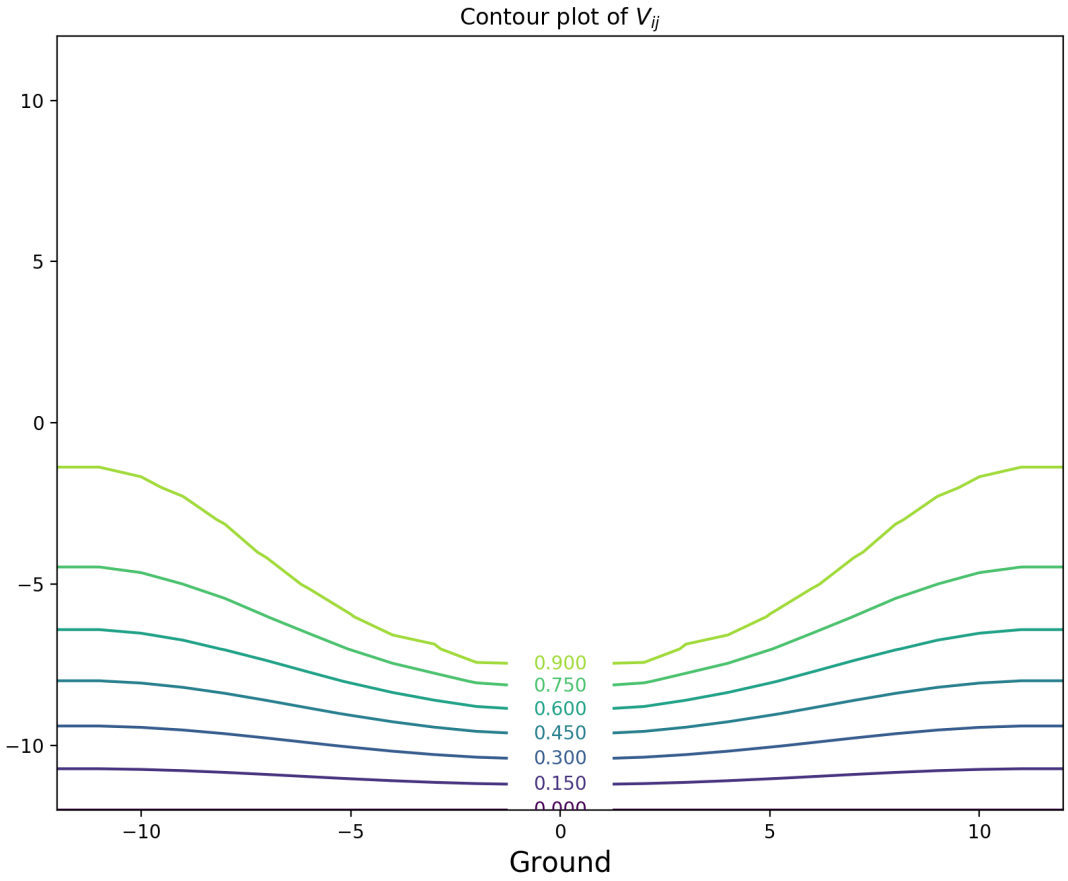
3.5.3 The 3-D surface plot of the Potential - ϕ:

```
fig4=figure(4) # open a new figure
ax=p3.Axes3D(fig4) # Axes3D is the means to do a surface plot
title('The 3-D surface plot of the potential')
surf = ax.plot_surface(-Y, -X, phi, rstride=1, cstride=1, cmap=cm.jet)
```



3.5.4 Contour plot of ϕ_{ij} :

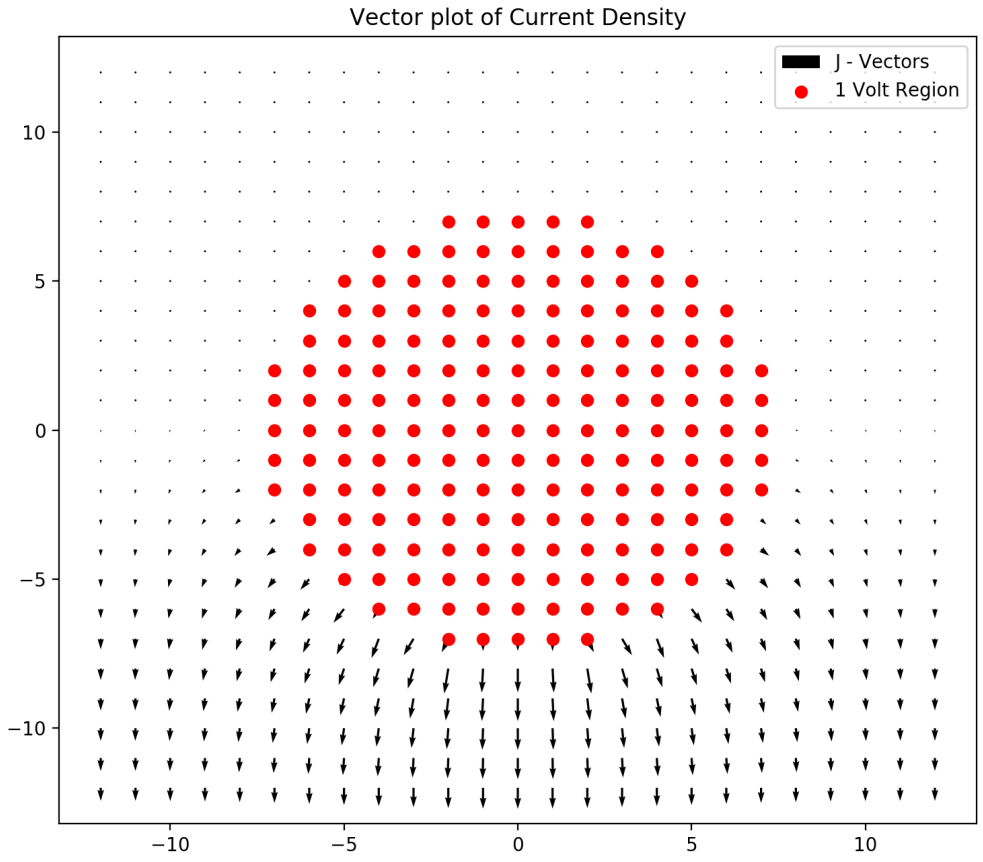
```
cp = plt.contour(Y, -X, phi, linestyles='-')
clabel(cp, inline=True,fontsize=10)
xlabel('Ground',size=15)
title(r'Contour plot of $V_{ij}$ ' )
```



3.5.5 Vector plot of Current Density:

```
title('Vector plot of Current Density')
q=quiver(-y,-x,j_x,j_y, scale=1.2, scale_units='inches',label="J - Vectors")
scatter(X[ii],Y[ii],color='red',label="1 Volt Region")
legend(loc="upper right")
show()
```

As, we also need to plot the $1V$ potential region, we use the scatter command to do that.



4. Conclusions

4.1 On Current Density:

As the entire current flows through the ground, thus the lower part has a more density of the current, where as the top plate is like a *floating node*-as it is not connected to any other reference potential node, so the current doesn't flow out of the top edge.
Thus the current which will go in the upper half of the plate, must return downwards, as no charge gets accumulated in the case of steady state assumption.
Hence the $|\vec{j}|$ is more denser in the bottom plate and less denser in the upper plate.