# EE2703: Assignment 5

Sarthak Vora (EE19B140)

March 25, 2021

## 1 Introduction

In this report, we try to analyse the currents in a square copper. We will also discuss how to find stopping condition after certain iterations. Subsequently, we will model the errors obtained using Least Squares after analysing the actual errors in **semilog** and **loglog** plots. Finally we find the currents in the resistor after applying boundary conditions and analyse the vector plot of current flow and conclude which part of resistor will become hot.

- A wire is soldered to the middle of a copper plate and its voltage is held at 1 Volt. One side of the plate is rounded, while the remaining are floating. The pblate is 1 cm by 1 cm in size.
- To solve for currents in resistor, we use following equations and boundary conditions mentioned below:
- Conductivity (Differential form of ohm's law)

$$\vec{J} = \sigma \vec{E} \tag{1}$$

- Electric field is the gradient of the potential

$$\vec{E} = -\nabla \phi \tag{2}$$

- Charge Continuity equation is used to conserve the inflow and outflow of charges

$$\nabla . \vec{J} = -\frac{\partial \rho}{\partial t} \tag{3}$$

- Combining the above equations above, we get

$$\nabla . (-\sigma \nabla \phi) = -\frac{\partial \rho}{\partial t} \tag{4}$$

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

$$\nabla^2 \phi = \frac{1}{\sigma} \frac{\partial \rho}{\partial t} \tag{5}$$

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

$$\nabla^2 \phi = 0 \tag{6}$$

- Here we use a 2-D plate, so the Numerical solutions in 2D can be easily transformed into a difference equation. The equation can be written out as

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0 \tag{7}$$

$$\frac{\partial \phi}{\partial x}_{(x_i, y_j)} = \frac{\phi(x_{i+1/2}, y_j) - \phi(x_{i-1/2}, y_j)}{\Delta x} \tag{8}$$

$$\frac{\partial^2 \phi}{\partial x^2}_{(x_i, y_j)} = \frac{\phi(x_{i+1}, y_j) - 2\phi(x_i, y_j) + \phi(x_{i-1}, y_j)}{(\Delta x)^2} \tag{9}$$

- Using above equations we get

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4} \tag{10}$$

- Thus, 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. So the solution process is to take each point and replace the potential by the average of its neighbours. Keep iterating till the solution converges (i.e., the maximum change in elements of $\phi$ which is denoted by $error_k$ in the code ,where 'k' is the no of iteration, is less than some tolerance which is taken as $10^{-8}$).
- At boundaries where the electrode is present, just put the value of potential itself. At boundaries where there is no electrode, the current should be tangential because charge can't leap out of the material into air. Since current is proportional to the Electric Field, what this means is the gradient of $\phi$ should be tangential. This is implemented by requiring that $\phi$ should not vary in the normal direction.
- At last, we solve for currents in the resistor using the above mentioned concept.

---

## 2 Procedure

### 2.1 Part A

#### 2.1.1 Procedure

- Firstly, we define the parameters $N_x$, $N_y$ and $N_{iter}$. The default values of these parameters are $N_x$=25, $N_y$=25 and $N_{iter}$=1500.
- Allocate the **potential array** $\phi$=0. Note that the array should have $N_y$ rows and $N_x$ columns.
- To find the datapoints which lie inside the circles of radius 0.35 using *numpy.meshgrid()* function and using the equation:
$$X^2 + Y^2 \leq 0.35^2 \tag{11}$$
- Assign 1V to the corresponding indices satisfying the above condition.
- Plot a contour plot for the potential $\phi$.

#### 2.1.2 Code

```
# Initializing the Electric Potential
phi = zeros((Ny,Nx))

# Defining the coordinates and scale
x = linspace(-0.5,0.5,Nx)
y = linspace(0.5,-0.5,Ny)
Y,X = meshgrid(x,y)

# Finding the desired coordinates
ii = where(square(X)+square(Y)<=radius**2)
phi[ii]=1.0

# Plotting Contour Plot
figure('Figure 1: Contour Plot of potential')
contourf(phi,cmap=cm.jet)
title('Figure 1: Contour plot of potential $\phi$')
xlabel(r'x $\rightarrow$')
ylabel(r'y $\rightarrow$')
show()
```

### 2.1.3 Plots

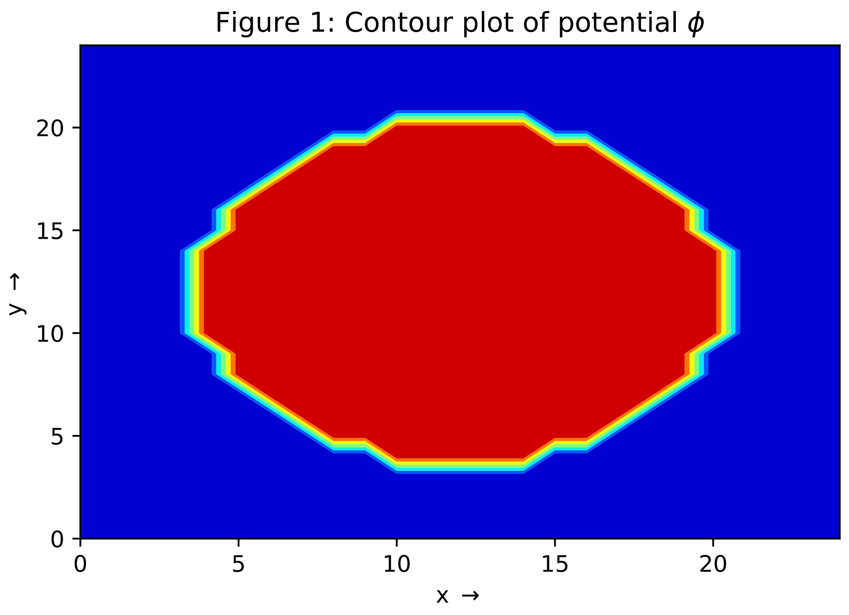


Figure 1 : Contour Plot of Potential $\phi$

### 2.1.4 Observation

- The contour plot of potential becomes smoother i.e it almost becomes circular as we increase $N_x$ and $N_y$. This is due to more number of datapoints and hence the potential gradient smoothens out between adjacent points.

---

## 2.2 Part B

### 2.2.1 Procedure

- Performing iterations using $N_{iter}$ parameter.
- Updating the **potential** $\phi$ according to the below equation:

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4} \tag{12}$$

- The gradient of $\phi$ should be **tangential** where there is **no electrode**. Hence, it means that there should be no normal variation in potential so we equate the last row or column to outermost row or column correspondingly whille applying boundary conditions.

$$\frac{\partial \phi}{\partial n} = 0 \tag{13}$$

- Finally, plotting the errors in *semilog* axis and *loglog* axis to observe the **variation** of errors.

### 2.2.2 Code

```
# Initialing the error and finding the error
errors = zeros((Niter))
for i in range(Niter):
    oldphi = phi.copy()

    #Updating potential
    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])

    # Boundary conditions for potential
    phi[1:-1,0] = phi[1:-1,1] # Left Edge
    phi[1:-1,Nx-1] = phi[1:-1,Nx-2] # Right edge
    phi[0,:] = phi[1,:] # Top Edge
    # Bottom Edge is already grounded

    phi[ii] = 1.0 # Assigning 1V to electrode region

    # Calculating error
    errors[i] = (abs(phi-oldphi)).max()

iterations = arange(0,Niter,1)

# Plotting loglog plot
figure('Figure 2: loglog plot of Error')
loglog(iterations,errors,'r')
ylabel(r'log(Error) $\rightarrow$')
xlabel(r'log(iterations) $\rightarrow$')
title('Figure 2: Loglog plot of error')
grid()
show()

# Plotting semilog plot
figure('Figure 3: Semilog plot of error')
semilogy(iterations,errors,'r')
ylabel(r'log(Error) $\rightarrow$')
xlabel(r'No. of iterations $\rightarrow$')
title('Figure 3: Semilog plot of error')
grid()
show()
```

### 2.2.3 Plots

Figure 2: Loglog plot of error

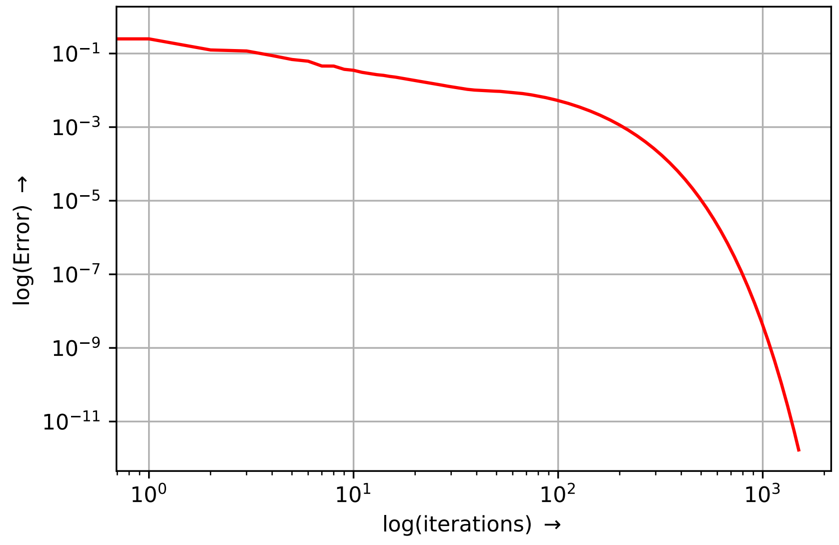
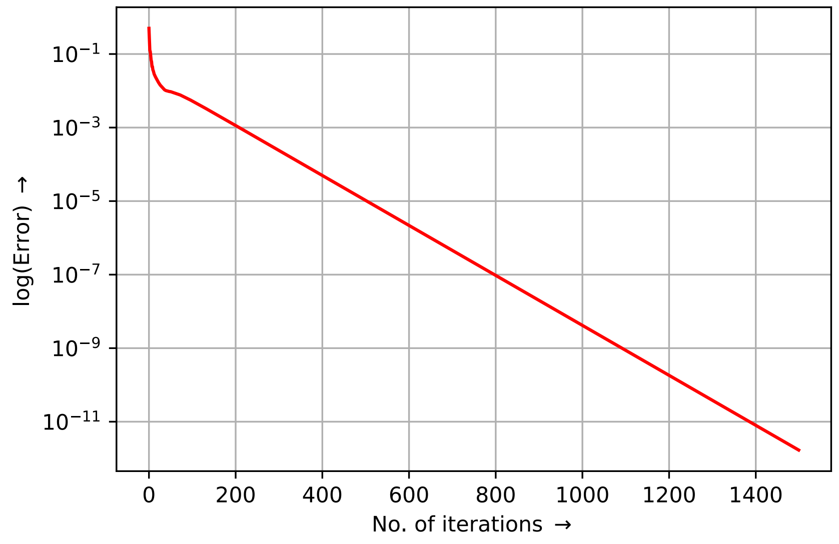


Figure 3: Semilog plot of error



### 2.2.4 Observations

- As we can see from the *semilog* plot above, the error **decreases** linearly with increase in number of iterations. Hence, we can say that, for large iterations values, the variation in error is similar to decaying **exponential** i.e of the form $Ae^{Bx}$ as can be seen from the *semilog* plot.
- The error is almost **decreasing linearly** for *loglog* plot for smaller number of iterations, hence it follows an expression of the form $c^x$ for smaller iterations.
- As a result of above observations, the error follow $Ae^{Bx}$ for larger iterations ($\approx 500$) and it follows $c^x$ form for smaller values of iterations.

---

## 2.3 Part C

### 2.3.1 Procedure

- Finding the fit using Least squares method for all iterations, named as **fitAll** and for iterations $\geq 500$ named as **fitAfter500** separately.
- As we discussed before, the error follows $Ae^{Bx}$ variation at large iterations. Hence, we use the below equation to fit the errors using least squares method:

$$\log(y) = \log(A) + Bx \tag{14}$$

- To find the **time constant** of error function obtained for the two cases using *lstsq()* and compare them.
- Plotting both the calculated **fits** and observing the variation.

### 2.3.2 Code

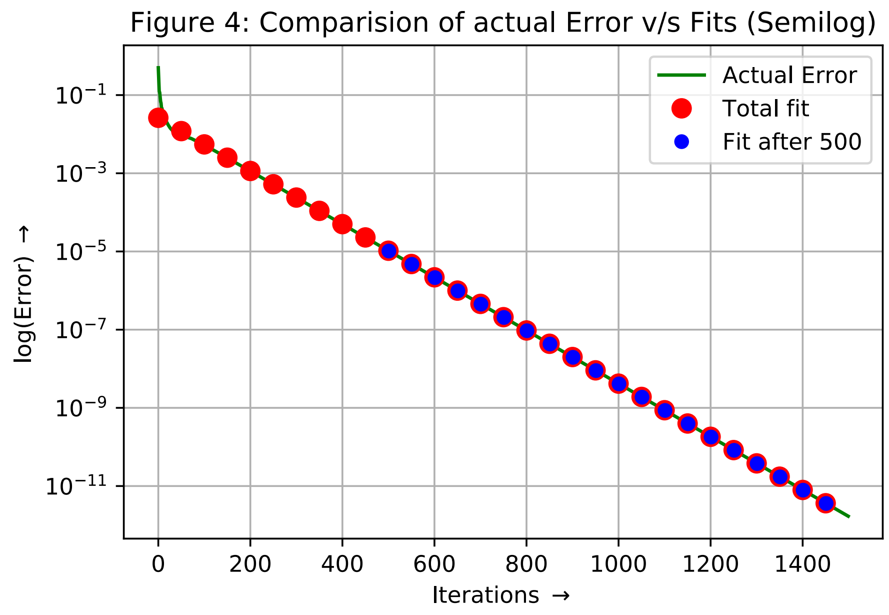
```
# Defining the best fit function
def fit(iterations,errors):
    coeff = zeros((len(errors),2))
    const = zeros((len(errors),1))
    coeff[:,0] = 1
    coeff[:,1] = iterations
    const = log(errors)

    fit = scipy.linalg.lstsq(coeff, const)[0]
    estimate = dot(coeff,fit)
    return fit, estimate

# Finding the fit and estimate of the fit for all values and the last 500 values
fitAll, estimate1 = fit(iterations, errors)
fitAfter500, estimate2 = fit(iterations[501:], errors[501:])
print('A = '+ str(exp(fitAll[0]))+' B = '+str(fitAll[1]) + ' from Fit1')
print('A = '+ str(exp(fitAfter500[0]))+' B = '+str(fitAfter500[1]) + ' from Fit2')

# Plotting the comparision between actual error and fits in semilog
figure('Figure 4: Comparision of actual error and fits (Semilog)')
semilogy(iterations,errors,'g',markersize=6,label='Actual Error')
semilogy(iterations[::50],exp(estimate1[::50]),'ro',markersize=8,label='Total fit')
semilogy(iterations[501::50], exp(estimate2[::50]),'bo',markersize=5,label='Fit after 500')
ylabel(r'log(Error) $\rightarrow$')
xlabel(r'Iterations $\rightarrow$')
title('Figure 4: Comparision of actual Error v/s Fits (Semilog)')
legend()
grid()
show()
```

### 2.3.3 Plots



### 2.3.4 Observations

- **fitAll** : A = 0.026215571 B = -0.015655264
  **fitAfter500** : A = 0.026043988 B = -0.015648069
- As we observe, the **fitAll**'s is slightly higher than **fitAfter500**'s time constant, so the error **decreases** slowly at larger iterations compared to **fitAfter500**.
- Ideally the time constant for **fitAfter500** should be larger than **fitAll**, since less number of datapoints are present i.e stepsize $N_x$ and $N_y$ being less, we get less difference between their time constants, but if we increase the $N_x$ and $N_y$ by 100 times each, then:
  - Time Constant for **fitAll** : 269.467s
  - Time Constant for **fitAfter500** : 269.544s
- As we see that there is a significance difference between the two, since we **increased** the stepsize to 100.
- So the time constant **increase** with increase in $N_x$ and $N_y$.

---

## 2.4 Stopping Condition

### 2.4.1 Procedure

- Finding the **cumulative error** for all iterations and comparing the values with an **error tolerance**, beyond which, the iteration stops.
- To find the **cumulative error**, we add all the absolute values of errors for each iteration, in the worst case, all errors being added up.
- The equations are as given below:

$$Error = \sum_{N+1}^{\infty} error_k \tag{15}$$

- The above error can be approximated to -

$$Error \approx -\frac{A}{B}\exp(B(N+0.5)) \tag{16}$$

N : Number of iterations

### 2.4.2 Code

```
# Defining stopping condition error function
def cumerror(error,N,A,B):
    return -(A/B)*exp(B*(N+0.5))

# Defining a function to find stopping condition
def findStopCondn(errors, Niter, error_tol):
    cum_Err=[]
    for n in range(Niter):
        cum_Err.append(cumerror(errors[n],n, exp(fitAll[0]), fitAll[1]))
        if(cum_Err[n-1] <= error_tol):
            print("last per-iteration change in error is ",(abs(cum_Err[-1]-cum_Err[-2])))
            return cum_Err[n-1], n

    print("last per-iteration change in the error is ", (abs(cum_Err[-1]-cum_Err[-2])))
    return cum_Err[-1], Niter

# Defining error tolerance and finding the stopping condition and error for N=1500(default)
errorTol = 10e-8
cum_Err, nStop = findStopCondn(errors, Niter, errorTol)
print("Stopping Condition ----> N: %g and Error: %g" % (nStop, cum_Err))
```

### 2.4.3 Observations

- We get the stopping condition as **N : 1063** and the total **cumulative error** till that iteration is $9.99924 * 10^{-9}$
- And the last per iteration change in **error**: $1.55321 * 10^{-9}$
- So we observe that the profile changes slowly after every iteration, but it keeps on **varying** continuously. Hence, the cumulative error is considerably **large**.
- Hence, we can conclude that the **Laplace's Equation** solving method is **not** one of the efficient solving techniques. This is because of the very slow coefficient with which the error reduces.

---

## 2.5 Part D

### 2.5.1 Procedure

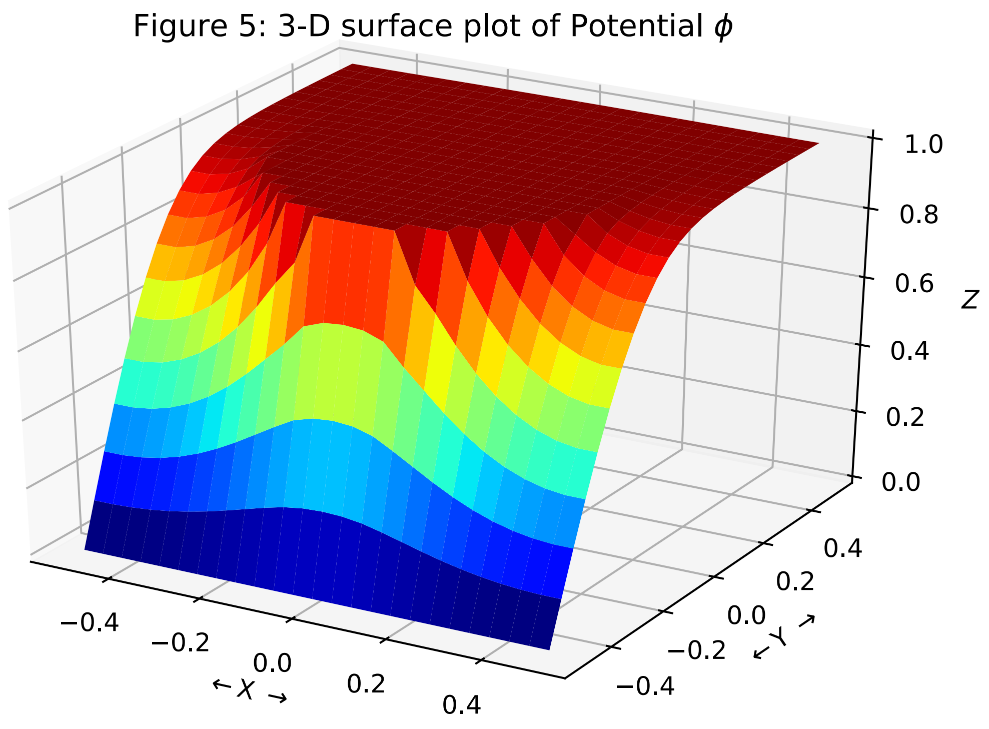
- Plotting a **3-D Surface Plot** of the potential $\phi$.
- Plotting a **Contour plot** of the potential $\phi$.
- Analysing the above two plots and making observations.

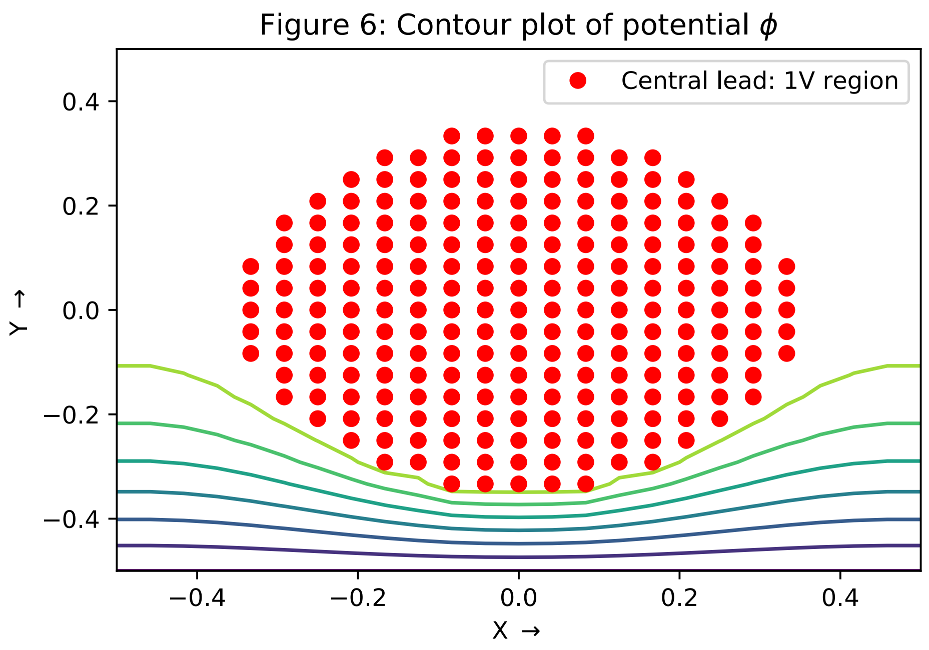
### 2.5.2 Code

```
# Plotting the 3-D Surface Potential
fig5=figure('Figure 5: 3-D plot of the potential') # open a new figure
ax=p3.Axes3D(fig5) # Axes3D is the means to do a surface plot
title('Figure 5: 3-D surface plot of Potential $\phi$')
surf = ax.plot_surface(Y, X, phi, rstride=1,cstride=1, cmap=cm.jet)
xlabel(r'$\leftarrow $X $\rightarrow$')
ylabel(r'$\leftarrow $Y $\rightarrow$')
ax.set_zlabel('$Z$')
show()

# Plotting the contour plot of potential phi
figure('Figure 6: Contour Plot of Potential $\phi$')
plot(x[ii[0]],y[ii[1]],'ro', label='Central lead: 1V region')
contour(Y,X,phi)
xlabel(r'X $\rightarrow$')
ylabel(r'Y $\rightarrow$')
title('Figure 6: Contour plot of potential $\phi$')
legend()
show()
```

### 2.5.3 Plots





### 2.5.4 Observations

- From the above **Surface plot**, we can conclude that after updating the potential, the potential gradient is **higher** in the lower region of the plate, since the lower side is grounded and the electrode is at 1 V, hence there is **high potential gradient** from electrode to the grounded plate.
- The upper region of the plate is approximately at **1V**, since it isn't acted upon by a forced Voltage. Hence, while updating the values, all the points are replaced with an **average** of the values of the neighbouring points.
- An almost same observation is seen using **Contour plot** in 2 dimensions. We observe the presence of gradients in the lower region of the plate and an almost negligible gradient in upper region of the plate.

## 2.6 Part E

### 2.6.1 Procedure

- Obtaining the **current density** by computing the potential gradient.
- The actual value of $\sigma$ doesn't affect the current profile, hence we set its value as 1.
- The current density equations are as follows -

$$J_x = -\frac{\partial \phi}{\partial x} \tag{17}$$

$$J_y = -\frac{\partial \phi}{\partial y} \tag{18}$$

- However, currently, we'll use the below formulas -

$$J_{x,ij} = \frac{1}{2}(\phi_{i,j-1} - \phi_{i,j+1}) \tag{19}$$

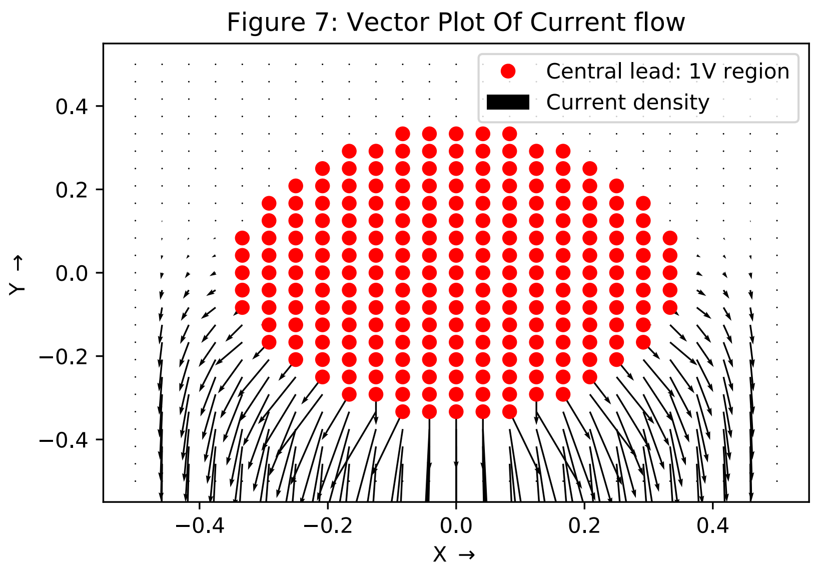
$$J_{y,ij} = \frac{1}{2}(\phi_{i-1,j} - \phi_{i+1,j}) \tag{20}$$

### 2.6.2 Code

```
# Initializing and defining the current densities
Jx = zeros((Ny,Nx))
Jy = zeros((Ny,Nx))
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])

# Plotting the vector plot of current flow
figure('Figure 7: Vector Plot of the Current Flow')
plot(x[ii[0]],y[ii[1]],'ro', label='Central lead: 1V region')
quiver(x,y,Jx,Jy, label='Current density')
xlabel(r'X $\rightarrow$')
ylabel(r'Y $\rightarrow$')
title('Figure 7: Vector Plot Of Current flow')
legend(loc='upper right')
show()
```

### 2.6.3 Plots



### 2.6.4 Observation

- As we noted before, the **potential gradient** was higher in lower region of the plate, and we know that **Electric field** is the gradient of the potential as given below -

$$\vec{E} = -\nabla\phi \tag{21}$$

- So $\vec{E}$ is **larger** when the potential gradient is high and is inverted since it is negative of the gradient. As a result, it is **higher** in the bottom region, which is closer to grounded bottom plate.
- Also, we know that

$$\vec{J} = \sigma\vec{E} \tag{22}$$

- $\vec{J}$ is **higher** and perpendicular to equipotential electrode region i.e the **Red dotted region**. Hence, the current is greater in bottom region of the plate and perpendicular to the electrode region since $I = \vec{J}.\vec{A}$
- As a result, most of the current flows from electrode to the bottom plate which is grounded because of higher potential gradient.
- There is almost no current in upper part of the plate since there is a negligible potential gradient as we observed from the **Surface plot** and **Contour plot** of the potential $\phi$.

---

## 3 Conclusion

- We inferred that most of the current is in the **narrow region** at the bottom which is the region getting **strongly heated**.
- Since there is almost no current in the upper region of plate, the bottom part of the plate gets **hotter** and temperature increases in the **bottom** region of the plate.
- And we know that heat generated is from $\vec{J}.\vec{E}$ (ohmic loss). Since $\vec{J}$ and $\vec{E}$ are higher in the bottom region of the plate, there will be higher **heat generation** and subsequent **temperature rise**.
- Hence, we studied the modelling of currents in a resistor. We also observed that the best method to solve the current problem is to **increase** $N_x$ and $N_y$ to very **high values** (100 or $\geq$ 100) or increasing the number of iterations, inorder to get accurate values i.e current in the resistor.
- However the disadvantage of this method is that, it is considerably **slow** even on using vectorized code because the **decrease** in error is very slow with respect to the number of iterations.