EE2703 Assignment 5: The Resistor Problem

Anvith Pabba EE19B070

25th March 2021

1 Introduction

In this assignment, we wish to find the currents and voltages produced in the given resistor. In order to obtain these values, we use suitable approximations. We then visualize the data in the form of various plots.

2 The Resistor Problem

2.1 Given info

In the given question, 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 grounded, while the remaining are kept floating. The plate is Ny cm x Nx cm in dimensions.

2.2 The Given Figure

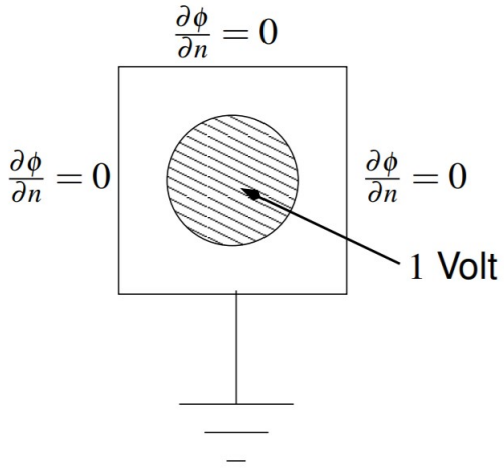

Figure 1: Visual representation of the problem

3 Theory

3.1 Solving the equations

The conductivity is given by:

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

Now the Electric field is the gradient of the potential:

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

and continuity of charges yields:

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

Combining these two equations and assuming the conductivity of the medium is constant and is σ gives us:

$$\nabla^2 \phi = \frac{1}{\sigma}\frac{\partial \rho}{\partial t} \tag{4}$$

For DC currents, the RHS is zero, this gives us:

$$\nabla^2 \phi = 0 \tag{5}$$

3.2 Approximations to solve in 2-D

Laplace's equation is easily transformed into a difference equation. The equation can be written out in Cartesian coordinates:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0 \tag{6}$$

assuming ϕ is available at points (x_j, y_j), we can write:

$$\left.\frac{\partial \phi}{\partial x}\right|_{(x_j, y_j)} = \frac{\phi(x_{j+1/2}, y_j) - \phi(x_{j-1/2}, y_j)}{\Delta x} \tag{7}$$

and

$$\left.\frac{\partial^2 \phi}{\partial x^2}\right|_{(x_j, y_j)} = \frac{\phi(x_{j+1}, y_j) - 2\phi(x_j, y_j) + \phi(x_{j-1}, y_j)}{(\Delta x)^2} \tag{8}$$

Combining this with the corresponding equations for the y-derivatives, we finally get:

$$\phi_{i,j} = \frac{\phi(x_{j+1}, y_j) + \phi(x_{j-1}, y_j) + \phi(x_j, y_{j+1}) + \phi(x_j, y_{j-1})}{4} \tag{9}$$

Thus, if the above solution holds, the potential at any point should be the average of its neighbours.

3.3 Boundary Conditions

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 ϕ should be tangential. This is implemented by requiring that ϕ should not vary in the normal direction.

Writing The Code

1 Getting the parameters

1.1 Getting the initial parameters

We have 4 initial parameters, **Nx, Ny, Radius, Niter**. We get these 4 parameters from the user. We get these parameters through the user:

```
python EE19B070_EE2703_Assignment5.py 25 25 8 1500
```

Figure 2: How the code should be executed

After extracting the inputs through *import sys*, we check if everything is correct or if there are any mistakes from the users side. We check the following:

1. All 4 parameters **must be given**
2. *Nx, Ny, Radius, Niter* should **all be integers**

We check whether the conditions above are satisfied, if they are not then an error is given to the user. The relevant code:

```
import sys

#Checking if correct number of arguments are given
if len(sys.argv)!=5:
  print("Please enter ALL 4 parameters!!")

#Checking if all the inputs are intgers
while True:
  try:
    Nx = int(sys.argv[1])
  except ValueError:
    print('argument 1 must be an integer')
    break
  else:
    Nx = int(sys.argv[1])
    break

\textbf{Similarly we do this for Nx,Ny,Radius and Niter}
```

1.2 Initialising the phi (potential) and other arrays

phi is a matrix of size Nx,Ny. It is initialed with initial value of 0. **x** is the shifted x axis, such that the middle value is 0 **y** is the shifted y axis, such that the middle value is 0 X,Y is the corresponding meshgrid we also define **n** and initialise **errors**

```
x = np.linspace(-((Nx-1)/2), ((Nx-1)/2), Nx)
y = np.linspace(-((Ny-1)/2), ((Ny-1)/2), Ny)

#initialising phi
phi = np.zeros([Ny,Nx])
X,Y=meshgrid(x,y)
n = arange(Niter)
errors = np.zeros([Niter,1])
```

1.3 finding and initialising points with V = 1.0

we use the **where** command to find points that satisfy the condition of $X * X + Y * Y <= (radius * radius)$, we store these points in an array *ii*. We then make every point in this have a potential of V = 1 (given in the problem statement).

```
ii = where((X*X + Y*Y) <= r*r)
phi[ii] = 1.0
```

1.4 Plot of phi

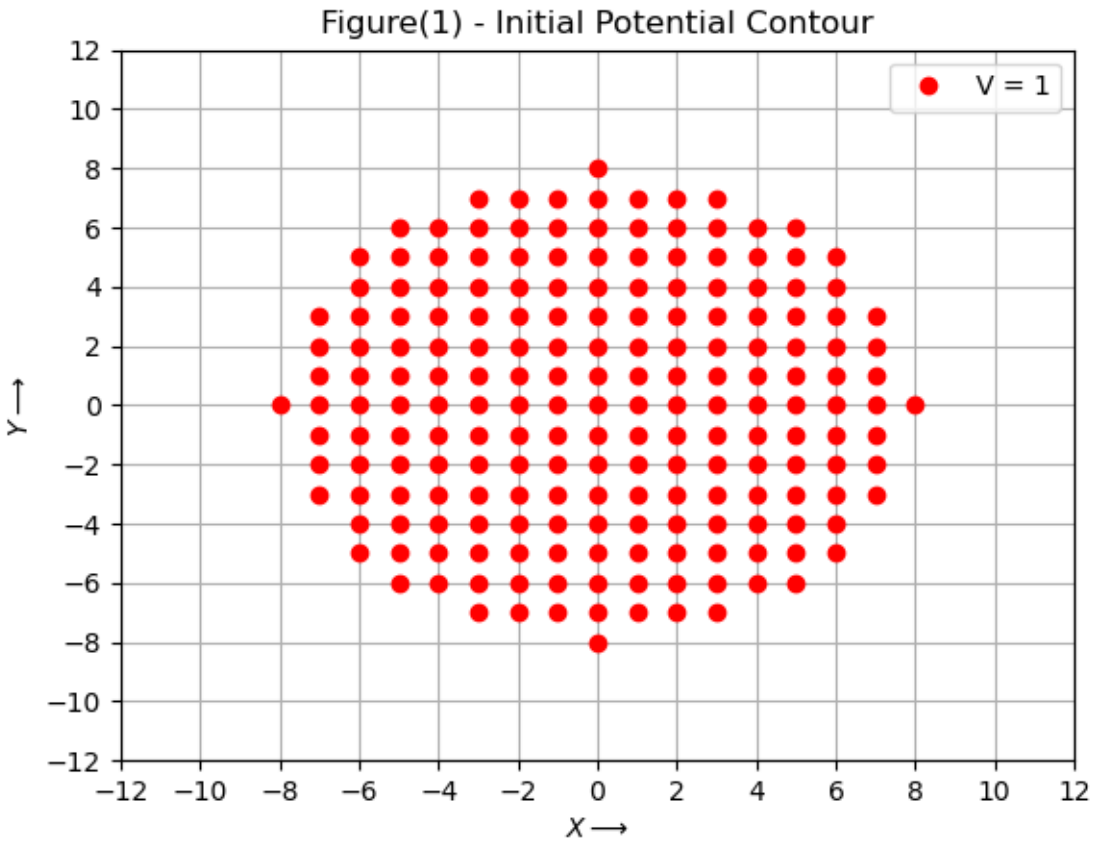

Figure 3: Initial plot of phi

2 Performing the iterations

2.1 The iteration

Using the theory given above, we know that the potential of each point is the average of the values around it. To get the required values of phi, we perform an iteration that comprises of 3 main parts:

- for k in range(Niter)
    1. save copy of phi
    2. update phi array
    3. assert boundaries
- errors[k]=(abs(phi-oldphi)).max()

2.2 Save copy of phi

To do this, we use the **.copy()** command

```
oldphi=phi.copy()
```

2.3 updating the potential

In order to perform the averaging function on every point in the phi array, we would have used 2 nested for loops. In reality, for loops are not as optimised in python as compared to vector loops, this means the time taken by the nested for loops is extremely high. Infact, for every nested for loop thats added, the time taken by the program increases exponentially. This is not at all favourable.

So instead of using for loops, we use vectorized code that performs the same functions. This vectorized code significantly reduces the time taken by the program.

so a conventional code:

```
for(i=1 ; i<Ny-1 ; i++ ){ // interior nodes
    for(j=1 ; j<Nx-1 ; j++ ){ // interior nodes
        phinew[i,j]=0.25*(phi[i,j-1]+phi[i,j+1]+phi[i-1,j]+phi[i+1,j]);
        }
}
```

Is now converted to vectorized form, as:

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

2.4 Boundary conditions

The theory for this is already given above, in the first Section 3.3.

At the **left boundary**, we need to set the normal derivative of potential to zero. This means that

$$\frac{\partial \phi}{\partial x} = 0. \tag{10}$$

We similarly do this for the **right side** and the **top side**. Also, the voltage of the bottom is constant at 0, but this is automatically taken care of.

The voltages of the points in the wire might get overwritten, so we have to keep updating them after each iteration. We do this by using phi[ii] = 1.0.

2.5 Code

All the above in vectorized code is:

```
    #assert boundaries
  phi[1:-1,0] = phi[1:-1,1] #left side
  phi[1:-1,-1] = phi[1:-1,-2] #right side
  phi[0,1:-1] = phi[1,1:-1] #top side
  #bottom is fixed with a value of 0

  phi[ii] = 1.0
```

3 Obtaining and Plotting the Errors

Plotting the Errors in semilogy and loglog form

```
title("Figure(2) - Error plot 1")
semilogy(n,errors,label="semilogy plot of error vs iteration")
xlabel('$n\longrightarrow$')
ylabel('$error,log[n]\longrightarrow$')
grid(True)
legend()
```

```
show()

title("Figure(3) - Error plot 2")
loglog(n,errors,label="loglog plot of error vs iteration")
xlabel('$n,log\longrightarrow$')
ylabel('$error[n],log\longrightarrow$')
grid(True)
legend()
show()
```

3.1 Semilogy Plot

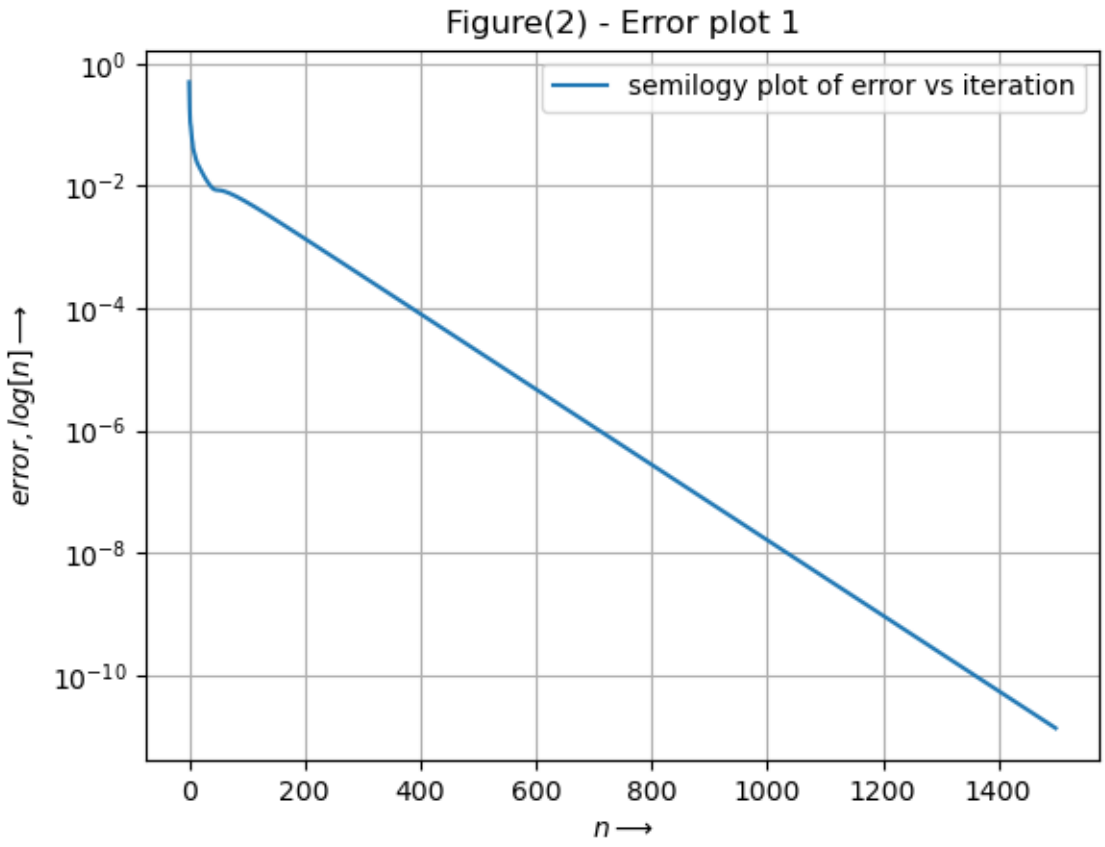

Figure 4: Semilogy plot of the error

3.2 Loglog Plot

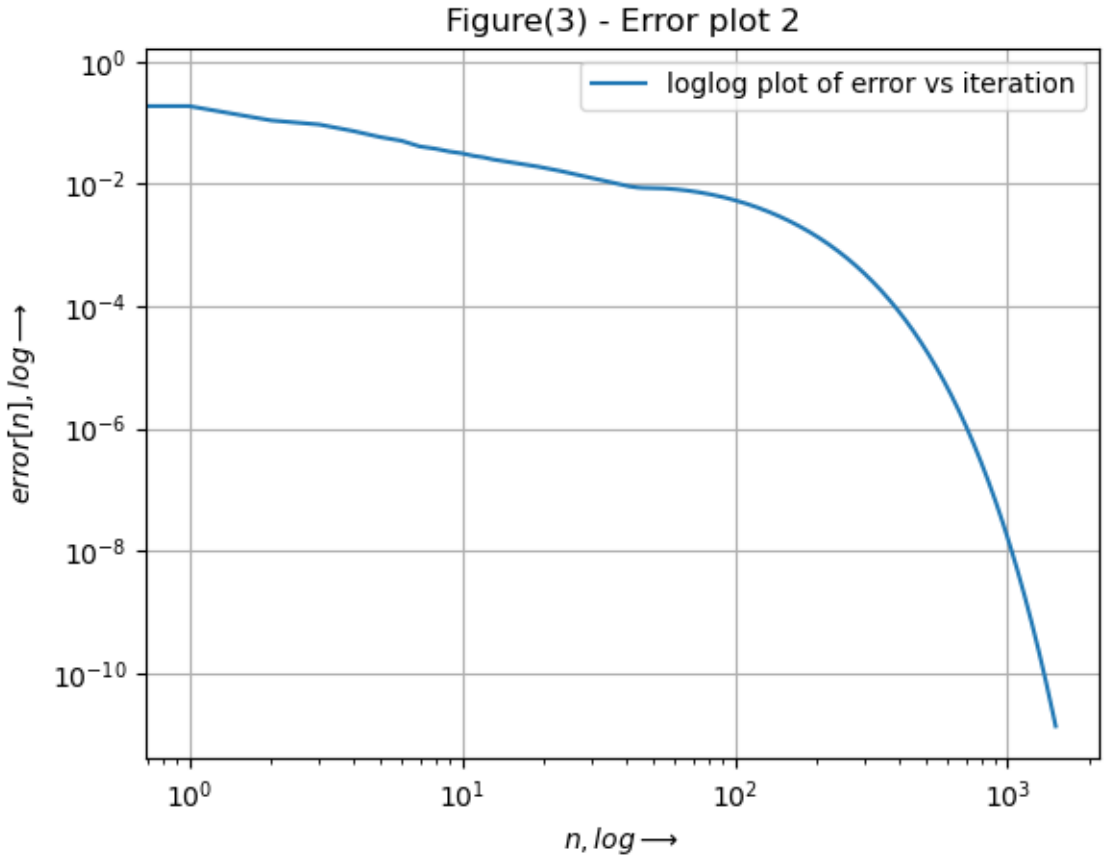

Figure 5: loglog plot of the error

3.3 Semilogy Plot for errors after the 500th iteration

As we can see from the first semilogy plot, there is a small deviation for the iterations less than 500. But for iterations greater than 500, the plot is a straight line. In order to test this, we plot the semilogy plot for iterations greater than 500.

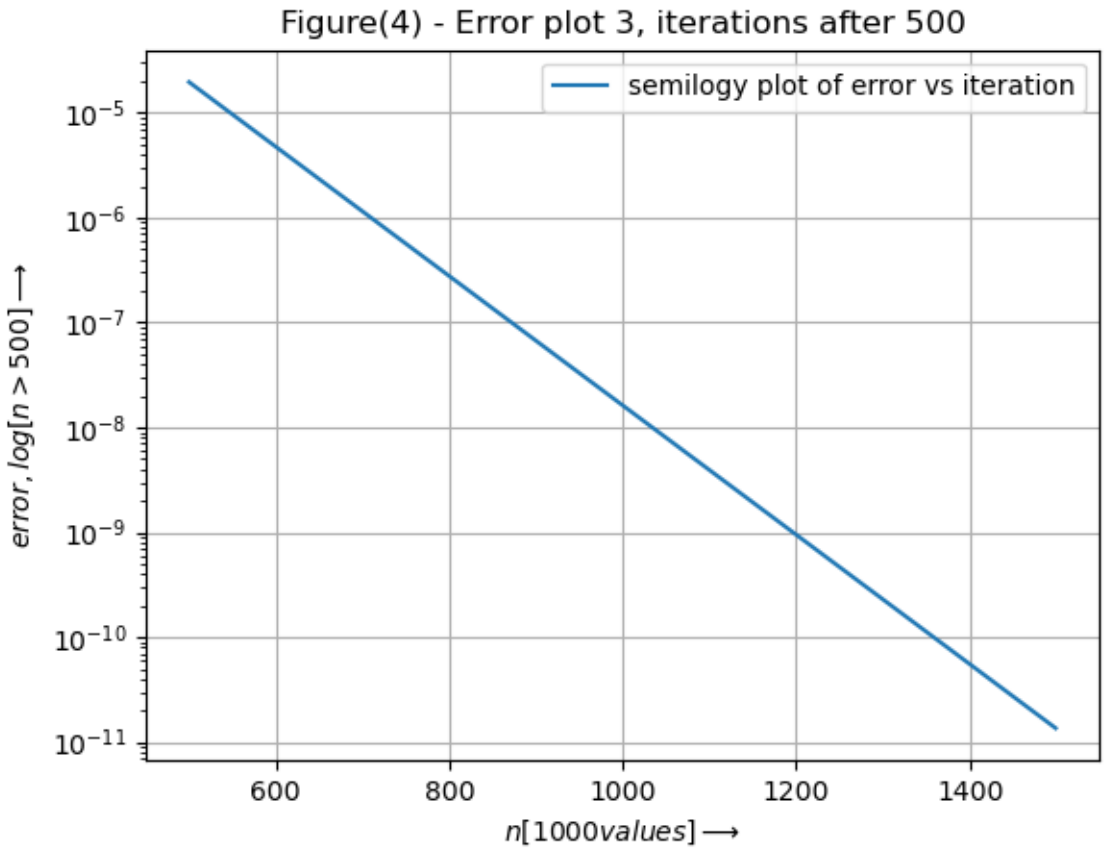

Figure 6: Semilogy plot of the error, for iterations >500

3.4 Finding the Best fit for the errors

As we can see form the graphs, the semilogy plot is a straight line this means that the error is of the form of:

$$errors_k = Ae^{Bk} \tag{11}$$

To obtain the best fit for A and B, we use lstsq on the linear equation obtained by taking the log of both sides. i,e:

$$log(error_k) = log(A) + Bk \tag{12}$$

We can see that taking the error as Ae^{Bk} is not very correct for iterations less than 500. So, we find A and b for 2 different conditions. 1) for all iterations 2) only for iterations ¿500

3.5 Code

```
fit1 = zeros([1000,1])
fit2 = zeros([1500,1])
#using lstsq
#lstsq for after 500
c1 = np.linalg.lstsq(c_[ones([1000,1]),n[500:1500]], log(errors[500:1500]),
                                                rcond = None)[0]
logA1 = c1[0]
A1 = exp(logA1)
B1 = c1[1]

#lstsq for the entire thing
```

```
c2 = np.linalg.lstsq(c_[ones([1500,1]),n], log(errors),rcond = None)[0]
logA2 = c2[0]
A2 = exp(logA2)
B2 = c2[1]
```

3.6 Values obtained from lstsq method

fitted values of A1 and B1 for iterations > 500 are :

- A1 = [0.02378819] and
- B1 = [-0.01419784]

fitted values of A2 and B2 for all iterations are :

- A2 = [0.02370508] and
- B2 = [-0.0141953]

We can see that both obtained values of A and B are very similar.

3.7 Semilogy Plot of obtained and actual errors

We can see that all 3 semilogy plots are all the same.

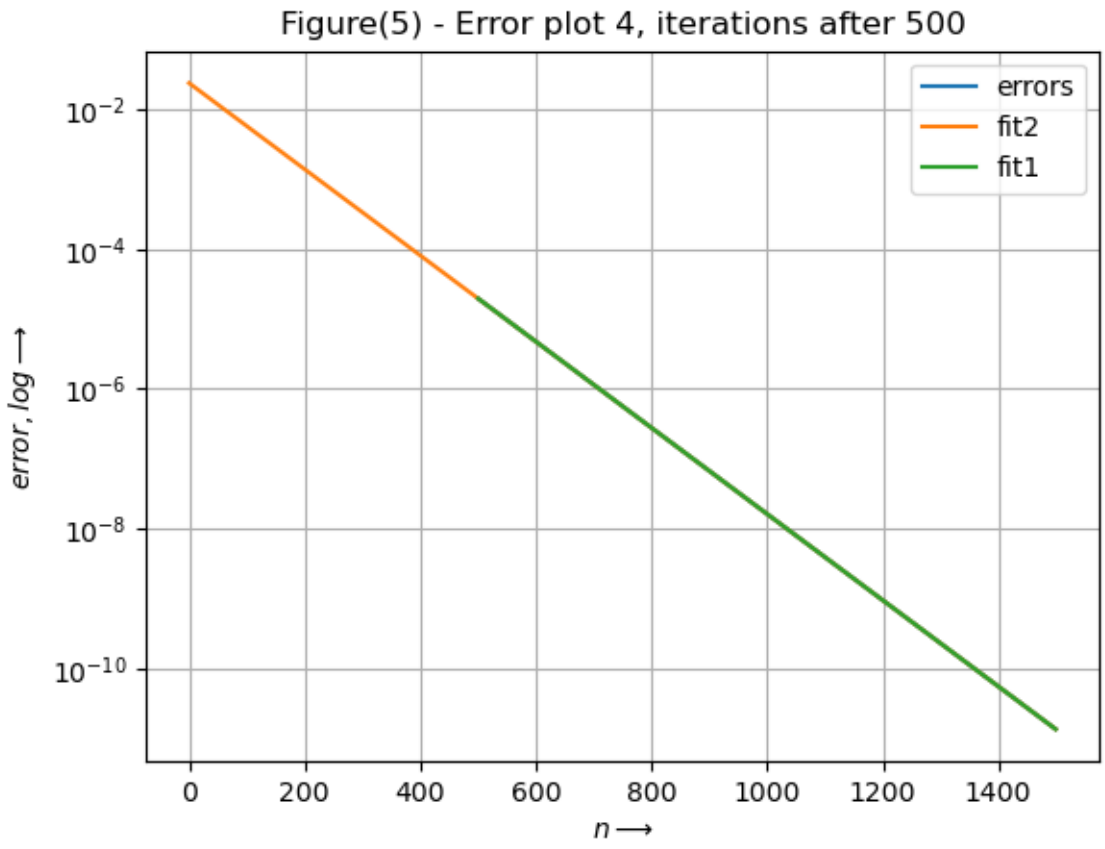

Figure 7: Semilogy plot of obtained and actual errors

3.8 Loglog Plot of obtained and actual errors

We can see that all 3 semilogy plots are all similar, but the actual error deviates for iterations < 500.

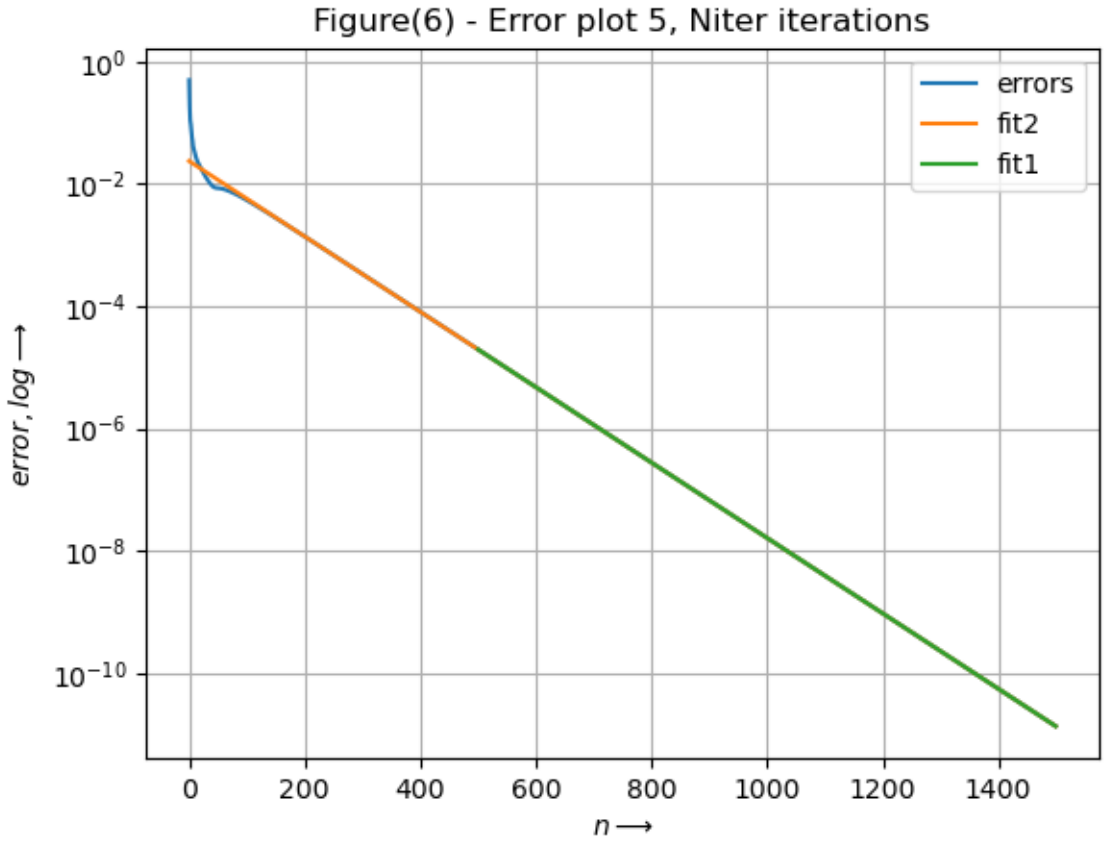

Figure 8: Loglog plot of obtained and actual errors

3.9 Stopping Error

The approximate Error obtained is:

$$Error \approx -\frac{A}{B}e^{B(N+0.5)} \tag{13}$$

Error we obtain is:

$$Error = \frac{0.02370508}{-0.0141953}e^{-0.0141953*1500} \tag{14}$$

this is equal to: 1.6699*5.67*1e-10
which is **9.4466*1e-10**

4 Contour plot of potential

This plots the voltages as contours with colorbar variations.

4.1 Code

```
title("Figure(9) - The contour plot")
contourf(x, -y, phi)
colorbar()
plot(ii[0] - (Nx-1)/2, ii[1] - (Ny-1)/2,linestyle="None", color='red', marker='
                                        o',label="V = 1")
xlabel("$x\longrightarrow$")
ylabel("$y\longrightarrow$")
show()
```

4.2 Plot

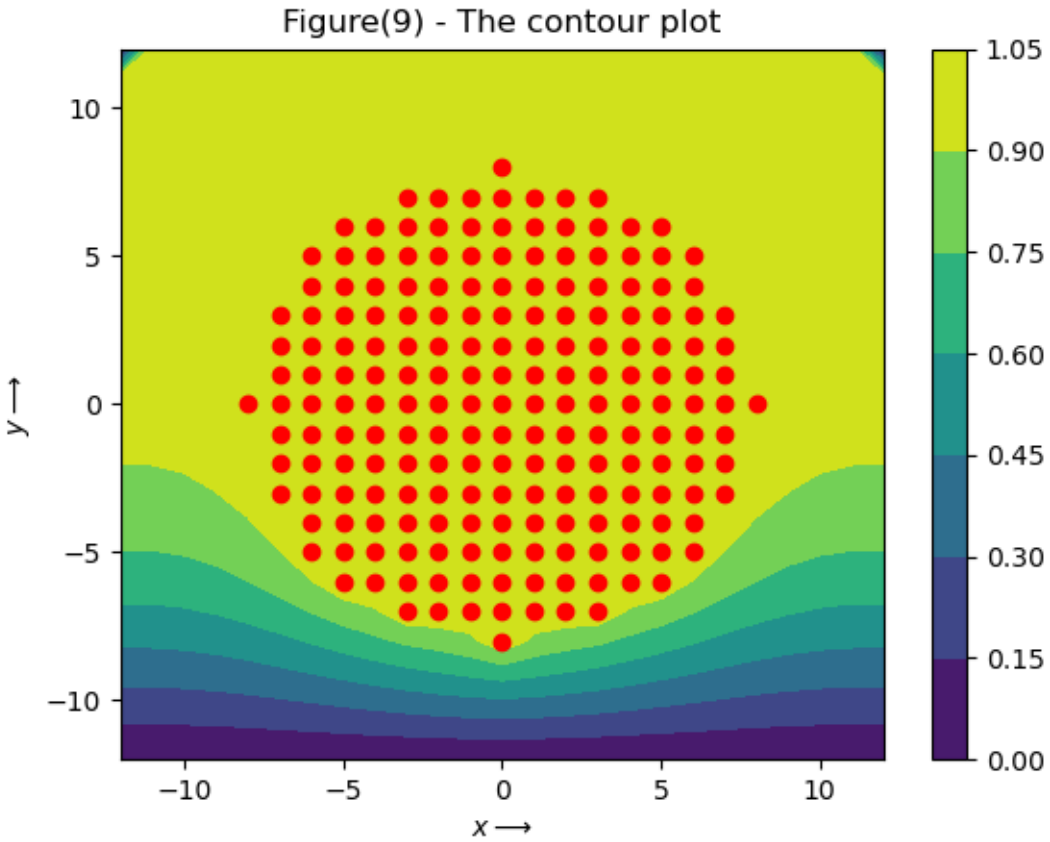

Figure 9: Contour plot

5 3-D Surface plot

we use the *mpltoolkits.mplot3d.axes3d* with alias *p3* command to plot.

5.1 Code

```
title("Figure(8) - The 3-D surface plot of the potential")
fig1=figure(8) # open a new figure
ax=p3.Axes3D(fig1) # Axes3D is the means to do a surface plot
surf = ax.plot_surface(Y, X, phi.T, rstride=1, cstride=1, cmap=cm.jet)
ylabel('$Ground$')
xlabel('$y\longrightarrow$')
show()
```

5.2 Plot, view 1

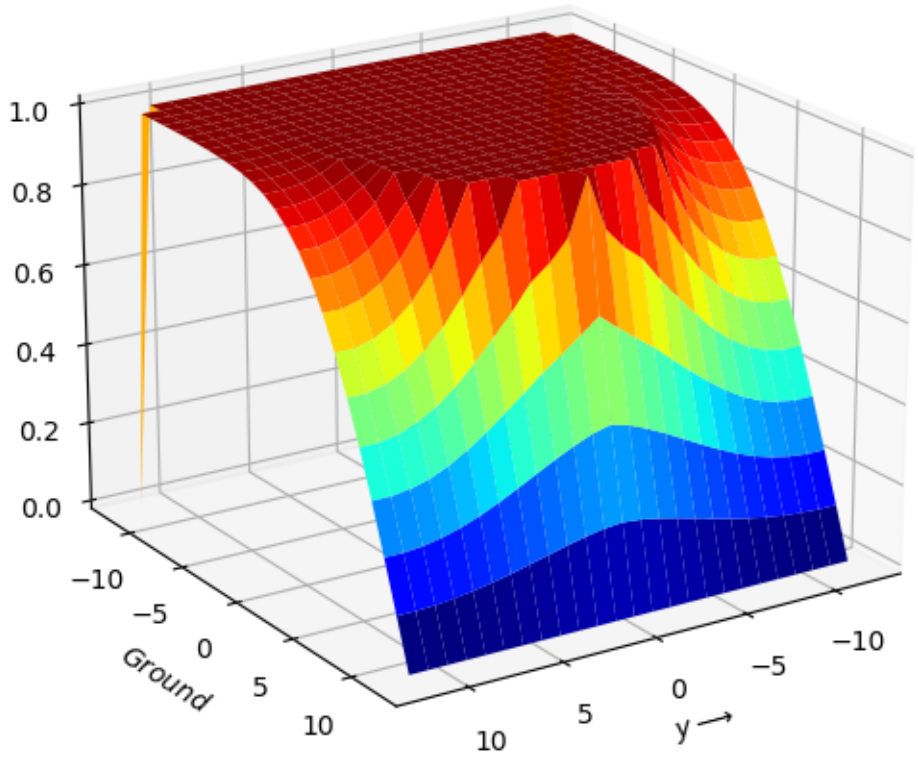

Figure 10: surface plot, view 1

5.3 Plot, view 2

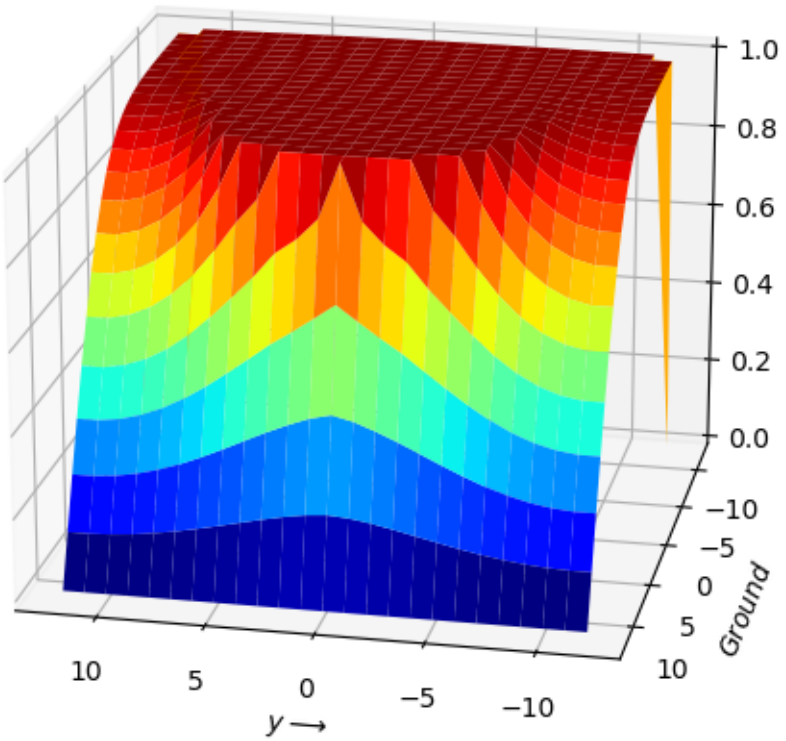

Figure 11: surface plot, view 2

6 Quiver Plot of The Currents

Now to obtain the currents, we need to compute the gradient. Our equations are:

$$j_x = -\frac{\partial \phi}{\partial x} \tag{15}$$

$$j_y = -\frac{\partial \phi}{\partial y} \tag{16}$$

The actual value of σ does not matter to the shape of the current profile, so we set it to unity.
This numerically translates to:

$$J_{x,i,j} = \frac{1}{2}(\phi_{i,j-1} - \phi_{i,j+1}) \tag{17}$$

$$J_{y,i,j} = \frac{1}{2}(\phi_{i-1,j} - \phi_{i+1,j}) \tag{18}$$

6.1 Code for Currents

This is the vectorized form of the code

```
Jx = np.zeros([Ny,Nx])
Jy = np.zeros([Ny,Nx])

#Jx,Jy for the quiver

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])
```

6.2 Code to plot the quiver plot

```
title("Figure(7) - Current quiver plot")
quiver(x,y,Jx[::-1,:],Jy[::-1,:],scale = 5,label = "current")
plot(ii[0] - (Nx-1)/2, ii[1] - (Ny-1)/2,linestyle="None", color='red', marker='
                                        o',label="V = 1")
xlabel('$x\longrightarrow$')
ylabel('$y\longrightarrow$')
grid(True)
legend(loc = 'upper right')
show()
```

6.3 Quiver plot

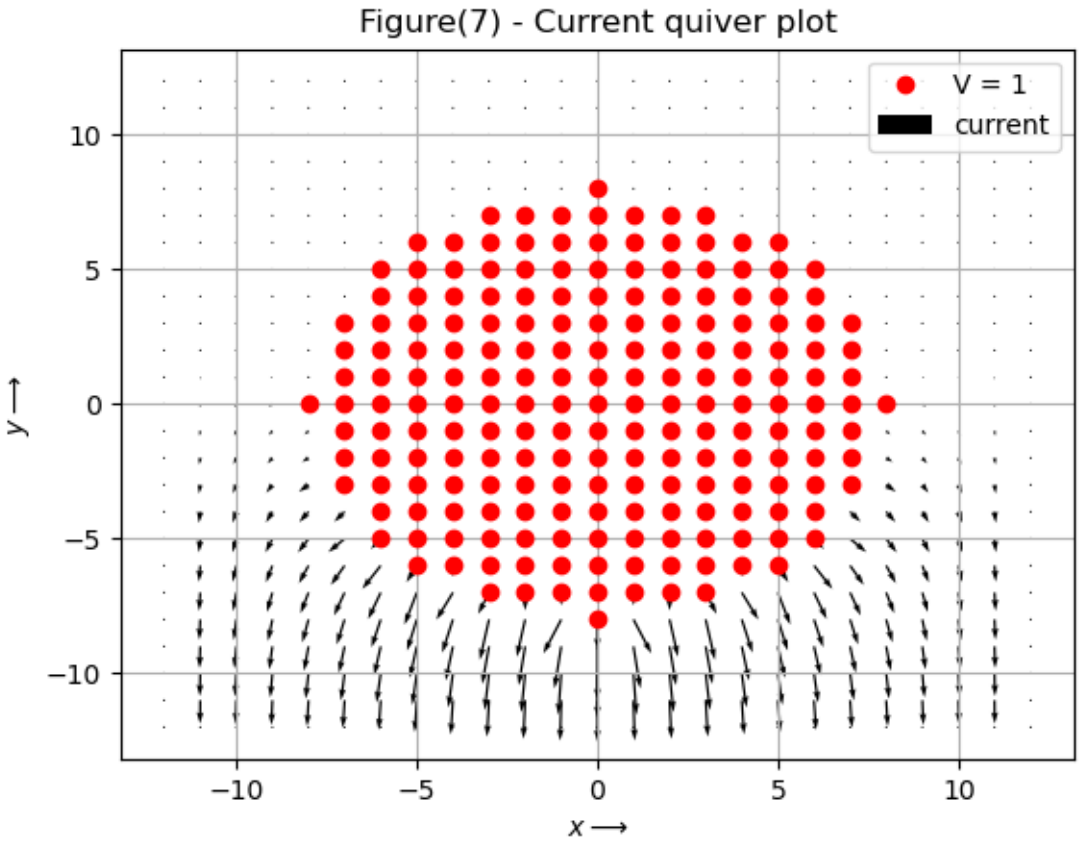

Figure 12: Quiver plot of the currents

7 Observations

1. We can clearly see that most current flows in the bottom of the plate, from the wire to the grounded side. And that there is almost no current in the rest of the plate.

2. we know from Ohms law,
$$\vec{j} = \sigma \vec{E} \tag{19}$$
and this tells us that the electric field is much higher in the lower part pf the plate as compared to the rest of the plate.

3. As the current as well as the electric field is higher in these regions, the ohmic loss is very high and centralized. Hence the temperature in these areas is also going to be extremely high. When modelling such a device, we need to be very careful it can handle the high temperatures.

4. in order to improve accuracy, we can increase Nx and Ny, but this will also increase the time the program takes. This is only required if we require higher accuracy over time.

8 Conclusions

1. In conclusion, we see that most current flows in the bottom of the plate, this will strongly heat it up and raise its temperature.

2. We learnt how to plot data in the form of contours and 3d surface plots.

3. we learnt how to plot the quiver plots, and how to find the current through vectorized code.

4. We saw how to improve the run time of our program by using vectorized code instead of un-optimized *for* loops.

5. we saw that the error is very close to the exponential form of Ae^{Bk} and that the values obtained by lstsq method are also extremely similar.