EE2703 : Applied Programming Lab
Assignment 3
Fitting Data to Models

Debojyoti Mazumdar
EE20B030

March 7, 2022

0.1 Abstract

Aim of the assignment: To understand how the flow of current in a conductor changes the potential around it and also how hot the conductor gets. We do so by:

- finding phi(Potential)
- update it using the poison differential condition and the boundary conditions
- finding the current density
- finding the temperature
- Plotting the necessary graphs

0.2 Introduction

In the code we implement the following tasks: We first generate the potential array and put 1 as the potential in the places where the electrode is present.Then we try to get the potential in other parts(except at the boundaries) using the poison differential equation. Then we get the potential at the boundaries using the boundary conditions. Then we try to find out the current density using ohm's law. Then we try to find out the temperature of the conductor using the current density given.

0.3 Assignment

0.3.1 Part 1

We import the standard libraries and initialize phi as an array of dimension (Nx,Ny).

```
phi = np.zeros((Nx, Ny), dtype=float)
```

Then we find out the points where the electrode is touching by doing the following

```
x, y = np.linspace(-0.5, 0.5, num=Nx,
                   dtype=float), np.linspace(-0.5, 0.5, num=Ny, dtype=float)
Y, X = np.meshgrid(y, x, sparse=False)
ii = np.where(X**2+Y**2 < (0.35)**2)
```

Now we assign 1 as the potential to all these points.

```
phi[ii] = 1.0
```

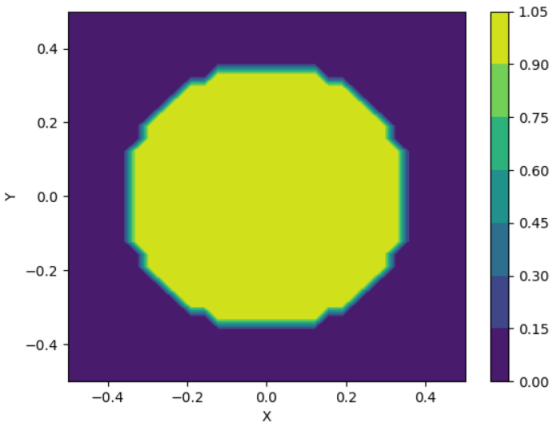

Figure 1: potential at different points

0.3.2 Part 2

Now we start performing the iterations. But before that we make an updating function for phi.The updating function is below:

```
def update_phi(phi):
    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])
    return phi
```

Then we make a function that puts potential values in the boundaries. The function is given below:

```
def assert_boundaries(phi):
    phi[1:-1, 0] = phi[1:-1, 1]  # left side normal component 0
    phi[1:-1, -1] = phi[1:-1, -2]  # right side normal component 0
    phi[0, 1:-1] = phi[1, 1:-1]  # top side normal component 0
    return phi
```

Now we iterate this in a loop running Niter times and calculate the error for each iteration.

```
errors = np.zeros(Niter)
for k in range(Niter):
    oldphi = phi.copy()
    phi = update_phi(phi)
    phi = assert_boundaries(phi)
    phi[ii] = 1.0  # To make the potential at the points of wire 1
    errors[k] = (abs(phi-oldphi)).max()
```

Then we plot the error function with the iteration and get the following graph:

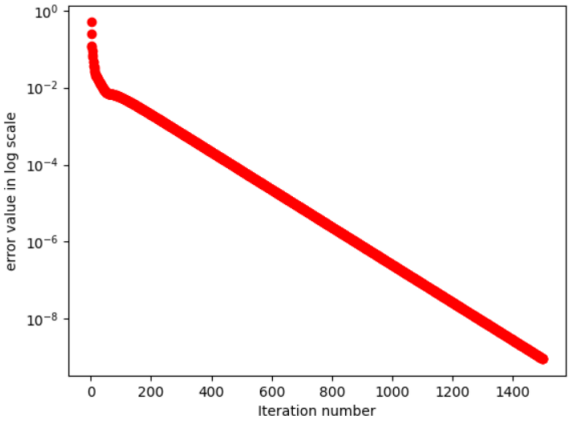

Figure 2: error(log) vs iteration number

0.3.3 Part 3

Now we try to fit a line through the log plot. For fitting we do:

```
def populate_M():
    x = np.ones(Niter).T
    y = np.arange(1, Niter+1)
    y = y.T
    return c_[x, y]

M = populate_M()
logA, B = lstsq(M, np.log(errors))[0]
A = math.e**logA
fitting_error = A*np.exp(B*x)
```

Now we plot this fitting in the graph above, we get:

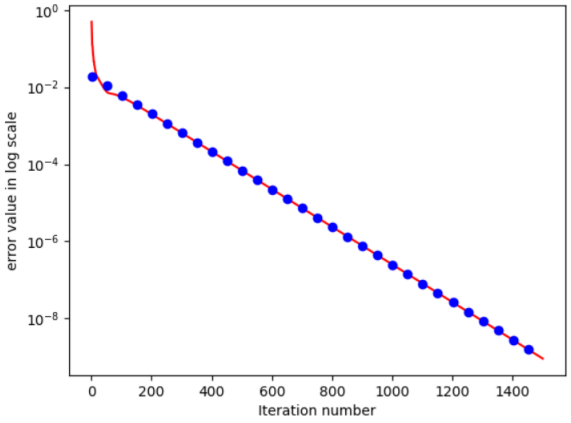

Figure 3: error vs iteration number

Now we draw the surface plot of the potential, we get:

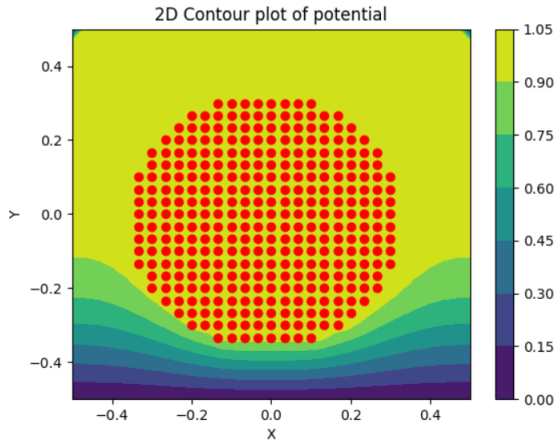

Figure 4: plot of potential in 2-d

And the 3-d plot of potential is:

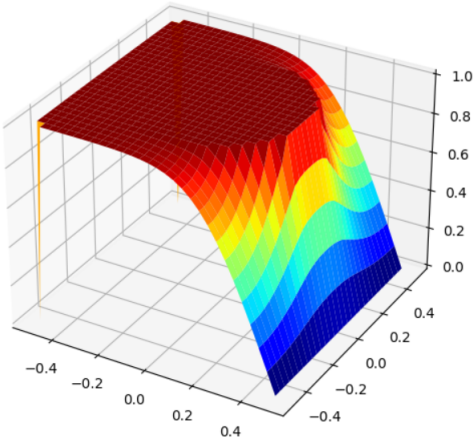

Figure 5: plot of potential in 3-d

0.3.4 Part 4

We now get the current density in the resistor. To do so we use the ohm's law and code it. So we have:

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

Now we plot the current density in the 2-d plot by the following code:

```
plt.title("Vector plot of current flow")
plt.quiver(Y[1:-1, 1:-1], -X[1:-1, 1:-1], -Jx[:, ::-1], -Jy)
x_c, y_c = np.where(X**2+Y**2 < (0.35)**2)
plt.plot((x_c-Nx/2)/Nx, (y_c-Ny/2)/Ny, 'ro')
plt.show()
```

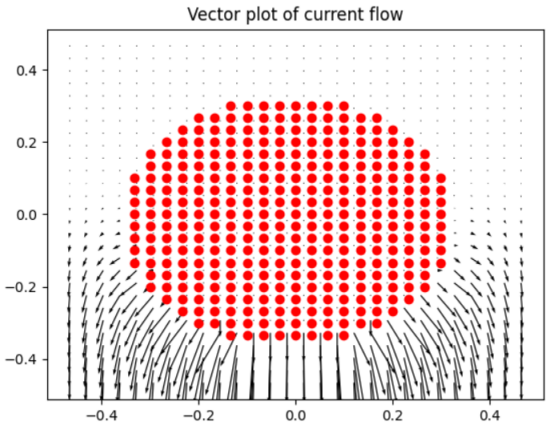

Figure 6: current density

Here, red dots indicate the position of the electrode.

0.3.5 Part 5

Now we calculate the heat generated by the following current density.To do so we first initialize the temperature at every point as 300K.

```
temp = 300 * np.ones((Nx, Ny), dtype=float)
```

Then we make a function for it to solve the laplace equations.

```
def tempdef(temp, oldtemp, Jx, Jy):
    temp[1:-1, 1:-1] = 0.25*(tempold[1:-1, 0:-2] + tempold[1:-1, 2:] +
                             tempold[0:-2, 1:-1] + tempold[2:, 1:-1]+(Jx)**2 + (Jy)**2)
    return temp
```

Then we make a function to make the temperatures follow the boundary conditions.

```
def temper(phi, mask=np.where(X**2+Y**2 < (0.35)**2)):
    phi[:, 0] = phi[:, 1]  # Left Boundary
    phi[:, Nx-1] = phi[:, Nx-2]  # Right Boundary
    phi[0, :] = phi[1, :]  # Top Boundary
    phi[Ny-1, :] = 300.0
    phi[mask] = 300.0
    return phi
```

Then we iterate it for Niter number of times to get the final value of temperature at each point.

```
for k in range(Niter):
    tempold = temp.copy()
    temp = tempdef(temp, tempold, Jx, Jy)
    temp = temper(temp)
```

Then we plot the temperature 2-d contour plot by the following code.

```
plt.title("2D Contour plot of temperature")
plt.xlabel("X")
plt.ylabel("Y")
plt.contourf(Y, X[::-1], temp)
plt.colorbar()
plt.show()
```

And we get:

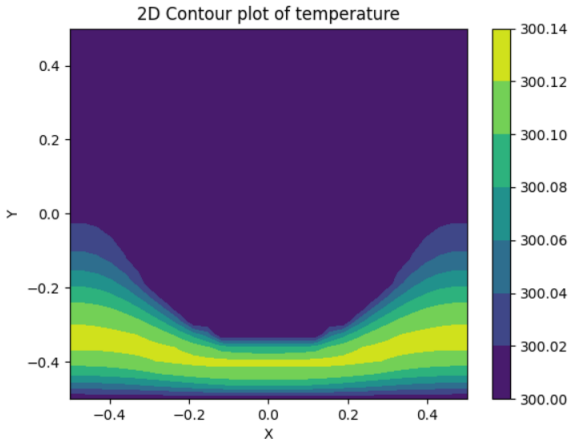

Figure 7: 2-d temperature contour plot

0.4 Conclusions

- We see that the current flows completely from below and none from top.
- Maximum temperature is in the narrow region lying below the electrode and above the bottom ground.