EE2703: Applied Programming Lab

Week 5: Laplace Equation

Author: Surya Prasad S, EE19B121

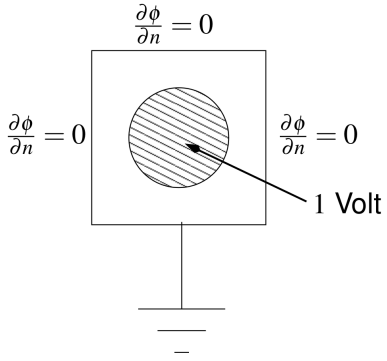
March 24, 2021

1 Abstract:

In this week's assignment, we shall be finding out the flow of currents in a resistor in a conductor and also visualise how much heating takes place in the conductor. For finding the flow of currents we shall solve the Laplace equation numerically and plot the 2D and 3D plots of potential distribution and current flow. The wire portion of the conductor is kept at 1 V and one side of the conductor is kept grounded while others remain floating. We shall also plot the Temperature distribution.

2 Theory:

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 floating. The plate is 1cm by 1cm in size.



By conductivity

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

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 get

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

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

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

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

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

This is called Laplace's Equation and it can be easily transformed into a 2D differential equation. The equation in Cartesian coordinates is

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

Solving this numerically we get:

$$\phi_{i,j} = \frac{\phi(x_{i+1},y_j) - 2\phi(x_i,y_j) + \phi(x_{i-1},y_j)}{4}$$

If the solution holds then the potential at any point is the average of its neighbours. At the boundaries, where there is no electrode, the current should be tangential. 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.
The equations for the current densities are:

$$\vec{j_x} = -\partial\phi/\partial x$$

$$\vec{j_y} = -\partial\phi/\partial y$$

Numerically,

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

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

The current flows from higher potential to lower potential so we expect most of the current to flow from the wire portion to the lower boundary. Hardly any current will flow out of the top part of the wire. The lower region will also get strongly heated. To find the temperature we use the following equation:

$$\nabla.(\kappa\nabla T) = q = \frac{1}{\sigma}|J|^2$$

where the right side represents heat generated from ohmic loss. The boundary condition her is that T=300 at the wire and the ground while $\partial T/\partial n = 0$ at the other edges.

3 Plotting and Code Analysis:

3.1 Libraries imported:

The following libraries are imported:

```
import numpy as np
import os, sys
import scipy
import scipy.linalg as sp
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as p3
```

3.2 Setting the parameters:

The following are the default values of the parameters:

```
# Default values are assigned to parameters
Nx = 25                 # Number of x coordinates
Ny = 25                 # Number of y coordinates
radius = 8              # Radius of wire portion
Niter = 1500            # Number of iterations
```

For this report, all the computations are done with respect to the default values. These values can be changed in the given order by passing arguments in the commandline. Error checking is also done here to prevent incorrect values from being used.
With these values we obtain the x coordinates and y coordinates. We then convert radius into cm based on the number of x coordinates and y coordinates.

3.3 Contour Plot of Initial Potentials:

We initialise all the potentials as 0 and then make the potential of the wire portion to be 1 V. This refers to all the points which satisfy the condition:

$$X * X + Y * Y \leq (radius)^2$$

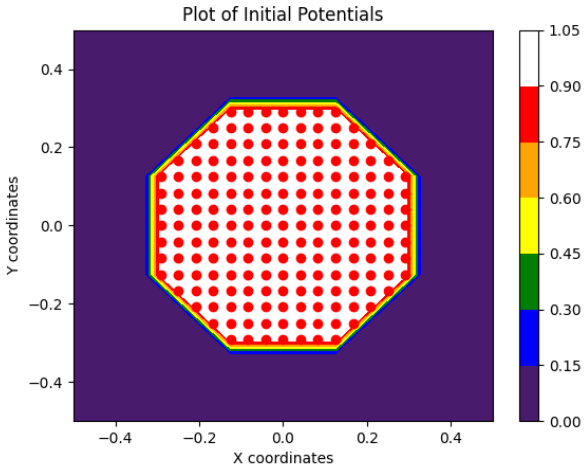


Figure 1: Contour Plot of Potential

3.4 Updating Potentials:

We shall iterate *Niter* times to get the final potential. During the iteration, we will use a second potential array, called *oldphi* to hold the values of the previous iteration. This is to keep track of how much the array changed during the current iteration. The difference between the two arrays is stored as a vector *errors*.

There are constraints while updating the potential. At the boundaries where the electrode is present, the potential will be equal to the potential of the electrode itself. At the other

boundaries, the gradient of ϕ should be tangential. This is implemented by making ϕ not vary in the normal direction. Code snippet for updating the potential:

```
errors = np.zeros(Niter)

for k in range(Niter):
    oldphi = phi.copy()

    phi[1:-1, 1:-1] = 0.25 * (oldphi[1:-1, 0:-2] + oldphi[1:-1, 2:] + oldphi
        [0:-2, 1:-1] + oldphi[2:, 1:-1])

    phi[:, 0] = phi[:, 1]
    phi[:, Nx-1] = phi[:, Nx-2]
    phi[0, :] = phi[1, :]
    phi[Ny-1, :] = 0
    phi[np.where(X**2 + Y**2 < radius**2)] = 1.0

    errors[k] = np.max(np.abs(phi - oldphi))
    if(errors[k] == 0):
        break
```

3.5 Fitting of errors:

Let's plot the errors in semilog scale and loglog scale.

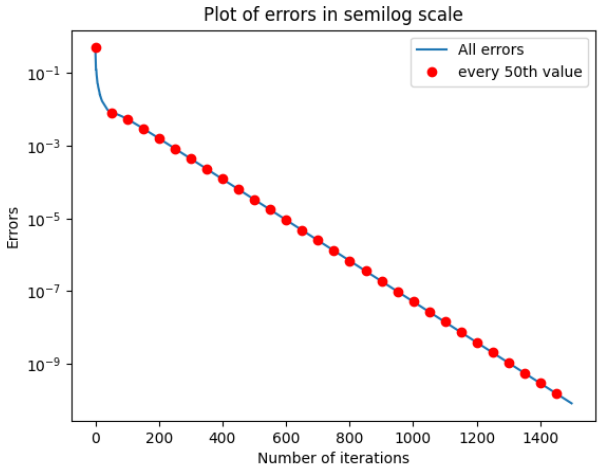


Figure 2: Errors in semilog scale

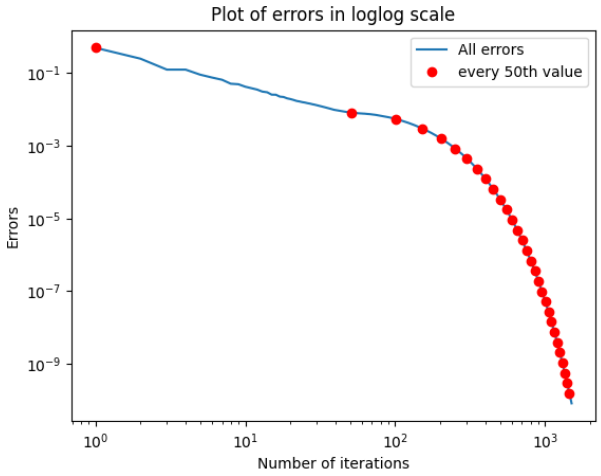


Figure 3: Errors in loglog scale

As we can see from the plots, the errors come out to be in almost a straight line in semi-log plot. So let's try to fit the errors in the form e^{A+B*x}.

$$y = e^{A+B*x}$$

$$logy = A + B * x$$

For fitting we need to fit the best values of A and B such that the error in determining y is minimum. For fitting we can use Least Squares Method to estimate A and B.

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \cdot \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} logy_1 \\ logy_2 \\ \vdots \\ logy_n \end{pmatrix}$$

The error in estimating y can be expected to reduce with increasing x. So we shall fit for the whole vector as well as after 500 iterations. Code snippet for fitting as an exponential:

```
def Log_Error_Fit(errors, start = 0):
        n_iter = errors.shape[0]
        log_errors = np.log(errors)
        return sp.lstsq(np.c_[np.ones((1, n_iter)).reshape(n_iter, 1), np.array(range(start

A1, B1 = Log_Error_Fit(errors)
A2, B2 = Log_Error_Fit(errors[500:], 500)
```

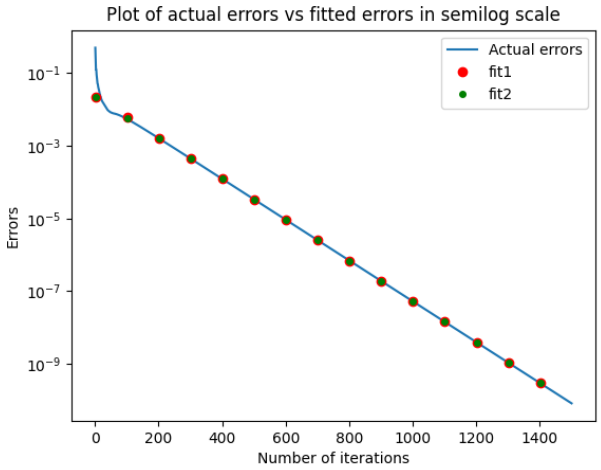


Figure 4: Best fit for error in semilog scale

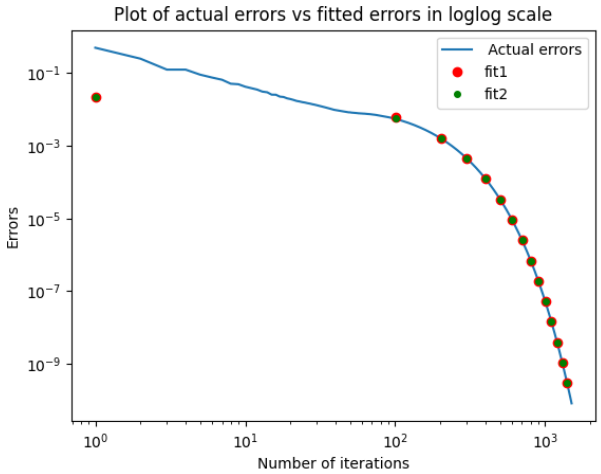


Figure 5: Best fit for error in loglog scale

Cumulative error can also be calculated and it will give us an idea of how much error might be existing in our calculations after $Niter$ iterations.

Figure 6: Plot of Cumulative error

3.6 Surface Plot of Potential:

Now let's do a 3-D plot of the Potential. Code snippet for 3-D plotting of the potential:

```
## Plotting 3-D Surface Plot of Final Potential values
fig1 = plt.figure(7)
plt.title('3-D Surface Plot of Potentials')
ax = p3.Axes3D(fig1)
surf = ax.plot_surface(Y, X, phi, rstride=1, cstride=1, cmap=plt.cm.jet)
plt.show()
```

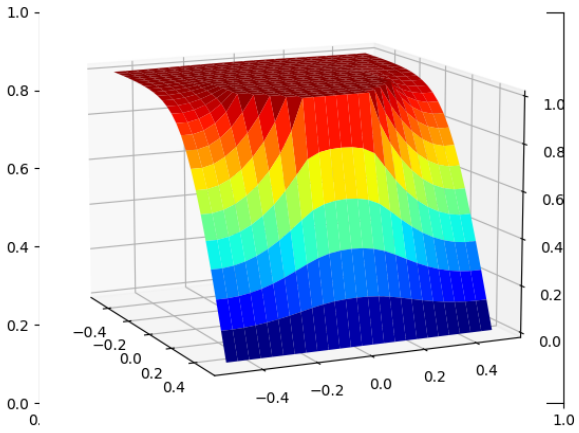


Figure 7: The 3-D Surface Plot of the Potential

3.7 Contour Plot of Potential:

Let's plot a contour plot of the Potential. The nodes in wire portion are marked by red dots.

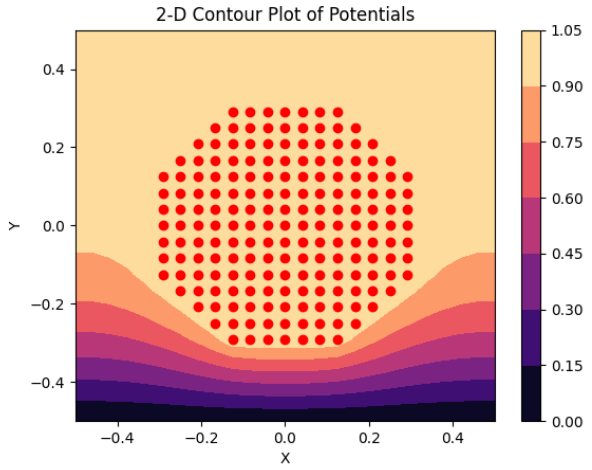


Figure 8: 2D Contour plot of Potential

3.8 Vector Plot of Currents:

Now let's obtain the currents. Here we shall set σ as unity as it's value won't affect the shape of the current profile. We shall create the arrays J_x and J_y and then use quiver command:

```
# Computing current density distribution
Jx, Jy = 1/2 * (phi[1:-1, 0:-2] - phi[1:-1, 2:]), 1/2 * (phi[:-2, 1:-1] - phi[2:, 1:-1])

# Plotting of current density
plt.figure(9)
plt.title("Vector plot of current flow")
plt.quiver(Y[1:-1, 1:-1], -X[1:-1, 1:-1], -Jx[:, ::-1], -Jy)
plt.plot(x[ii[0]], y[ii[1]],'ro')
plt.show()
```

The quiver command creates a vector plot in the desired direction.

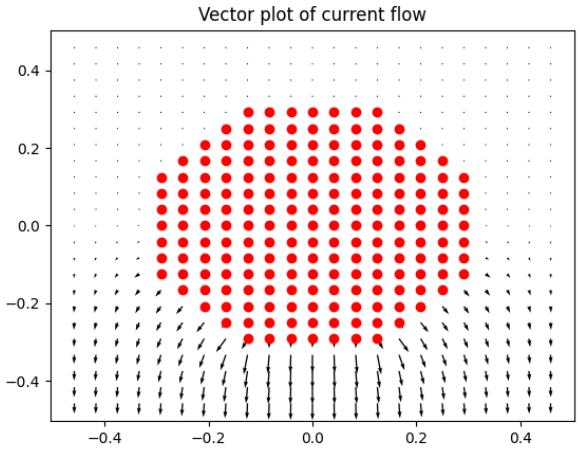


Figure 9: Vector plot of current flow

3.9 Heat Map of the conductor:

Here, similar to finding Potential we iterate over *Niter* times and also apply boundary conditions. Code snippet for finding temperature distribution:

```
for k in range(Niter):
    oldT = T.copy()
    T[1:-1, 1:-1] = 0.25 * (oldT[1:-1, 0:-2] + oldT[1:-1, 2:] + oldT[0:-2, 1:-1] + oldT[2:,
    T[:, 0] = T[:, 1]
    T[:, Nx-1] = T[:, Nx-2]
    T[0, :] = T[1, :]
    T[Ny-1, :] = 300.0
    T[ii] = 300.0
```

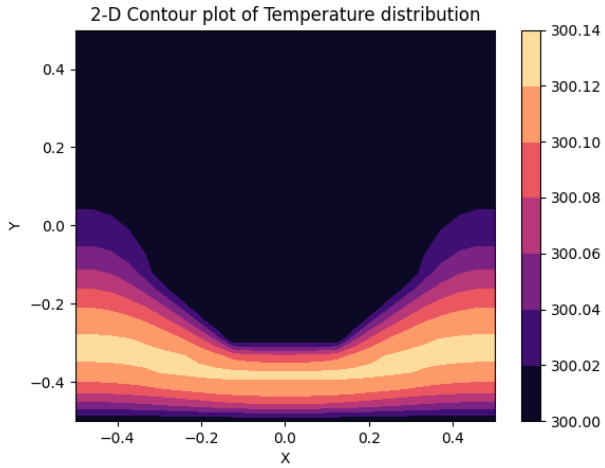


Figure 10: 2D Contour plot of Temperature