Laplace Equation

Name: Sankalp Saoji (EE16B063)

Date: 6/03/2018

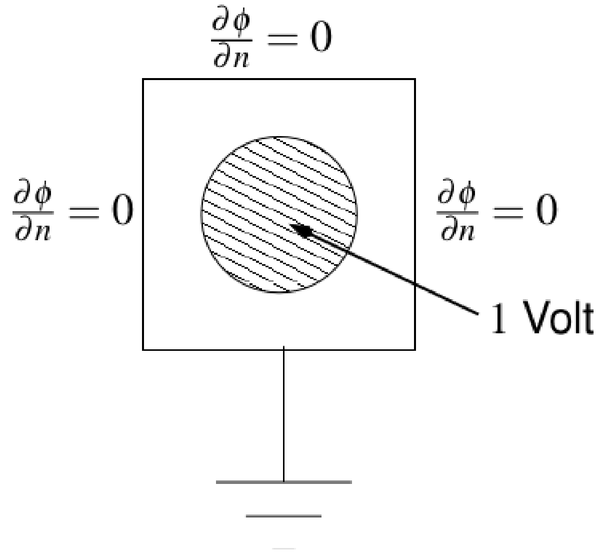
Abstract

In this week, we tried to solve for the currents in a resistor. As the currents depend on the shape of a resistor, we also tried to figure out the part which gets hottest.

1 INTRODUCTION

This week's Python assignment focusses on the following resistor problem.

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 1 cm by 1 in size.


$\frac{\partial\phi}{\partial n}=0$
$\frac{\partial\phi}{\partial n}=0$
$\frac{\partial\phi}{\partial n}=0$
1 Volt

The current at each point can be described by a "current density" $\overline{J}$. This current density is related to the local Electric Field by the conductivity:

$$\overline{J}=\sigma\overline{E}$$

Now the Electric field is the gradient of the potential,

$$\overline{E} = -\nabla\phi$$

Continuity of charge yields,

$$\nabla.\overline{J} = -\frac{\partial\rho}{\partial t}$$

Combining these equations we obtain,

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

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

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

For DC currents, we get solution for the above problem as a Laplace equation of potential as follows,

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

1.1 Numerical Solution of Laplace Equation in 2-Dimensions:

Laplace's equation is easily transformed into a difference eqaution. The equation cane be written in Cartesian coordinates as,

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

Assuming $\phi$to be avaliable at points (x$_i$, y$_j$) , we can write,

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

and,

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

Combining this with equation of y derivatives, we obtain,

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

So, potential t any point will be the average of its neighbours. But, 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.

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

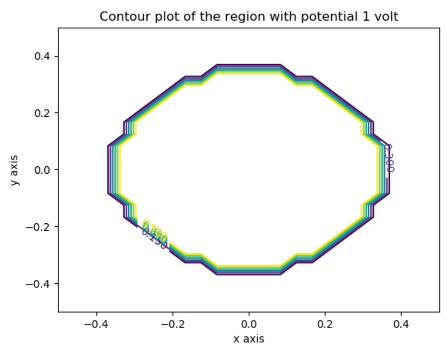
2 Lab Questions

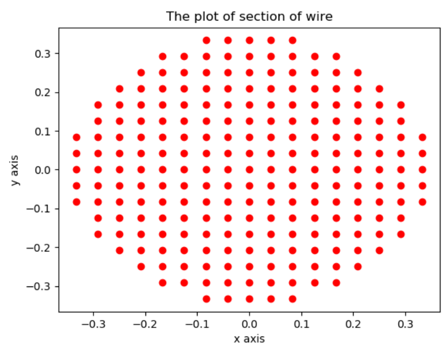
2.1 Plotting the circular region of the wire

In this question we were supposed to plot the wire as contour plot.

```
# Importing libraries and modules

import numpy
import matplotlib.pyplot as plt
import math
import scipy.special as spe
from pylab import *
import mpl_toolkits.mplot3d.axes3d as axes


    # Defining parameters

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

    # Allocating potential array and initializing

    phi = []
    phi = numpy.zeros((Ny,Nx))

    # Finding the circular region with meshgrid

    a = numpy.linspace(-0.5, 0.5, Nx)
    b = numpy.linspace(-0.5, 0.5, Ny)

    X,Y = meshgrid(a,b)
    print X, Y

    ii = numpy.where(X*X + Y*Y <= 0.35*0.35)
    #numpy.where stores indices of the matrix
    phi[ii] = 1.0

    print phi

    # Plotting the contour and circular region

    plt.figure(1)
    con=plt.contour(a, b, phi,inline=True)
    clabel(con,inline=1,fontsize=10)
```

```
plt.title('Contour plot of the region with potential 1 volt')
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.show()

plt.figure(2)
plt.plot(a[ii[0]], b[ii[1]], 'ro')
plt.title('The plot of section of wire')
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.show()
```





2.2 Iterating to get the converged potential values:

In this question we were supposed to do the following,
For k in range (Niter),
1) Save a copy of phi,

```
# Saving a copy of phi

        oldphi = phi.copy()
```

2) Update the phi array,

```
# Updating phi

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

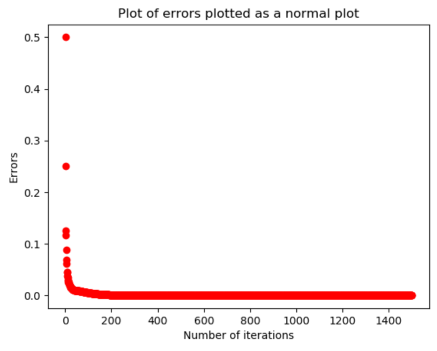
3) Assert boundaries,

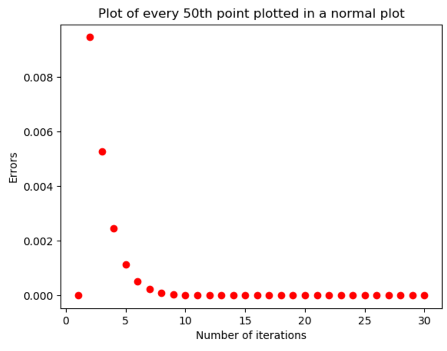
```
#Asserting boundaries

        phi[1:-1,0] = phi[1:-1,1]
        phi[1:-1,24] = phi[1:-1,23]
        phi[0,1:-1] = phi[1,1:-1]
        phi[ii] = 1.0
```

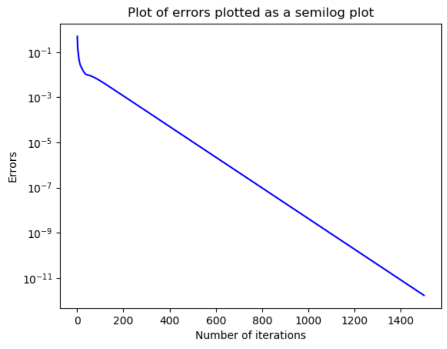
4) Find error, i.e the difference between the current phi and the phi of the previous iteration.

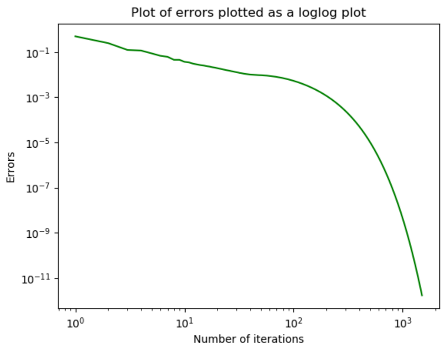
```
# Finding error

        errors[k] = numpy.max(abs(phi-oldphi))
```

Following are the graphs I obtained for errors in a normal plot, semilog plot and loglog plot.









As seen from the graph, the errors reduce slowly. From the loglog plot, we find that the plot gives a reasonably straight line upto about 500 iterations, but beyond that, we get into the exponential regime.

2.3 Combining all the plots and displaying on a single graph:

In this question, we were supposed to extract the exponent. It is an exponential decay only for larger iteration numbers.

The fit is of the form,

y = Ae^{Bx} .

where y is the error and x is the number of iterations.

So, the thing to do is to take the log. Then we have,

log(y) = log(A)+ Bx

That is why it looks like a straight line in a semi-log plot.

So, now we should extract A and B.

For that, we create two matrices of the form,

$$C = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ . & . \\ . & . \\ . & . \\ . & . \\ 1 & 1500 \end{bmatrix} \& \ M = \begin{bmatrix} log(A) \\ B \end{bmatrix}$$

We already have matrix of errors as,

$$\text{errors} = \begin{bmatrix} error_1 \\ error_2 \\ error_3 \\ . \\ . \\ . \\ error_{1500} \end{bmatrix}$$

We use 'lstsq' and obtain 'A' and 'B' now.

```
#Obtaining A and B values

p = [math.log(x) for x in errors]
C = numpy.zeros((Niter, 2))

for i in range(Niter):
        C[i,0] = 1
        C[i,1] = i

M = lstsq(C, p, rcond = -1)[0]
e = M[0]
f = M[1]
A = numpy.exp(e)
B = f
```

Also, we were asked to have three graphs,

1) Plot of errors on semilog scale (error plot)

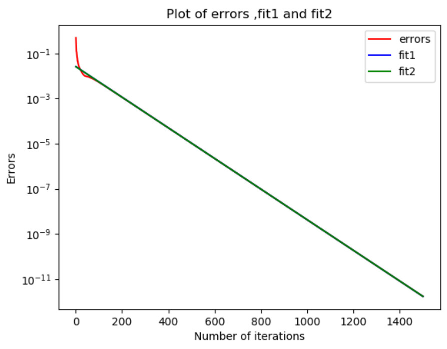
2) Plot of new errors obatined as a result of using lstsq and getting the error matrix again (fit 1 plot)

3) Fit 1 plot plotted after 500th iteration (fit 2 plot).

Here is the way its done.

```
#Obtaining plot of errors, fit1 and fit2
```

```
M = lstsq(C, p, rcond = -1)[0]
e = M[0]
f = M[1]
A = numpy.exp(e)
B = f
error_new1 = numpy.matmul(C, M)
s = numpy.exp(error_new1)

q = p[500:1500]
D = numpy.zeros((1000, 2))

for i in range(1, 1000):
        D[i,0] = 1
        D[i,1] = i+500

N = lstsq(D, q, rcond = -1)[0]
g = N[0]
h = N[1]
A_new = numpy.exp(g)
B_new = h
error_new2 = numpy.matmul(C, N)
t = numpy.exp(error_new2)

r = linspace(1,1500,1500)
n = linspace(1,1500,1500)

plt.figure(7)
plt.semilogy(r, errors, 'r', label = 'errors')
plt.semilogy(r, s, 'b', label = 'fit1')
plt.semilogy(n, t, 'g', label = 'fit2')
plt.title('Plot of errors ,fit1 and fit2')
plt.xlabel('Number of iterations')
plt.ylabel('Errors')
plt.legend()
plt.show()

Error = abs((-A/B)*(numpy.exp(B*(Niter + 0.5))))
print Error
```



Above is the obtained plot with the given legend.

The following are the values of A and B, I obtained,

A = 0.0262155711103

B = -0.0156552640624

$$error_k = 0.0262155711103 * e^{-0.0156552640624*k}$$

$$Error = \Sigma_{k=N+1}^{\infty} 0.0262155711103 * e^{-0.0156552640624*k}$$

$$Error \approx \int_{N+0.5}^{\infty} 0.0262155711103 * e^{-0.0156552640624*k}$$

$$Error = \frac{0.0262155711103}{0.0156552640624} exp(-0.0156552640624(N+0.5))$$

For N=500, we get

$$Error < 0.000662219600621$$

Error went from .62 to .00066 in 500 iterations which can be explained by half life of the exponential.

Half-Life time of the decaying exponential is $\frac{1}{B} = 64$. And 0.00066/64=1.03$*10^{-5}$which can be understood as rate of decrease of error with number of iterations. The

profile was changing very little with every iteration, but was continuously changing. So the cumulative error was still large.

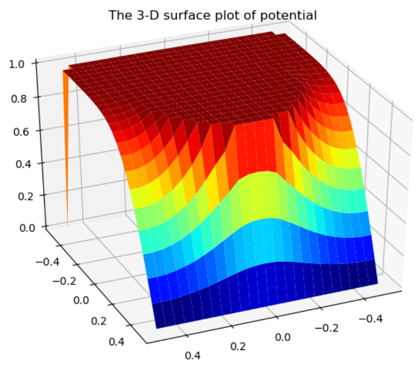
That's why, this method of solving Laplace's Equation is known to be worst one of all the available. This is because of the very slow coefficient with which the error reduces.

2.4 Obtaining surface plot of potential:

In this question, we were asked to create the surface plot of potential.

```
# Creating a surface plot of potential

figure = figure(8)
plotting = axes.Axes3Dfigure)
#Axes3D is the means to do a surface plot
plt.title('The 3-D surface plot of the potential')
surf = plotting.plot_surface(Y, X, phi.T, rstride=1, cstride=1, cmap=cm.jet)
plt.show()
```

The surface plot is obtained as below.

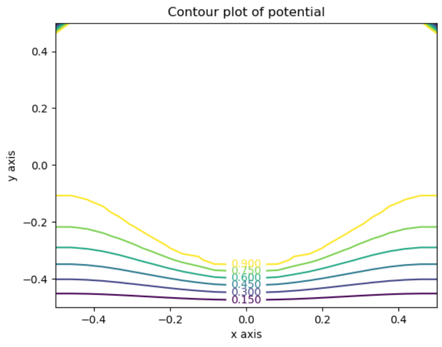


2.5 Obtaining the contour plot of potential:

Here, we had to plot the contour plot of potential.

```
# Creating a contour plot of potential

plt.figure(9)
contour = plt.contour(a, -b, phi,inline=True)
clabel(contour,inline=1.0 , fontsize=10)
plt.title('Contour plot of potential')
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.show()
```

Following is the obtained contour plot.

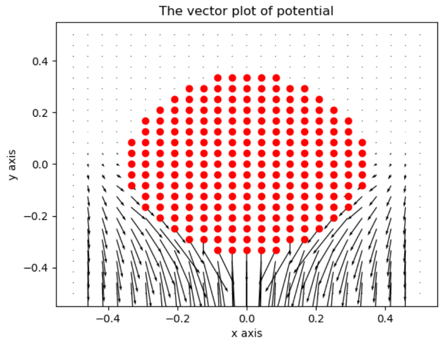


2.6 Obtaining the vector plot of potential:

We were supposed to create the vector plot of potential.

```
# Creating a vector plot of potential

Jx = zeros((Ny,Nx))
#always initialize to zero before playing with matrices
Jy = zeros((Ny,Nx))
Jy[1:-1,1:-1] = -0.5*(phi[1:-1,0:-2]-phi[1:-1,2:])
Jx[1:-1,1:-1] = -0.5*(phi[0:-2,1:-1]-phi[2:,1:-1])

plt.figure(10)
plt.plot(a[ii[0]], b[ii[1]], 'ro')
quiver(b, a, Jy[::-1,:], Jx[::-1,:])
plt.title('The vector plot of potential')
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.show()
```

Following is the obtained vector plot.



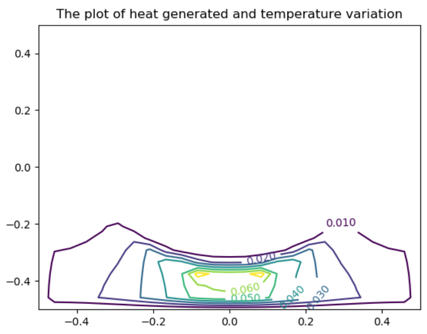
2.7 Finding heat generated

At last, we had to find how the heat is generated in the plate as result of current flowing through the resistor.

Assuming the conductivity σ is 1,

$\nabla.(\kappa\nabla T) = q = |J|^2$

```
#Finding the heat generated

q = Jx**2 + Jy**2
#the value of sigma is taken to be 1
plt.figure(11)
contour1 = plt.contour(b,-a, q, inline = True)
clabel(contour1,inline = 1,fontsize = 10)
plt.title('The plot of heat generated and temperature variation')
plt.show()
```



The heat generated gives rise to the temperature variation.

```
A= 0.0262155711103 B= 0.0156552640624
A= 0.0260439874602 B= 0.0156480693697
```