EE2703 : Applied Programming Lab

Assignment 5

Harisankar K J
EE20B043

11 March 2022

Abstract

The goal of this assignment is the following.

- To solve for currents in a system.
- To solve 2-D Laplace equations in an iterative manner.
- To understand how to vectorize code in python.
- To plot graphs to understand the 2-D Laplace equation.

Introduction

A cylindrical 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*1 cm..
We shall use these equations:
The Continuity Equation:

$$\nabla \cdot \vec{j} = -\frac{\partial \rho}{\partial t} \tag{1}$$

Ohms Law:

$$\vec{j} = \sigma \vec{E} \tag{2}$$

The above equations along with the definition of potential as the negative gradient of Field give:

$$\nabla^2 \phi = \frac{1}{\rho} \frac{\partial \rho}{\partial t} \tag{3}$$

For DC Currents, RHS of equation (3) is 0. Hence:

$$\nabla^2 \phi = 0 \tag{4}$$

0.1 Assignment

0.1.1 Defining Parameters

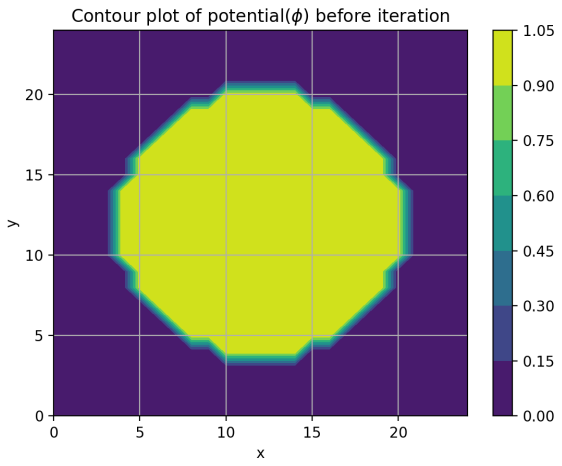
We have chosen a 25x25 grid with a circle of radius 8 centrally located maintained at $V = 1V$ by default. We also choose to run the difference equation for 1500 iterations by default

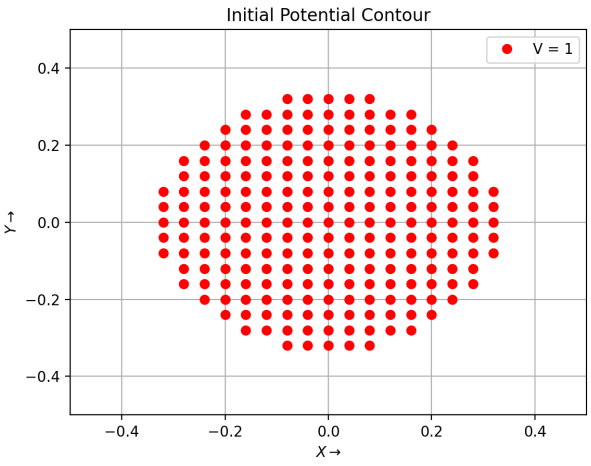
```
import sys
from pylab import *
import mpl_toolkits.mplot3d.axes3d as p3
import numpy as np
import matplotlib.pyplot as plt

Nx = 25          # size along x
Ny = 25          # size along y
radius = 8       # radius of central lead
Niter = 1500     # number of iterations to perform
n = arange(Niter)

if len(sys.argv) == 5:
    Nx = int(sys.argv[1])
    Ny = int(sys.argv[2])
    radius = int(sys.argv[3])
    Niter = int(sys.argv[4])
```

0.1.2 Initializing Potential

We start by creating an zero 2-D array of size Nx x Ny andcoordinates lying within the radius is generated and these points are initialized to 1.

```
phi = np.zeros((Ny, Nx), dtype=float)
x = np.linspace(-0.5, 0.5, Nx)
y = np.linspace(-0.5, 0.5, Ny)
Y, X = np.meshgrid(y, x)
volt1Nodes = np.where(X**2+Y**2 <= 0.35**2)
phi[volt1Nodes] = 1.0
```





0.1.3 Updating Potential Applying Boundary Conditions and Finding The error

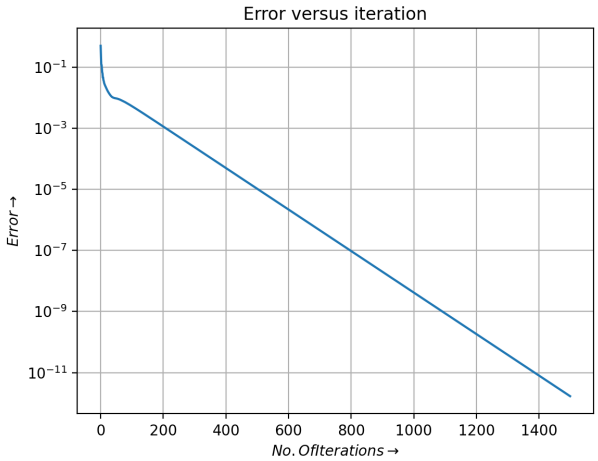
We use Equation(4) to do this.But Equation (4) is a differential equation. We need to first convert it to a difference equation as all of our code is in discrete domain.

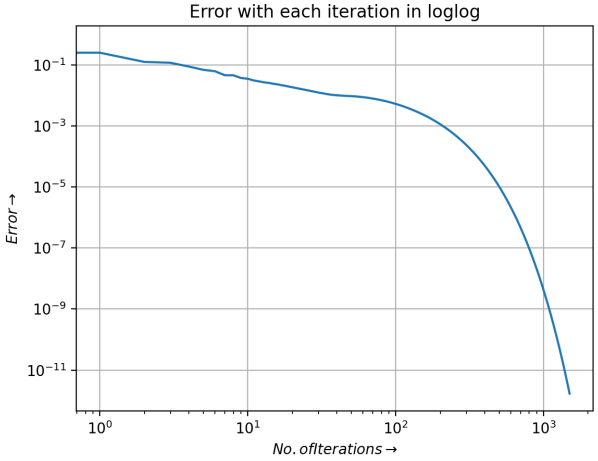
```
errors = np.zeros(Niter)
for k in range(Niter):

    # saving the old value of phi
    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])  # updating phi

    # boundary conditions
    phi[0, 1:-1] = 0                    # bottom edge
    phi[1:-1, 0] = phi[1:-1, 1]         # left edge
    phi[-1, 1:-1] = phi[-2, 1:-1]       # top edge
    phi[1:-1, -1] = phi[1:-1, -2]       # right edge

    phi[0, 0] = 0.5*(phi[0, 1] + phi[1, 0])  # updatig the left
        out corners
    phi[0, -1] = 0.5*(phi[0, -2] + phi[1, -1])
    phi[-1, 0] = 0.5*(phi[-1, 1] + phi[-2, 0])
    phi[-1, -1] = 0.5*(phi[-1, -2] + phi[-2, -1])

    # resetting the voltage = 1 in region in contact with
        electrode
    phi[volt1Nodes] = 1.0

    # maxmum error after each ieration is stored
    errors[k] = (abs(phi-oldphi)).max()
```

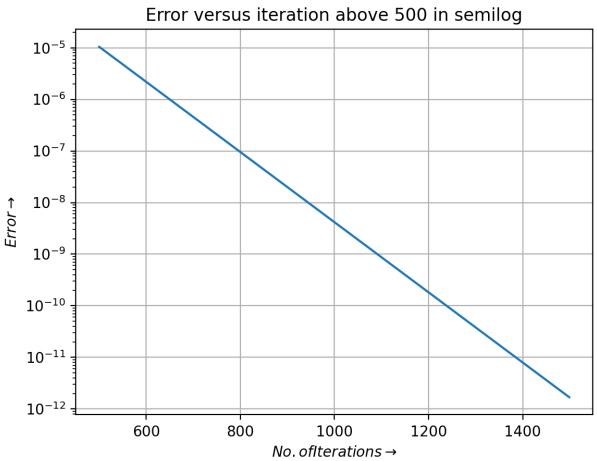
0.1.4 Plotting the Error in Semilog and Loglog

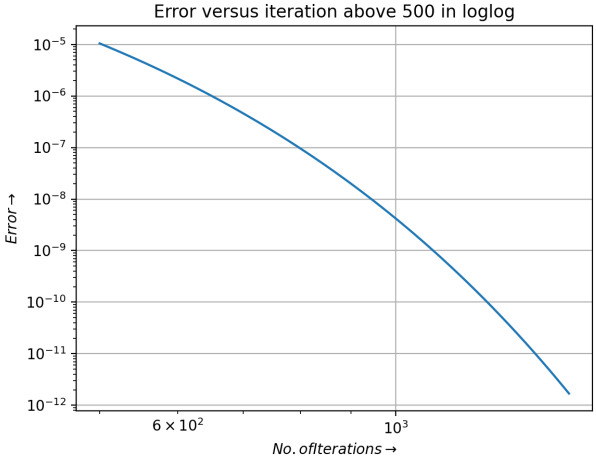
```
# Plotting the value of error vs iteration in semilog.
plt.figure(3)
plt.title("Error versus iteration")
plt.xlabel(r'$No. Of Iterations \rightarrow$')
plt.ylabel(r'$Error \rightarrow$')
plt.semilogy(range(Niter), errors)
plt.grid(True)
plt.show()
```

```
# Plotting the value of error vs iteration in loglog.
plt.figure(4)
plt.title("Error with each iteration in loglog")
plt.xlabel(r'$No. of Iterations\rightarrow$')
plt.ylabel(r'$Error\rightarrow$')
plt.loglog(n, errors)
plt.grid(True)
plt.show()

# Plotting the value of error vs iteration above 500 in semilog
    .
plt.figure(5)
plt.title("Error versus iteration above 500 in semilog")
plt.xlabel(r'$No. of Iterations\rightarrow$')
plt.ylabel(r'$Error\rightarrow$')
plt.semilogy(arange(Niter)[500:], errors[500:])
plt.grid(True)
plt.show()

# Plotting the value of error vs iteration above 500 in loglog .
plt.figure(6)
plt.title("Error versus iteration above 500 in loglog")
plt.xlabel(r'$No. of Iterations\rightarrow$')
plt.ylabel(r'$Error\rightarrow$')
plt.loglog(arange(Niter)[500:], errors[500:])
plt.grid(True)
plt.show()
```





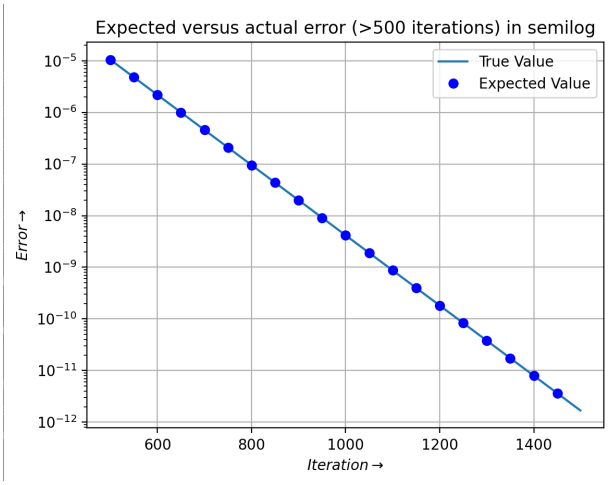

Error versus iteration above 500 in semilog
Error →
No. ofIterations →
600
800
1000
1200
1400

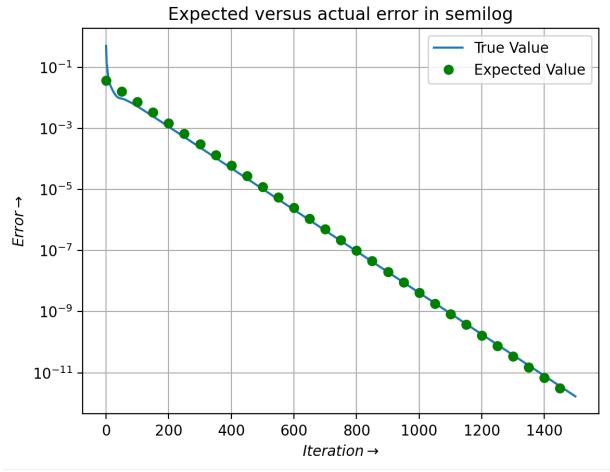

Error versus iteration above 500 in loglog
Error →
No. ofIterations →
6×10^2
10^3

0.1.5 Fitting The Error

Using least squares method to apprximate the errors wrt iterations.We note that the error is decaying exponentially for higher iterations. Plotting expected error vs actual error values in semilog and loglog. Notice that the exected value comes closer to the actual values as iterations increase.

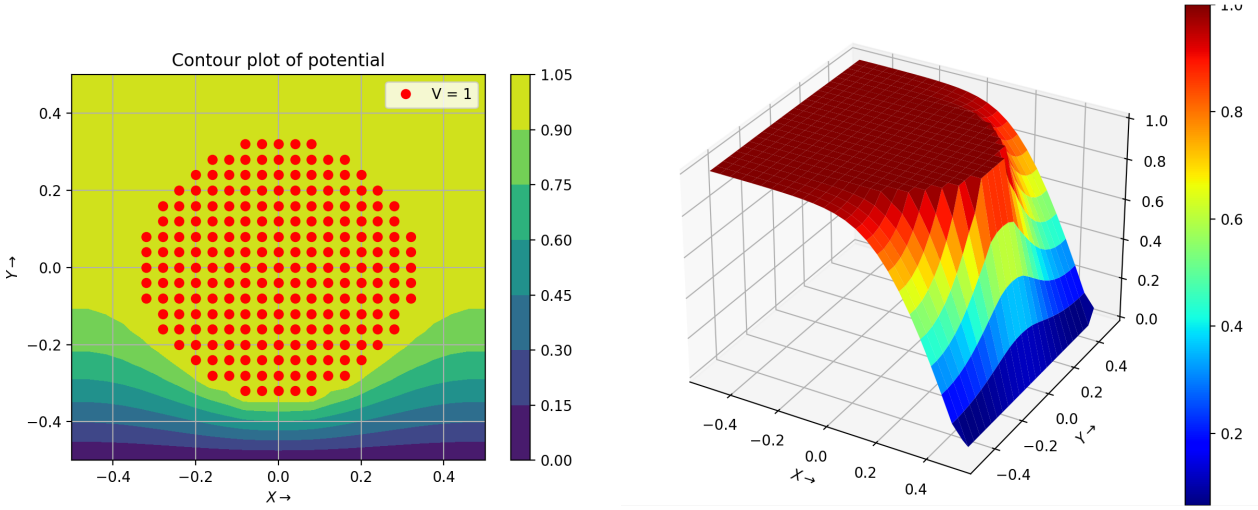
```
# Plotting of actual and expected error (above 500 iterations)
    in semilog.
plt.figure(7)
plt.semilogy(arange(Niter)[500:], errors[500:], label="True␣
    Value")
plt.semilogy(arange(Niter)[500::50], np.exp(
     p500[1])*exp(p500[0]*arange(Niter)[500::50]), "bo", label="
         Expected␣Value")
plt.title("Expected␣versus␣actual␣error␣(>500␣iterations)␣in␣
    semilog")
plt.xlabel(r'$Iteration\rightarrow$')
plt.ylabel(r'$Error\rightarrow$')
plt.grid(True)
plt.legend()
plt.show()

# Plotting of actual and expected error in semilog.
plt.figure(8)
plt.semilogy(arange(Niter), errors, label="True␣Value")
plt.semilogy(arange(Niter)[::50], np.exp(
     p[1])*exp(p[0]*arange(Niter)[::50]), "go", label="Expected␣
         Value")
plt.title("Expected␣versus␣actual␣error␣in␣semilog")
plt.xlabel(r'$Iteration\rightarrow$')
plt.ylabel(r'$Error\rightarrow$')
plt.grid(True)
plt.legend()
plt.show()
```





0.1.6 Plotting the Potential and Surface Plot After Iteration

```
# Plotting of the contour of phi (potential).
plt.figure(9)
plt.contourf(Y, X, phi)
plt.plot(volt1Nodes[0]/Nx-0.48, volt1Nodes[1]/Ny-0.48, 'ro',
    label="V = 1")
plt.title("Contour plot of potential")
plt.xlabel(r'$X\rightarrow$')
plt.ylabel(r'$Y\rightarrow$')
plt.colorbar()
plt.grid(True)
plt.legend()
plt.show()

# Plotting the surface plots of phi.
fig10 = figure(11)
ax = p3.Axes3D(fig10)
title("The 3-D surface plot of the potential")
xlabel(r'$X\rightarrow$')
ylabel(r'$Y\rightarrow$')
surf = ax.plot_surface(-X, -Y, phi, rstride=1, cstride=1, cmap=
    cm.jet)
fig10.colorbar(surf)
show()
```



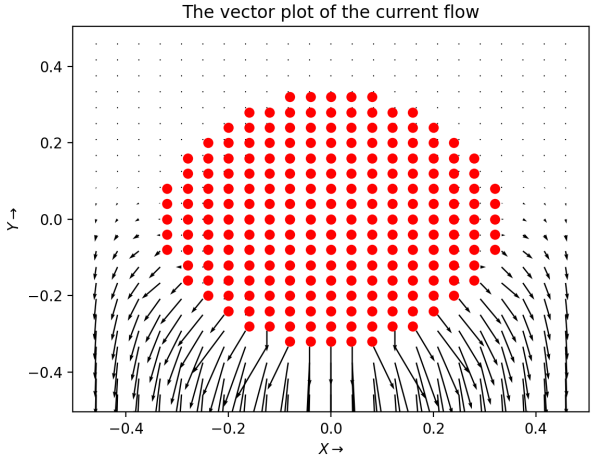
0.1.7 Finding and Plotting J

$$J_{x,ij} = 0.5 * (\phi_{i,j-1} - \phi_{i,j+1}) \tag{5}$$

$$J_{y,ij} = 0.5 * (\phi_{i-1,j} - \phi_{i+1,j}) \tag{6}$$

Using the above equation to find the values of current density at each point and plotting the current density vector along with potential.

```
# the current density at each point is calculated by using the
    formula given
Jx = zeros((Ny, Nx))
Jy = zeros((Ny, Nx))
Jx, Jy = (1/2*(phi[1:-1, 0:-2]-phi[1:-1, 2:]),
          1/2*(phi[:-2, 1:-1]-phi[2:, 1:-1]))
print(size(Jx), "    ", size(Jy))

# plotting of the current vector plot along with the potential.
plt.figure(10)
plt.quiver(-Y[1:-1, 1:-1], X[1:-1, 1:-1], Jx[:, ::-1], Jy)
plt.title("The vector plot of the current flow")
plt.xlabel(r'$X\rightarrow$')
plt.ylabel(r'$Y\rightarrow$')
plt.plot(volt1Nodes[0]/Nx-0.48, volt1Nodes[1]/Ny-0.48, 'ro')
plt.show()
```



Conclusion

- To conclude , most of the current is in the narrow region at the bottom. So this region will get 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 down region of the plate.