EE2703 : Applied Programming Lab
Assignment 5

Rakshit Pandey
EE20B106

March 11, 2022

1. AIM OF THE ASSIGNMENT

In week 5 assignment we were supposed to do the following things:

- To solve the Laplace's equation for 2D in iterative manner.
- To plot various curves,i.e. normal plots,contour plots and 3D plots.

2.INTRODUCTION TO THE ASSIGNMENT

A wire is soldered to the middle part of a copper plate and it's voltage is held at 1V. One side of the plate is grounded, while the remaining are floating. The plate is 1 cm by 1 cm in size.

Equations

The equation relating conductivity, current density and Electric field is:

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

Now, The electric field is the gradient of the potential and is given by:

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

and the continuity of charge yields:

$$\nabla \cdot \tilde{\mathbf{J}} = -\frac{\partial \rho}{\partial t}$$

Now, assuming that the resistor contains a material of constant conductivity, the equation becomes:

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

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

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

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

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

Solving this reduces to the result that, the potential at any point should be the average of it's neighbours.

3.ASSIGNMENT QUESTIONS

1.Importing important libraries

In order to do this assignment we have to import some important libraries for working with matrices, plotting 3D curves, plotting curves, etc.

Code

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

```

2.Taking the command line argument from the user

In order to do this assignment we need the values of Nx(no. of points in x direction), Ny(no. of points in y direction), r(radius), Niter(no. of iterations). In case, if user doesn't enter commandline arguments, we will take the default values as follows:

$$N_x = 25, N_y = 25, r = 8, Niter = 1500,$$

Code

```
    if(len(argv)>1):
         Nx = int(argv[1])
         Ny = int(argv[2])
         r = float(argv[3])
         Niter = int(argv[4])
       else:
         Nx = 25
         Ny = 25
         r = 8
         Niter = 1500


```

3. Intitalizing the potential matrix

We initialize the potential matrix (ϕ) with the dimensions N_y x N_x, by putting all values as zeros, then we find the set of points where $X^2 + Y^2$ ¡= 0.35^2 and call them ii, and the potential at those points is set as 1.0V, all this is executed in python as follows:

Code

```
1
2     phi = np.zeros((Ny,Nx)) #Initializing the phi matrix
3     n = np.arange(Niter)
4     x = np.linspace(-0.5,0.5,Nx)
5     y = np.linspace(-0.5,0.5,Ny)
6
7     Y,X = np.meshgrid(y,x)
8
9     ii = np.where(X*X + Y*Y <= 0.35*0.35)
10    phi[ii] = 1.0
11
```

4. Contour plot of potential

Code

```
1     plt.figure(1)
2     plt.xlabel("x$\Rightarrow$")
3     plt.ylabel("y$\Rightarrow$")
4     plt.contourf(range(Nx),range(Ny),phi)
5     plt.colorbar()
6     plt.title("Contour plot of potential")
```

We obtain the following curve for the given code :

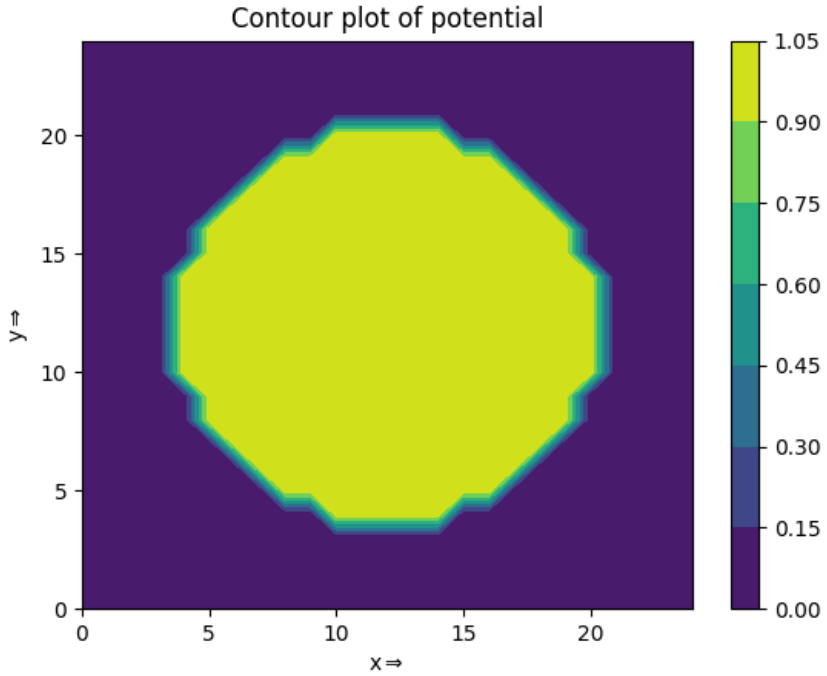

Figure 1: Contour plot of potential

5.Calculating phi error

We then calculate the successive errors in phi by iterating it for Niter times. The updating scheme of phi matrix follows the rule that the potential at any point is equal to the average of it's neighbour potentials, and in case of boundary points we equate boundary points potential to the potential of their adjacent layer points. While doing this care must be taken to store the intial potential array as oldphi and then, only make changes to it. This is executed in python as follows:

Code

```
    phi_error = []
    for k in range(Niter):
      oldphi = phi.copy()
      phi[1:-1,1:-1] = (oldphi[0:-2,1:-1] + oldphi[2:,1:-1] + oldphi[1:-1,2:] +
                              oldphi[1:-1,0:-2])*0.25
      phi[:,0] = phi[:,1]
      phi[:,-1] = phi[:,-2]
      phi[0,:] = phi[1,:]
      phi[-1,:] = 0.0
      phi[ii] = 1.0
      phi_error.append((abs(phi-oldphi)).max())
```

6.Making curves

For our analysis and ease of understanding solutions we make the curves for error vs iteration in loglog and semilog scale , and that too for two cases, one which involves complete iteration points and the other one which involves iteration points above 500. This is executed in python as follows:

Code

```
    plt.figure(2)
    plt.xlabel("No. of iterations")
    plt.ylabel("Error")
    plt.grid()
    plt.semilogy(n,phi_error)
    plt.title("Error vs Iterations in a semilog scale")

    plt.figure(3)
    plt.xlabel("No. of iterations")
    plt.ylabel("Error")
    plt.grid()
    plt.loglog(n,phi_error)
    plt.title("Error vs iterations in a loglog scale")

    plt.figure(4)
    plt.xlabel("No. of iterations")
    plt.ylabel("Error")
    plt.grid()
    plt.semilogy(n[500:],phi_error[500:])
    plt.title("Error vs iterations in semilog scale for >500 iterations")

    plt.figure(5)
    plt.xlabel("No. of iterations")
    plt.ylabel("Error")
    plt.grid()
    plt.loglog(n[500:],phi_error[500:])
    plt.title("Error vs iterations in loglog scale for >500 iterations")

```

And the following curves are obtained:

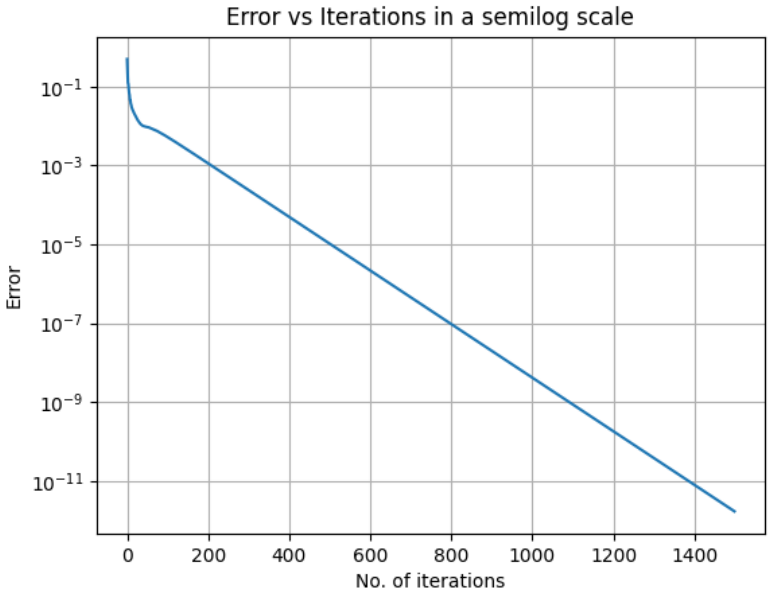

Figure 2: error vs iteration in semilog scale

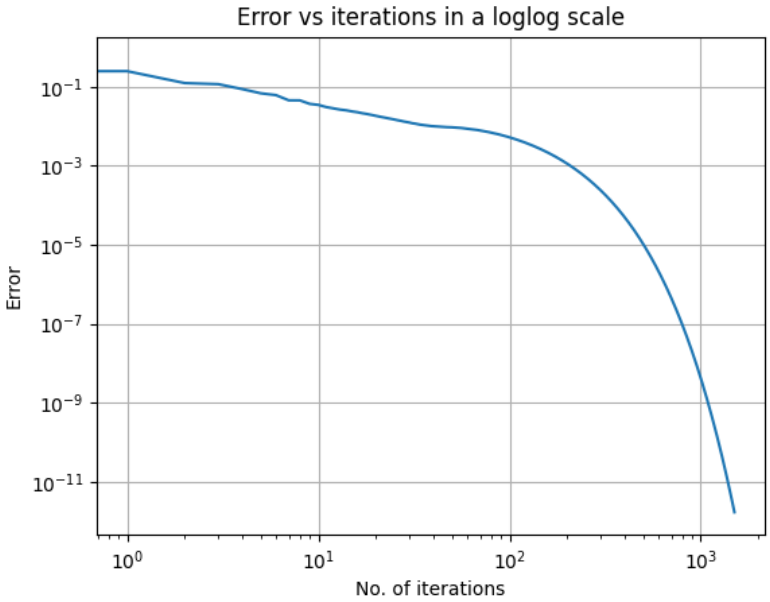

Figure 3: error vs iteration in loglog scale

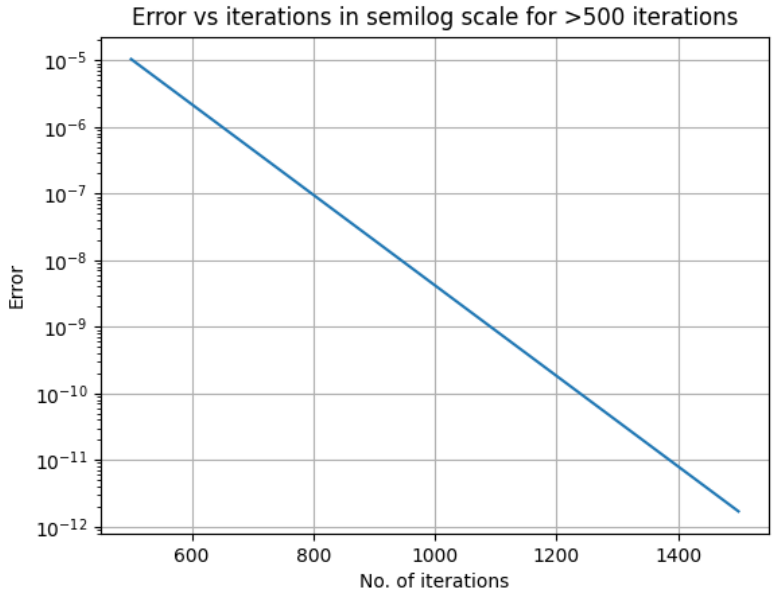

Figure 4: error vs iteration in semilog scale for iterations ¿500

Figure 5: error vs iteration in loglog scale for iterations ¿500

7. Finding the Stopping condition

We find the best fit for log(A) and B in the error equation and fit the following exponential to our error vs iteration and then plot it against the true value to analyze the difference. This is done in python as follows:

Code

```
        def exp_fit(x,A,B):
          return(A*np.exp(B*x))

        def fit(x,y):
          logy = np.log(y)
          xvec=np.zeros((len(x),2))
          xvec[:,0]=x
          xvec[:,1]=1
          B,logA=np.linalg.lstsq(xvec, np.transpose(logy),rcond = None)[0]
          return (np.exp(logA),B)

        A1,B1 = fit(n,phi_error)
        A2,B2 = fit(n[500:],phi_error[500:])

        def max_error(n,A,B):
          return -1.0*(A/B)*exp(B(n+0.5))

        plt.figure(6)
        plt.xlabel("No. of iterations")
        plt.ylabel("Error")
        plt.grid()
        plt.semilogy(n,phierror,label="Actual plot")
        plt.semilogy(n[::50],expfit(n[::50],A1,B1),"go",label="Fit for all points
    ")
        plt.semilogy(n[::50],expfit(n[::50],A2,B2),"bo",label="Fit for points
    >500")
        plt.legend(loc="upper right")
        plt.title("Errorfit and Error vs iterations")

```

The following curve is obtained:

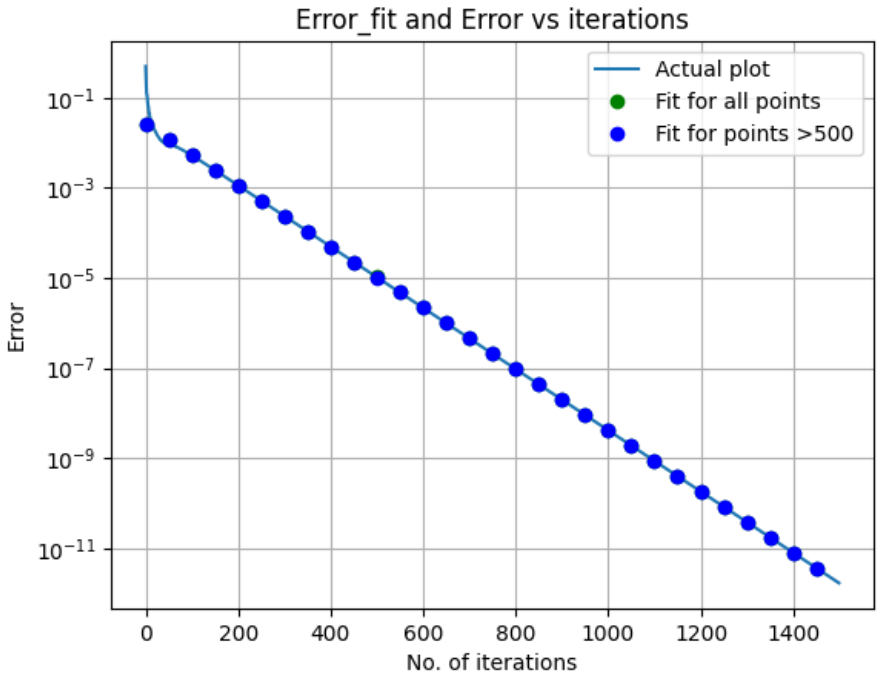

Figure 6: error fit and error vs iteration

From the curve is can be easily observed that for iterations greater than 500 the two almost conincide exactly whereas the coinciding of curves is not exact in case of the first few points.

8.Making the 3D surface plot of potential

Code

```
        fig7= plt.figure(7)
        ax=p3.Axes3D(fig7)
        surf = ax.plot_surface(Y, X, phi.T, rstride=1, cstride=1,cmap=pylab.cm.
    jet)
        pylab.xlabel(r'x$\rightarrow$',fontsize=15)
        pylab.ylabel(r'y$\rightarrow$',fontsize=15)
        ax.set_zlabel(r'$\phi\rightarrow$',fontsize=15)
        ax.set_title("The 3D surface plot of the potential")

```

The curve obtained is as follows:

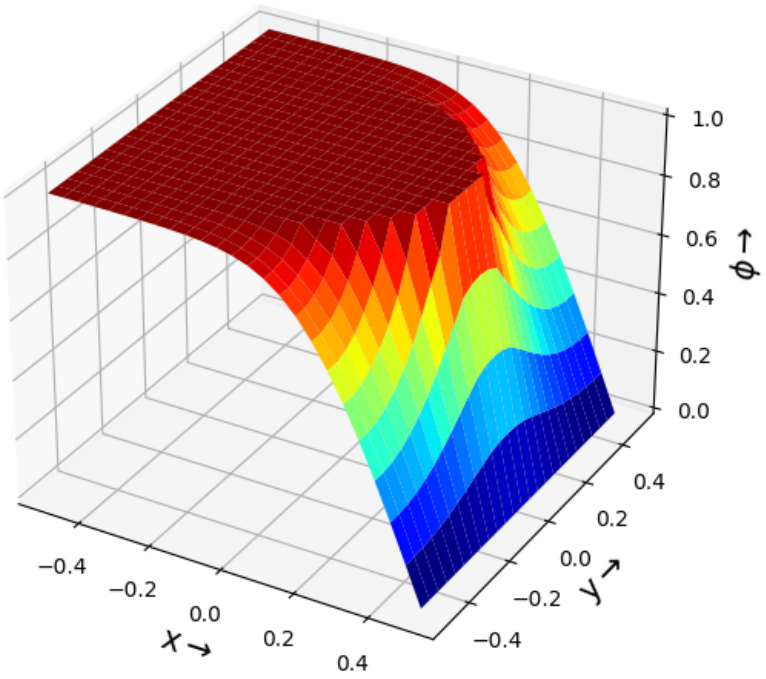

Figure 7: 3D surface plot for potential

9.Making the contour plot for potential with electrode marked as red dots

Code

```
    plt.figure(8)
    plt.xlabel(r'$\rightarrow$')
    plt.ylabel(r'$\rightarrow$')
    plt.plot((ii[0]-Nx/2)/Nx,(ii[1]-Ny/2)/Ny,'ro')
    plt.contour(Y,X[::-1],phi)
    plt.colorbar()
```

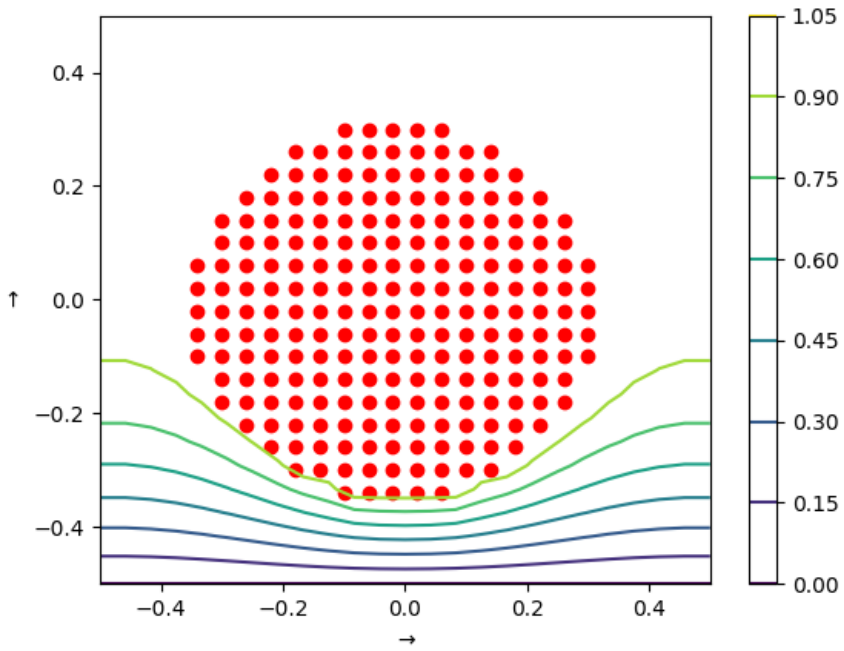

Figure 8: Contour plot of ϕ

10. Vector plot for current flow

Code

```
    plt.figure(9)
    plt.quiver(x,y,Jx[::-1,:],Jy[::-1,:])
    plt.xlabel(r'$\rightarrow$')
    plt.ylabel(r'$\rightarrow$')
    plt.plot(ii[0]/Nx-0.48,ii[1]/Ny-0.48,'ro')
    plt.title("The vector plot of the current flow")

    plt.show()
```

The following curve is obtained:

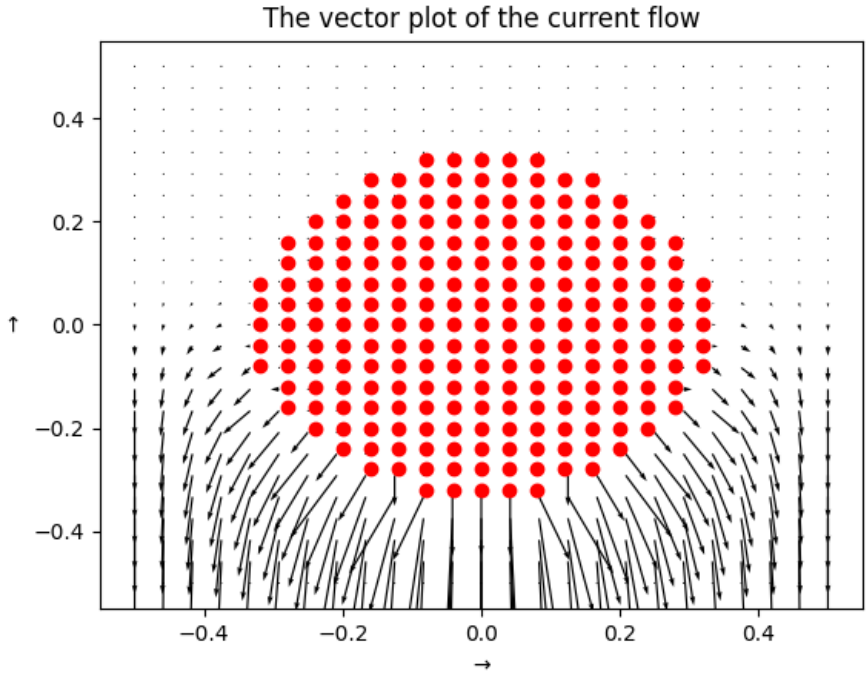

Figure 9: Vector plot of current flow

RESULTS

Our solution involved the use of Laplace's equation for the given analysis using a finite differentiation approximation. It is found out that the chosen method of Laplace's equation is inefficient. The error decays at a highly gradual pace. If we carefully analyze the vector plot of current then it is found that the currently was mostly restricted to the bottom part of the wire, and was perpendicular to the surface of the electrode and the conductor.

Also, since there is almost no current in upper regions and almost all the current flows in the bottom region, hence the bottom part gets the hottest among all the parts.