EE2703 : Applied Programming Lab
Experiment 5

Ch Venkat Sai
EE18B042

March 04, 2020

Overview

The experiment is about:
Solving for the numerical solution for laplace equation in 2-D with the boundary conditions and initial conditions given using the iterative technique.
Plotting the error in each iteration and finding the best fit function for the error.
Plotting the surface plot and contour plot of the potential along with the vector plot of the current.

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 1 cm by 1 cm in size. As a result, current flows. The current at each point can be described by a "current density"- j. This current density is related to the local Electric Field by the conductivity:

$$\vec{j} = \sigma \vec{E} \tag{1}$$

Now the electric field is the gradient of the potential,

$$\vec{E} = - \bigtriangledown \phi \tag{2}$$

and continuity of charge yields

$$\bigtriangledown . \vec{j} = -\frac{\partial \rho}{\partial t} \tag{3}$$

Combining these equations and assuming a material with constant conductivity and DC currents flowing, we obtain

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

In 2-D the equation becomes:

$$\frac{\partial^2 \phi}{\partial^2 x} + \frac{\partial^2 \phi}{\partial^2 y} = 0 \tag{5}$$

Assuming ϕ is available at (x_i, y_j) we obtain,

$$\phi_i j = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi i,_{j-1}}{4} \tag{6}$$

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 should be tangential. This is implemented by requiring that should not vary in the normal direction.

Define the Parameters

We have to define the x and y length of the plate and radius and the number of iterations to be performed.

```
    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])
else:
    Nx=25;
    Ny=25;
    radius=8;
    Niter=1500;
```

Allocate the potential array and initialize it

First we create the ϕ array and initialize it to zero. And then we find the points which occupy the electrode using the radius and initialize them to zero. We find the points using where() function.

```
phi = zeros([Ny, Nx])

x = linspace(-int((Nx-1)/2),int((Nx-1)/2),Nx)
y = linspace(int((Ny-1)/2),-int((Ny-1)/2),Ny)
X,Y = meshgrid(x,y)

ii = where(X*X + Y*Y <= radius*radius)
phi[ii] = 1.0
errors = zeros(Niter)
```

Updating the Potential

We use python vectors to update the potential in each iteration using the equation we obtained finally and the boundary conditions. Using python vectors instead of for loops drastically reduces the time of execution.

```
x_scale = []
for k in range(Niter):
    oldphi = phi.copy()
    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])
    phi[1:-1,0] = phi[1:-1,1]
    phi[1:-1,-1] = phi[1:-1,-2]
    phi[0] = phi[1]
    phi[ii] = 1.0
    errors[k] = (abs(phi-oldphi)).max()
    x_scale.append(k)
```

Here in the vectorized code $phi[a, b]$, a corresponds to the rows and b to columns. The boundary conditions being potential at the top, left, right boundary are equal to the previous row or column. And the bottom one stays at zero. While doing this the the electrode may get disturbed so we once again assign it to one. We also note the errors in each iteration,

Graph the results

We plot the semilog plot and the loglog plot of the errors using every 50th data point. We see that the semilog plot is linear(Figure.1) and the loglog(Figure.2) plot gives a reasonably straight line upto about 500 iterations, but beyond that, we get into the exponential regime. So we assume the error varies as exponential and find the best fit for it (1) using all the data and (2) using data after the 500th iteration. And we then plot this bestfit 1 and 2 along with the error in the Figure.1.

```
x_scale = array(x_scale)
semilogy(x_scale[::50], errors[::50], '-r', label='errors semilogy')
s = []
errors1 = []
for k in range(Niter):
    s.append([1, x_scale[k]])
for k in range(Niter):
```

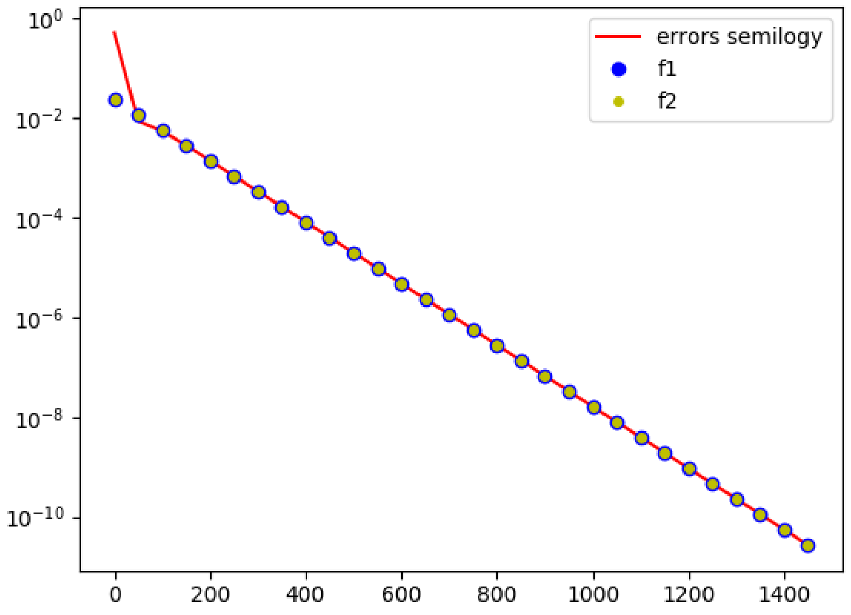

Figure 1: Errors, bestfit1, 2- semilog

```
    errors1.append(log(errors[k]))
errors1 = array(errors1)
s = array(s)
m = array(lstsq(s, errors1, rcond=None)[0])
A = exp(m[0])
B = m[1]
semilogy(x_scale[::50], A*exp(B*x_scale[::50]),'ob', label='f1')
m = array(lstsq(s[500:], errors1[500:], rcond=None)[0])
A = exp(m[0])
B = m[1]
semilogy(x_scale[::50], A*exp(B*x_scale[::50]),'og', label='f2')
legend(loc='upper right')
show()
loglog(x_scale[::50], errors[::50], 'og', label='errors loglog')
legend(loc='upper right')
show()
```

The figures:

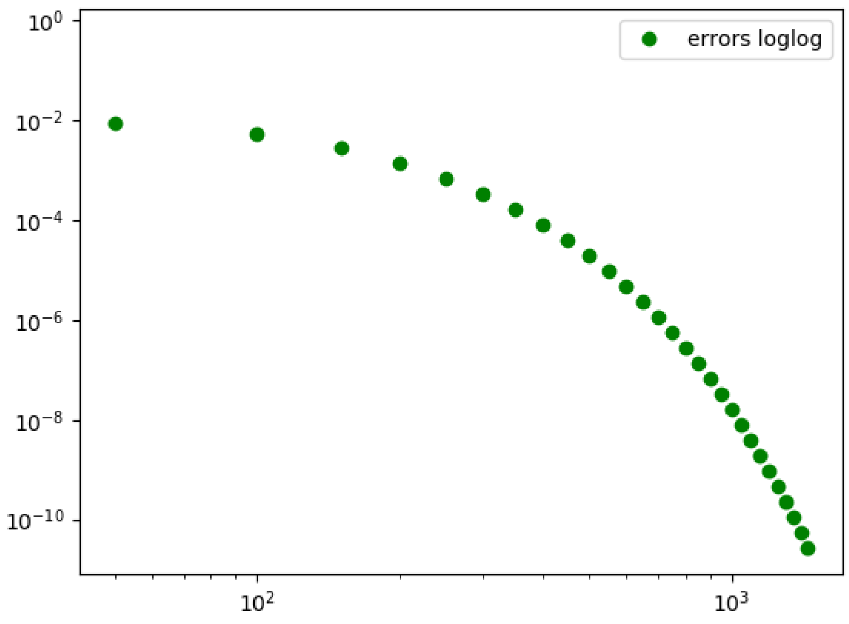

Figure 2: Errors - loglog

Contour Plot of the Potential

Plotting the contour plot of the potential w.r.t the points as shown in Figure.3

```
c = contourf(X, Y, phi)
colorbar(c)
show()
```

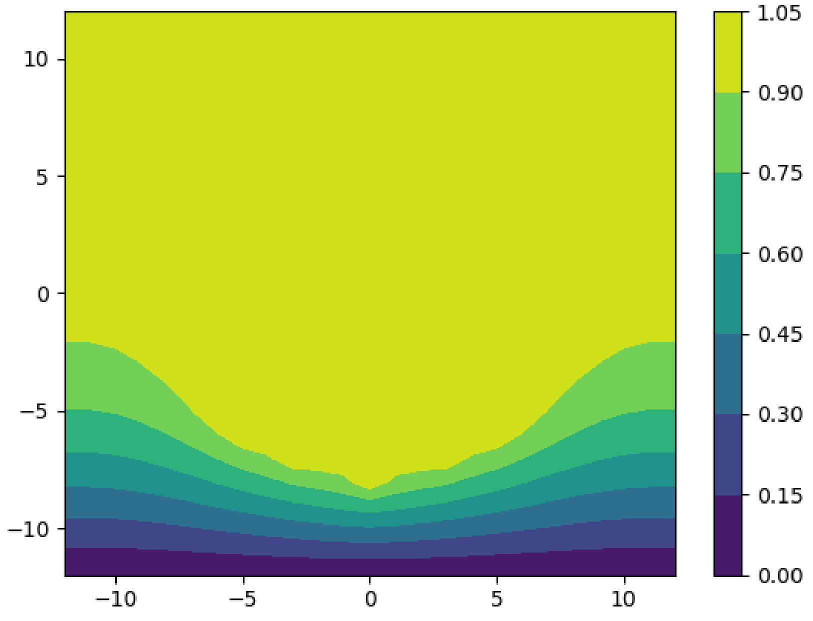

Figure 3: Contour plot

Surface Plot of Potential

Plotting the surface plot of the potential w.r.t the points as shown in Figure.4

```
fig4 = figure(4) # open a new figure
ax = p3.Axes3D(fig4) # Axes3D is the means to do a surface plot
title(r'The 3-D surface plot of the potential')
surf = ax.plot_surface(X, Y, phi, rstride=1, cstride=1, cmap=cm.jet,
linewidth=0, antialiased=False)
show()
```

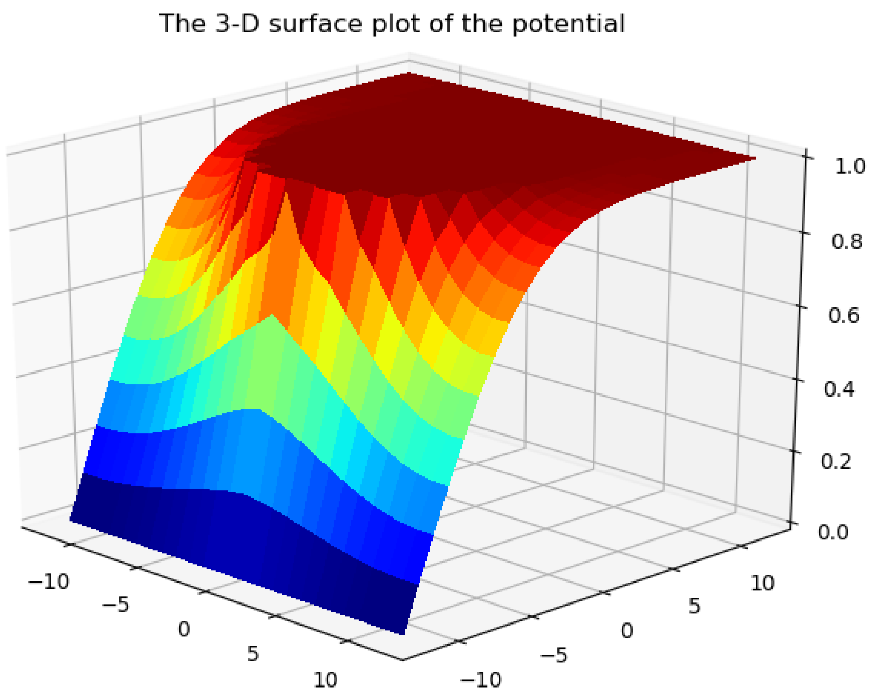

Figure 4: Surface plot 3-D

Vector Plot of Currents

Now obtain the currents. This requires computing the gradient. The actual value of *sigma* does not matter to the shape of the current profile, so we set it to unity. Our equations are:

$$j_x = -\frac{\partial \phi}{\partial x} \tag{7}$$

$$j_y = -\frac{\partial \phi}{\partial y} \tag{8}$$

This numerically translates to:

$$j_{x,ij} = \frac{\phi_{i,j-1} - \phi_{i,j+1}}{2} \tag{9}$$

$$j_{y,ij} = \frac{\phi_{i-1,j} - \phi_{i+1,j}}{2} \tag{10}$$

And then we use quiver to plot the current vector plot.

```
x1 = linspace(0, Nx, Nx)
y1 = linspace(Ny, 0, Ny)
X1,Y1 = meshgrid(x1,y1)
Jx = zeros([Ny, Nx])
Jy = zeros([Ny, Nx])
Jx[1:-1,1:-1] = 0.5*(phi[1:-1, 0:-2] - phi[1:-1, 2:])
Jy[1:-1,1:-1] = 0.5*(phi[0:-2, 1:-1] - phi[2:, 1:-1])
figure(5)
plot(X1[ii],Y1[ii], 'r.')
quiver(X1, Y1, -Jx, -Jy, scale=0.85, headwidth=2, scale_units='inches')
show()
```

The plot shows up like:

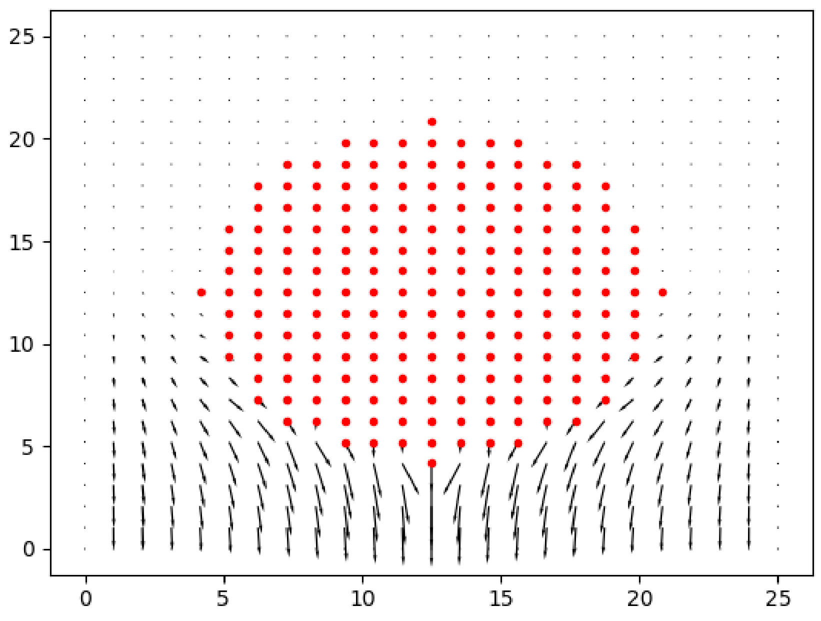

Figure 5: Current Vector plot

Conclusion

In conclusion we learnt how to solve the laplace equation numerically and the use of python vectors. As we increase the number of iterations the error keeps on decreasing. We also plotted the surface plot, contour plot and the vector current plot. We even fund out that he error in each iteration decreases exponentially.