Assignment No 5

Sumanth R Hegde, EE17B032

March 5, 2019

Aim

The aim of this assignment is to obtain the solution to potential in a region by solving *Laplace's* equation in 2-dimensions.

Numerical Solution to Laplace's equation

Laplace's equation in 2-dimensions can be written in Cartesian coordinates as

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

An approximate solution for the above equation for a 2-dimensional grid of points would be

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

ie. the solution at any point is a sum of values at neighbouring points. The algorithm implemented makes use of the above equation to update ϕ over many iterations till it converges within an acceptable error.

The Potential Solution

Along with the update mentioned in Eqn.1 we also need to take care of the boundary conditions imposed on the system. The following code blocks update the value of ϕ over `Niter` iterations, along with the error profile :

```
def phi_update(phi,oldphi):
    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])
    return phi
```

```
def boundary_conditions(phi,inds):
    phi[1:-1,0] = phi[1:-1,1]
    phi[0,1:-1] = phi[1,1:-1]
    phi[1:-1,-1] = phi[1:-1,-2]
    phi[-1,1:-1] = 0
    phi[inds] = 1.0
    return phi

for i in range(Niter):
    oldphi = phi.copy()
    phi = phi_update(phi,oldphi)
    phi = boundary_conditions(phi,inds)
    error[i] = (abs(phi-oldphi)).max()
```

The initial potential configuration of the system can be visualised using the `pylab.contourf()` function :

```
pylab.contourf(range(Nx),range(Ny),phi.T,cmap=pylab.cm.get_cmap("autumn"))
pylab.xlabel(r'x$\rightarrow$',fontsize=15)
pylab.ylabel(r'y$\rightarrow$',fontsize=15)
```

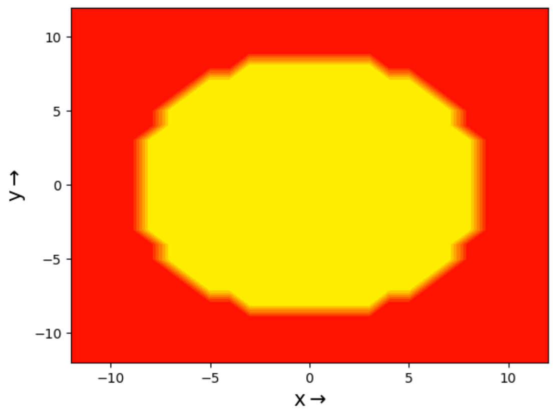

Figure 1: Plot of the initial potential configuration

Error in estimation

The evolution of the error with the number of iterations is

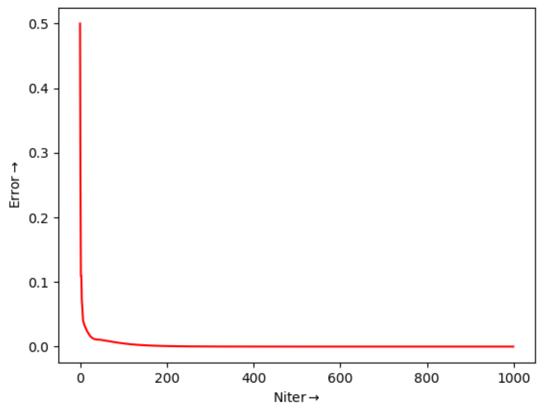

Figure 2: Error vs *Niter*

Using a semilog plot, and considering only every 50th point for the sake of better visualization, we obtain the following plot :

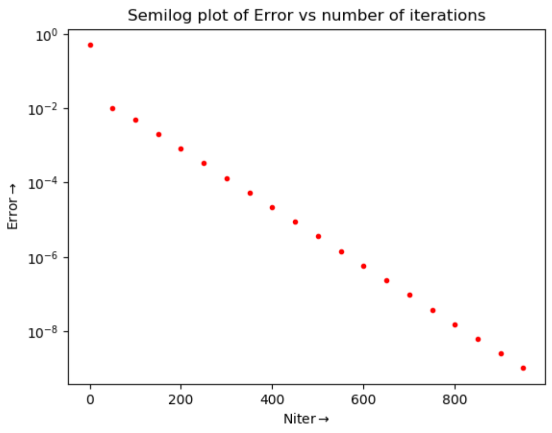

Figure 3: Semilog plot of Error vs number of iterations

Similarly, the following is the loglog plot :

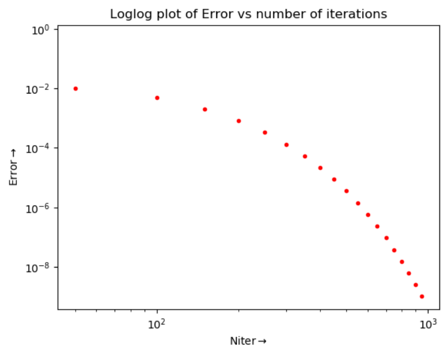

Figure 4: Loglog plot of Error vs number of iterations

The error function appears to vary exponentially with *Niter*. We attempt to extract the exponent of this dependence. In particular, we attempt to fit a function of the form

$$y = Ae^{Bx}$$

Thus ,

$$logy = logA + Bx$$

$logA$ and B can be estimated using the least squares method. The following code block accomplishes the same :

```
def error_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))[0]
    return (np.exp(logA),B)
```

The parameters A and B are obtained in two ways :

- Fitting the entire error vector (*fit1*)
- Fitting only the points beyond 500 iterations(*fit2*)

The fit in both cases when plotted with the original error function yield the following graph :

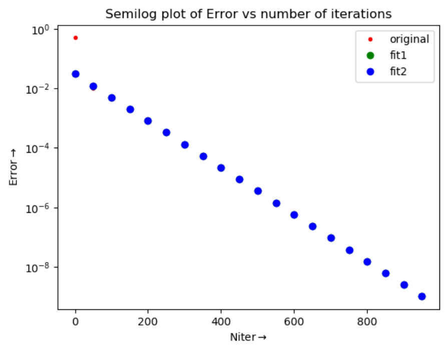

Figure 5: Semilog plot of Error vs number of iterations

Notice that the plots almost exactly coincide, with differences hardly visible.

Stopping condition

The upper bound for the error estimated with each iteration is given by

$$\text{Error} = -\frac{A}{B}exp(B(N+0.5))$$

This can be plotted with the number of iterations *Niter* . The following graph is obtained :

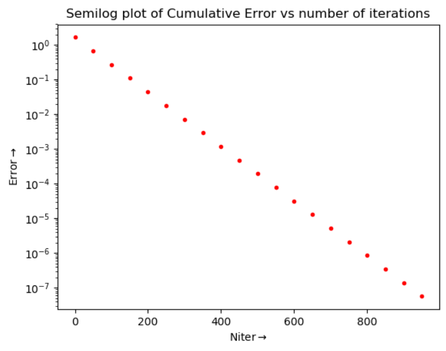

Figure 6:

Surface plot of potential

After running the update for 1000 iterations, we obtain the following 3-D plot for the potential using the `plot_surface` function :

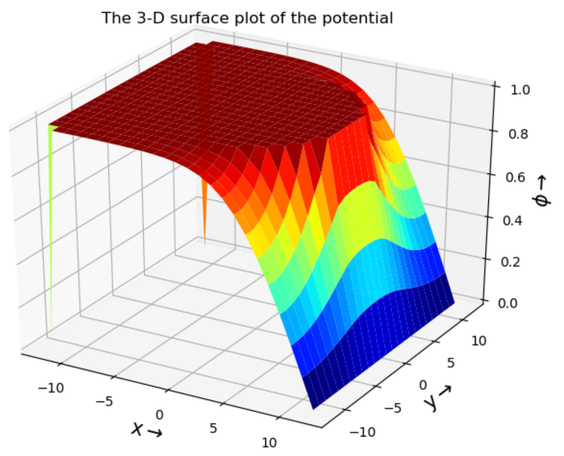

Figure 7: 3-D surface plot of ϕ after *1000* iterations

The contour plot of ϕ is obtained using the `pylab.contour` function.

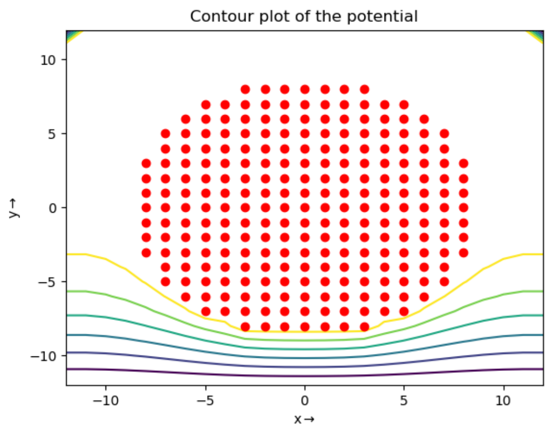

Figure 8: Contour plot of ϕ

Vector plot of Currents

The currents in the system in Cartesian form can be expressed as :

$$J_x = -\frac{\partial \phi}{\partial x}$$

$$J_y = -\frac{\partial \phi}{\partial y}$$

Numerically, this can be expressed as

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

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

The following code block evaluates the currents J_x and J_y :

```
Jx = np.zeros((Ny,Nx))
Jy = np.zeros((Ny,Nx))
Jx[:,1:-1] = 0.5*(potential[:,0:-2]-potential[:,2:])
Jy[1:-1] = 0.5*(potential[2:, :]-potential[0:-2,:])
```

The current density vector is now plotted using the `quiver` function. The following plot is obtained :

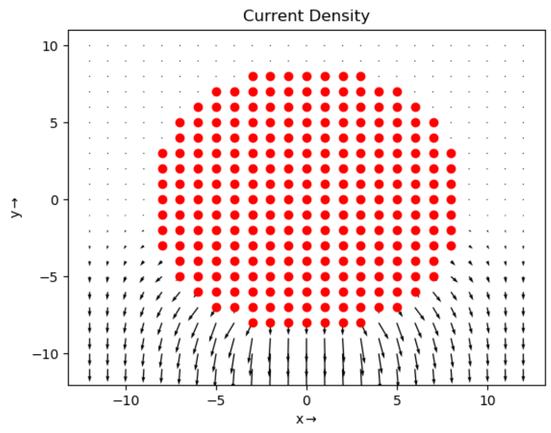

Figure 9: Current density profile

From the current density plot, we notice that hardly any current flows through the top part of the wire. With a little thought, we observe that the lower surface being grounded, the easiest way for charge carriers to flow from the electrode would be directly through the lower half of the wire, thus avoiding a longer, more resistive path through the top half of the wire.

Conclusion

Using a finite differentiation approximation, we have found a solution to Laplace's equation for a given system. The error is seen to decay at a highly gradual pace. Thus the chosen method of solving Laplace's equation is inefficient. On analysing the quiver plot of the currents, it was noticed that the current was mostly restricted to the bottom of the wire, and was perpendicular to the surface of the electrode and the conductor.