Assignment 3

K.R.Srinivas - EE18B136

March 5, 2020

Abstract

This week's Python assignment focuses on the following topics.

- Solving for currents and potential in resistor
- Transforming Laplace equations into a Difference equation.

1 Introduction

The goal of this assignment is to discuss about the solver for the currents in a resistor and discusses about the current's dependency on the shape of the resistor.Here we analyse the currents in a square copper plate to which a wire is soldered to the middle of it.It also discuss about how to find stopping condition for the solver after certain iterations,and to model the errors obtained using Least Squares after analysing the actual errors in semilog and loglog plots.And finally we find the currents in the resistor after applying boundary conditions

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 cm by 1 cm in size.

The Continuity Equation:

$$\nabla.\vec{J} = -\frac{\partial \rho}{\partial t} \tag{1}$$

Charge Continuity equation is used to conserve the inflow and outflow charges.

Conductivity (Differential form of ohm's law):

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

Electric field is the gradient of the potential:

$$\vec{E} = -\nabla \phi \tag{3}$$

Combining the above equations above, assuming that our resistor contains a material of constant conductivity, the equation becomes

$$\nabla^2\phi = \frac{1}{\sigma}\frac{\partial\rho}{\partial t} \tag{4}$$

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

$$\nabla^2\phi = 0 \tag{5}$$

we use a 2-D plate so the Numerical solutions in 2D can be easily transformed into a difference equation. The equation can be written out as

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

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

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

Using above equations we get

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

2 Assignment Tasks

2.1 Defining Parameters and Initialising Potential

We have chosen a $25X25$ grid with a circle of radius 8 centrally located maintained at $V = 1V$ by default.We start by creating an zero $2-D$ array of size $NxXNy$. then a list of coordinates lying within the radius is generated and these points are initialized to 1.

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

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

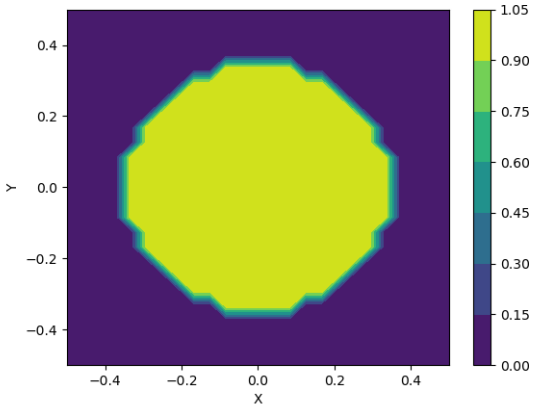

Figure 1: Initial Potential

The contour plot of potential becomes smoother i.e it almost becomes circular as we increase N_x and N_y,because we get more no of points,so the potential gradient is smoothed out between adjacent points since there are more no of points.The contour plot of potential becomes smoother i.e it almost becomes circular as we increase N_x and N_y,because we get more no of points,so the potential gradient is smoothed out between adjacent points since there are more no of points

2.2 Performing Iterations

We update the potential at all points in space using the difference equation given as equation(6). It is necessary to take care of the boundary conditions.At boundaries where the electrode is present, just put the value of electrode potential itself. At boundaries where there is no electrode, the gradient of ϕ should be tangential. This is implemented by requiring that ϕ should not vary in the normal direction.The central portion is more difficult. During an iteration, the boundary values of the electrode can get over-written. So we have to recreate those values.

```
#function to update potential
  def phi_new(phi,phiold):
      phi[1:-1,1:-1]=0.25*(phiold[1:-1,0:-2]+phiold[1:-1,2:
                                ]+ phiold[0:-2,1:-1] +phiold[2:,1:-1])
```

```
        return phi

# function to update potential at boundaries
 def boundary(phi,central_portion = np.where(X**2+Y**2<(0.35)**2)):
 phi[:,0] = phi[:,1] # Left Boundary
 phi[:,Nx-1] = phi[:,Nx-2] # Right Boundary
 phi[0,:] = phi[1,:] # Top Boundary
 phi[Ny-1,:] = 0 # Bottom boundary connected to ground
 phi[central_portion] = 1.0 #recreating boundary values at electrode
 return phi

err = np.zeros(Niter)# initialise error array

for k in range(Niter):
        phiold = phi.copy()# copy the old phi
        phi = phi_new(phi,phiold)# Updating the potential
        phi = boundary(phi)# applying boundary conditions
        err[k] = np.max(np.abs(phi-phiold))# Appending errors for each
                                            iterations
        if(err[k] == 0):
            print("Reached steady state at ",k," Iterations")
            break
```

We will plot the errors on semi-log and log-log plots.

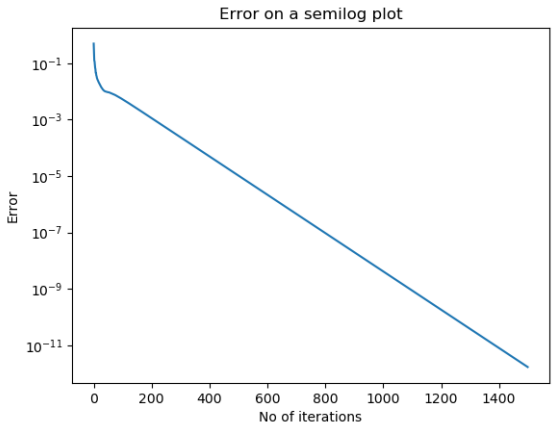

Figure 2:

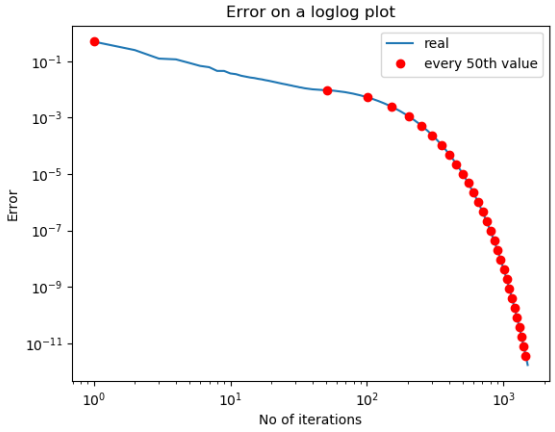

Figure 3:

error decreases linearly for higher no of iterations,so from this we conclude that for large iterations error decreases exponentially with No of iterations i.e it follows Ae^{Bx}

2.3 Error Fitting

We find the fit using Least squares for all iterations named as **fit1**and for iterations ≥ 500 named as **fit2** separately and compare them.As we know that error follows Ae^{Bx} at large iterations, we use equation given below to fit the errors using least squares.

$$logy = logA + Bx \tag{7}$$

```
def best_fit(y,Niter,lastn=0):
    log_err = np.log(err)[-lastn:]
    X = np.vstack([(np.arange(Niter)+1)[-lastn:],np.ones(log_err.shape)])
    log_err = np.reshape(log_err,(1,log_err.shape[0])).T
    return s.lstsq(X, log_err)[0]

def plot_error(err,Niter,a,a_,b,b_):
    title("Best fit for error on a loglog scale")
    xlabel("No of iterations")
    ylabel("Error")
    x = np.asarray(range(Niter))+1
    loglog(x,err)
    loglog(x[::100],np.exp(a+b*np.asarray(range(Niter)))[::100],'ro')
    loglog(x[::100],np.exp(a_+b_*np.asarray(range(Niter)))[::100],'go')
    legend(["errors","fit1","fit2"])
```

```
show()

title("Best fit for error on a semilog scale")
xlabel("No of iterations")
ylabel("Error")
semilogy(x,err)
semilogy(x[::100],np.exp(a+b*np.asarray(range(Niter)))[::100],'ro')
semilogy(x[::100],np.exp(a_+b_*np.asarray(range(Niter)))[::100],'go')
legend(["errors","fit1","fit2"])
show()
```

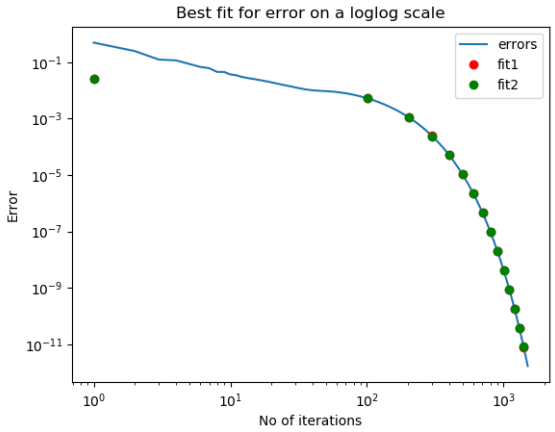

Figure 4:

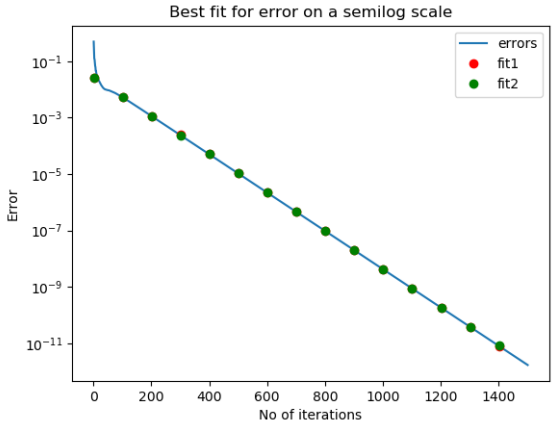

Figure 5:

2.4 Plotting Cumulative Error

To find the cumulative error, we add all the absolute values of errors for each iteration since worst case is, all errors add up. So we use the equations given below:

$$Error = \sum_{N+1}^{\infty} error_k \tag{8}$$

The above error is approximated to

$$Error \approx -\frac{A}{B} exp(B(N + 0.5)) \tag{9}$$

where N is no of iteration

```
def find_net_error(a,b,Niter):
    return -a/b*np.exp(b*(Niter+0.5))
iter=np.arange(100,1501,100)
grid(True)
title(r'Plot of Cumulative Error values On a loglog scale')
loglog(iter,np.abs(find_net_error(a_,b_,iter)),'ro')
xlabel("iterations")
ylabel("Net maximum error")
show()
```

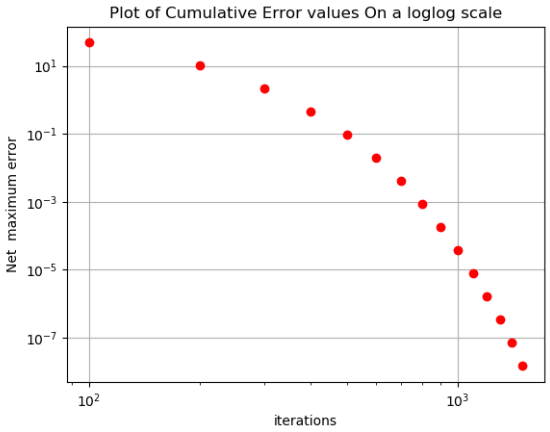

Figure 6:

2.5 Surface and Contour Plot of the Potential

In this section we generate a 3-D surface Plot and Contour plot of the potential and analyse them. The code plot them is given below.

```
#plotting 3d contour of final potential
fig1=plt.figure(4)      # open a new figure
ax=p3.Axes3D(fig1)      # Axes3D is the means to do a surface plot
title('The 3-D surface plot of the potential')
surf = ax.plot_surface(Y, X, phi.T, rstride=1, cstride=1,
                       cmap=plt.cm.jet)
fig1.colorbar(surf, shrink=0.5, aspect=5)
show()

#plotting 2d contour of final potential
title("2D Contour plot of potential")
xlabel("X")
ylabel("Y")
x_c,y_c=np.where(X**2+Y**2<(0.35)**2)
plot((x_c-Nx/2)/Nx,(y_c-Ny/2)/Ny,'ro')
contourf(Y,X[::-1],phi)
colorbar()
show()
```

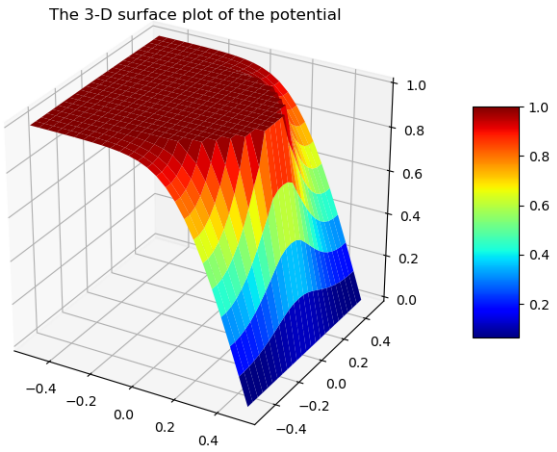

Figure 7:

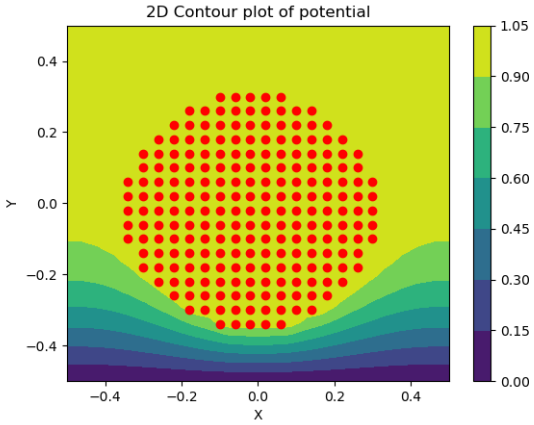

Figure 8:

We observe that the surface plot we conclude that after updating the potential,the potential gradient is higher in down part of the plate since, the down side is grounded and the electrode is at 1 V,so there is high potential gradient from electrode to grounded plate.

Same observation we see using contour plot in 2 dimensions, we note that there are gradients in down part of the plate and almost negligible gradient in upper part of the plate.

2.6 Vector Plot of Currents

In order to obtain the currents by computing gradients, we usethe following equations and since actual values of conductivity doesn't matter we assume it to be 1.

$$J_{x,ij} = \frac{1}{2}(\phi_{i,j-1} - \phi_{i,j+1}) \tag{10}$$

$$J_{y,ij} = \frac{1}{2}(\phi_{i-1,j} - \phi_{i+1,j}) \tag{11}$$

```
Jx = (1/2*(phi[1:-1,0:-2]-phi[1:-1,2:]))
Jy = (1/2*(phi[:-2,1:-1]-phi[2:,1:-1]))

title("Vector plot of current flow")
quiver(Y[1:-1,1:-1],-X[1:-1,1:-1],-Jx[:,::-1],-Jy)
x_c,y_c=np.where(X**2+Y**2<(0.35)**2)
plot((x_c-Nx/2)/Nx,(y_c-Ny/2)/Ny,'ro')
show()
```

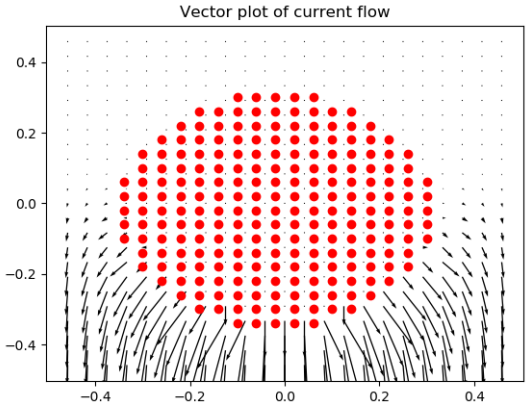

Figure 9:

There is almost zero current in upper part of the plate since there is not much potential gradient as we observed from the surface and contour plot of the potential,whereas most of the current is in the narrow region at the bottom due to greater potential gradient.So that is what will get strongly heated. ϕ

3 Conclusion

We have used discrete differentiation to solve Laplace's Equations.This method of solving Laplace's Equation is known to be one of the worst available. This is because of the very slow coefficient with which the error reduces.We looked the modelling of the currents in resistor in this report ,and we observe that the best method to solve this is to increase N_x and N_y to very high values(100 or $\geq$ 100)and increase the no of iterations too, so that we get accurate answers i.e currents in the resistor.But the tradeoff is this method of solving is very slow even though we use vectorized code because the decrease in errors is very slow w.r.t no of iterations