EE2703 : Applied Programming Lab
Assignment 3

Andapally Snehitha
EE20B008

March 11, 2022

Abstract

This week we wish to solve for the currents in a resistor. The currents depend on the shape of the resistor. We also want to know which part of the resistor is likely to get hottest.

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 cm by 1 cm in size.
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:

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

Assignment 5

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

```
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])
    print("Using user provided params")
else:
    Nx=25 # size along x
    Ny=25 # size along y
```

```
    radius=8 #radius of central lead
    Niter=1500 #number of iterations to perform
    print("Using default Parameters")
```

Initializing Potential

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

```
phi=np.zeros((Nx,Ny),dtype = float)
x,y=np.linspace(-0.5,0.5,num=Nx,dtype=float),np.linspace
    (-0.5,0.5,num=Ny,dtype=float)
Y,X=np.meshgrid(y,x,sparse=False)
phi[np.where(X**2+Y**2<(0.35)**2)]=1.0
plt.xlabel("X")
plt.ylabel("Y")
plt.contourf(X,Y,phi)
plt.colorbar()
plt.show()
```

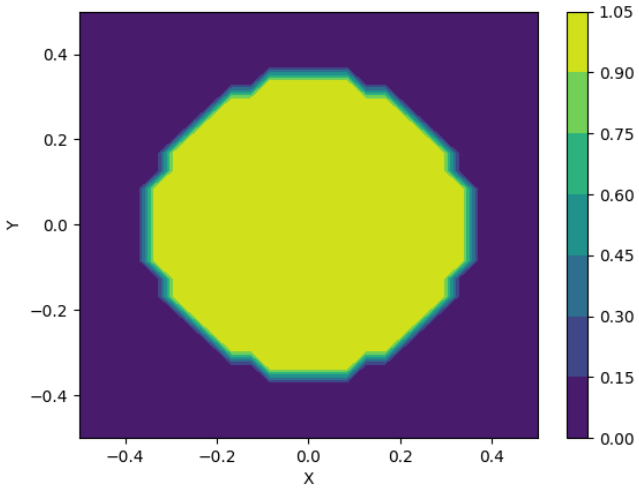

Figure 1: Initial Potential

Performing Iterations

Updating Potential

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. We write it as :

$$\phi_{i,j} = 0.25 * (\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j+1} + \phi_{i,j-1}) \tag{5}$$

```
def update_phi(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
```

Applying Boundary Conditions

The bottom boundary is grounded. The other 3 boundaries have a normal potential of 0

```
def boundary(phi,mask = np.where(X**2+Y**2<(0.35)**2)):
    phi[1:-1,0]=phi[1:-1,1] # Left Boundary
    phi[1:-1,Nx-1]=phi[1:-1,Nx-2] # Right Boundary
    phi[0,1:-1]=phi[1,1:-1] # Top Boundary
    phi[Ny-1,1:-1]=0
    phi[mask]=1.0
    return phi
```

Calculating error and running iterations

```
for k in range(Niter):
    phiold = phi.copy()
    phi = update_phi(phi,phiold)
    phi = boundary(phi)
    err[k] = np.max(np.abs(phi-phiold))
```

Plotting the errors

We will plot the errors on semi-log and log-log plots. We note that the error falls really slowly and this is one of the reasons why this method of solving the Laplace equation is discouraged

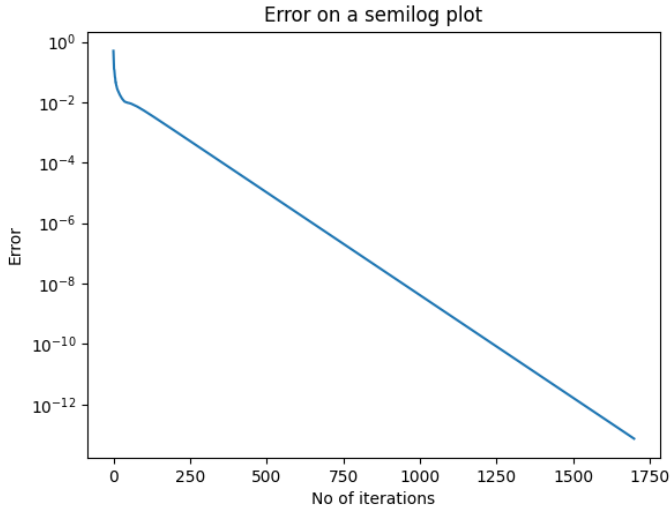

Figure 2: Semi-log plot of error

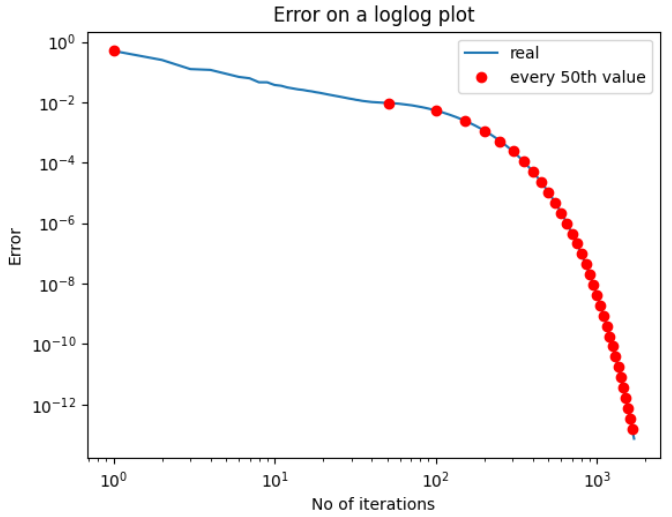

Figure 3: Log-log plot of error

Fitting the error

We note that the error is decaying exponentially for higher iterations.I have plotted 2 fits. One considering all the iterations(fit1) and another without considering the first 500 iterations. There is very little difference between the two fits.

```
1
2  def get_fit(y,Niter,lastn=0):
3      log_err = np.log(err)[-lastn:]
4      X = np.vstack([(np.arange(Niter)+1)[-lastn:],np.ones(
       log_err.shape)]).T
5      log_err = np.reshape(log_err,(1,log_err.shape[0])).T
6      return s.lstsq(X, log_err)[0]
7
8
9  def plot_error(err,Niter,a,a_,b,b_):
10     plt.title("Best fit for error on a loglog scale")
11     plt.xlabel("No of iterations")
12     plt.ylabel("Error")
13     x = np.asarray(range(Niter))+1
14     plt.loglog(x,err)
15     plt.loglog(x[::100],np.exp(a+b*np.asarray(range(Niter)))
       [::100],'ro')
16     plt.loglog(x[::100],np.exp(a_+b_*np.asarray(range(Niter))
       )[::100],'go')
17     plt.legend(["errors","fit1","fit2"])
18     plt.show()
19
20 b,a = get_fit(err,Niter)
21 b_,a_ = get_fit(err,Niter,500)
22 plot_error(err,Niter,a,a_,b,b_)
```

Plotting Maximum Possible Error

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.

```
1  def find_net_error(a,b,Niter):
2      return -a/b*np.exp(b*(Niter+0.5))
3  b,a = get_fit(err,Niter)
4  b_,a_ = get_fit(err,Niter,500)
5  plot_error(err,Niter,a,a_,b,b_)
6  iter=np.arange(100,1501,100)
7  plt.grid(True)
8  plt.title(r'Plot of Cumulative Error values On a loglog scale
       ')
```

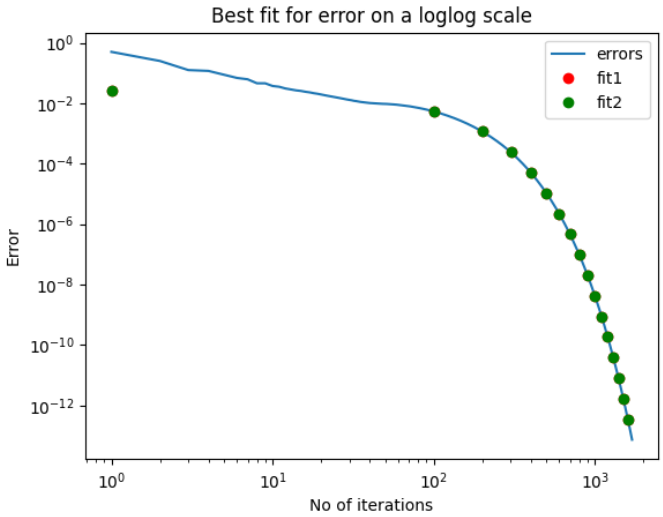

Figure 4: Best Fit of error

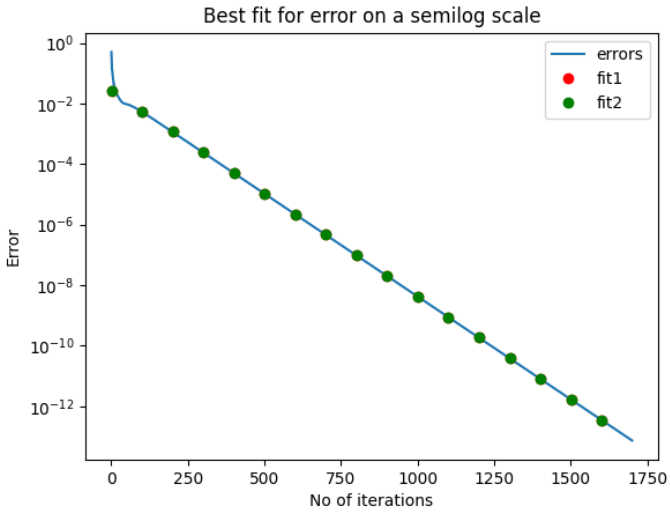

Figure 5: Best Fit of error

```
plt.loglog(iter,np.abs(find_net_error(a_,b_,iter)),'ro')
plt.xlabel("iterations")
plt.ylabel("Net  maximum  error")
plt.show()
```

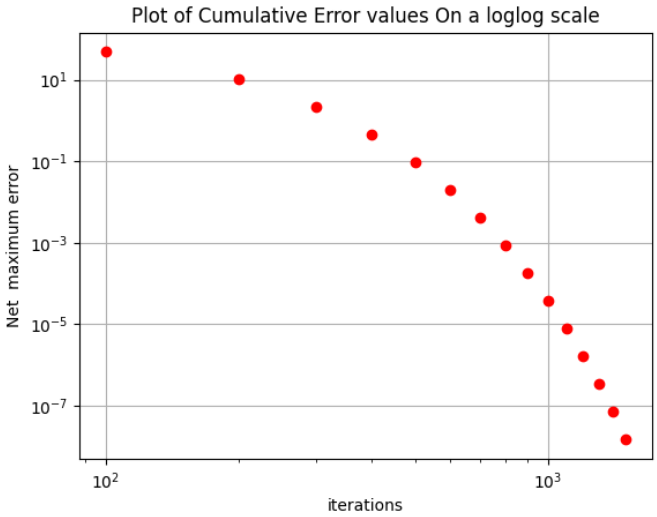

Figure 6: Cumulative error values on a log log scale

Plotting ϕ

```
fig1=plt.figure(4)      # open a new figure
ax=p3.Axes3D(fig1) # Axes3D is the means to do a surface plot
plt.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)
plt.show()

plt.title("2D Contour plot of potential")
plt.xlabel("X")
plt.ylabel("Y")
plt.contourf(Y,X[::-1],phi)
plt.colorbar()
plt.show()
```

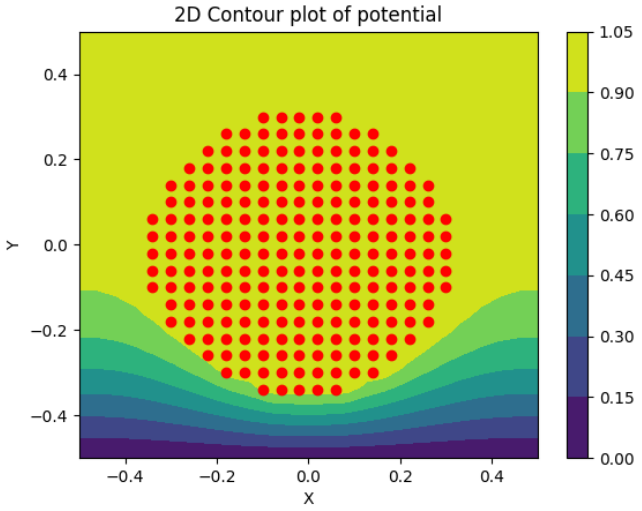

Figure 7: 2d Plot of Potential

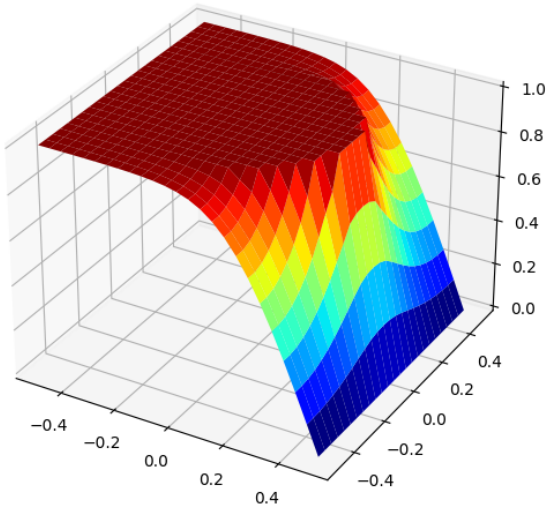

Figure 8: 3d Plot of Potential

Finding and Plotting J

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

$$J_{y,ij} = 0.5 * (\phi_{i-1,j} - \phi_{i+1,j}) \tag{7}$$

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

plt.title("Vector plot of current flow")
plt.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)
plt.plot((x_c-Nx/2)/Nx,(y_c-Ny/2)/Ny,'ro')
plt.show()
```

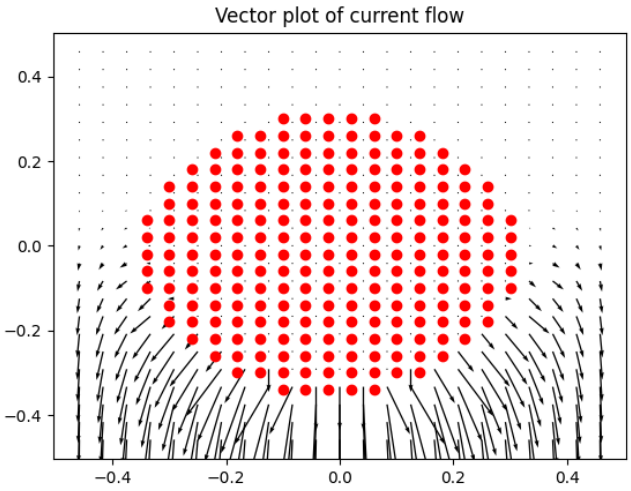

Figure 9: Vector plot of current flow

Finding temperature at different regions in the plate

We solve the Laplace equation for heat flow:

$$\nabla^2 T = -\frac{q}{\kappa} = -\frac{|J|^2}{\sigma\kappa}$$

We again assume the electrical and thermal conductivites of the plate are 1. We calculate the heat generated below:

```
#initialize potential
temp=300 * np.ones((Nx,Ny),dtype = float)
tempold= np.zeros((Nx,Ny),dtype = float)


def temper(phi,mask = 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,:]=300.0
    phi[mask]=300.0
    return phi
def tempdef(temp,oldtemp,Jx,Jy):
    temp[1:-1,1:-1]=0.25*(tempold[1:-1,0:-2]+ tempold
   [1:-1,2:]+ tempold[0:-2,1:-1] + tempold[2:,1:-1]+(Jx)**2
   +(Jy)**2)
    return temp

#the iterations
for k in range(Niter):
    tempold = temp.copy()
    temp = tempdef(temp,tempold,Jx,Jy)
    temp = temper(temp)


#plotting 2d contour of final potential
plt.title("2D Contour plot of temperature")
plt.xlabel("X")
plt.ylabel("Y")
plt.contourf(Y,X[::-1],temp)
plt.colorbar()
plt.show()
```

This is very similar to the previous case of finding potential except that at initial condition, everything is at 300K, and the boundary conditions set the central circle and ground to 300K for all iterations.

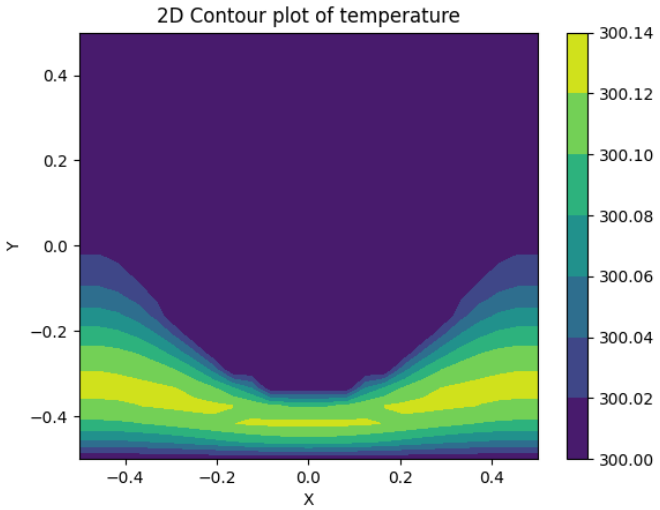

Figure 10: Vector plot of current flow

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 rate with which the error decays. The error decays exponentially.As the current magnitudes were relatively higher in the bottom half of the plate,the relative heating in those regions was higher