# EE2703 : Applied Programming Lab
# Assignment 5
# Report

Kesava Aruna Prakash R L
EE20B061

March 8, 2022

## Abstract

This week's assignment focuses on solving electric potential and current density for a resistor of given dimensions and for a certain conditions on the applied external potentials. The given resistor is shaped like a square with the dimensions of 1cm $*$ 1cm with three of it's sides disconnected from the surroundings, while one of it's sides grounded. A circular region of given radius at the centre of the resistor is held at 1V. The resultant distribution of potential and current density are calculated using the equations:

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

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

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

## 1 Setting up $\phi$

For this analysis, $\phi$ is taken in the form of a matrix of discrete points spread across the resistor. The parameters affecting the $\phi$ are given as:

```
Nx = 25 # No of points in x-axis
Ny = 25 # No of points in y-axis
radius = 8 # Radius of the centre region
N_iteration = 1500 # Number of iterations for which phi is recalculated
```

First, the centre circular region of the resistor is set at 1V using the help of `meshgrid()` and `where()`. Those points are marked with a red crosses in the contour plot. The python code executing the process:

```
# Creating matrix for phi
phi = np.zeros([Nx,Ny])

# Since we know the plate is 1cm x 1cm, we can split the sides into Nx and Ny po
x = np.linspace(-0.5,0.5,Nx)
y = np.linspace(-0.5,0.5,Ny)
#y = np.flip(y)

# Forming the meshgrid
Y,X = np.meshgrid(y,x)
```

```
# Conditions for checking
X1,Y1=np.where(Y*Y+X*X<=(radius/(Nx-1)+0.01)**2)
for i in range(len(X1)):
    phi[X1[i]][Y1[i]]=1
```

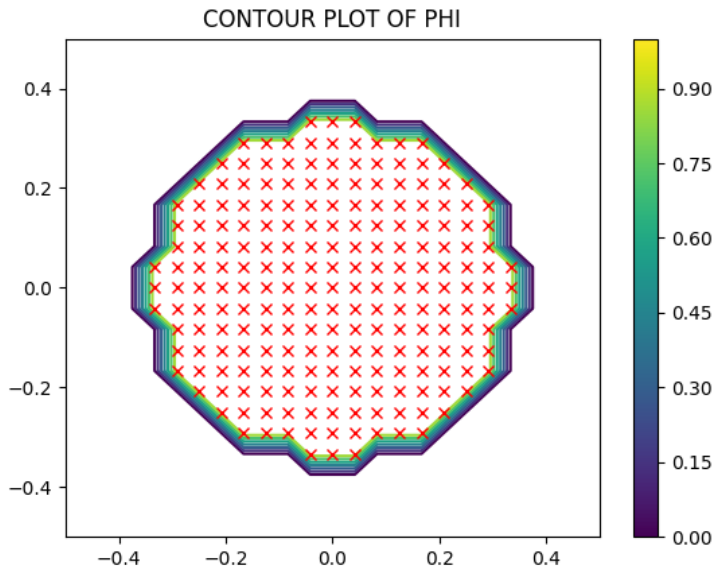


Figure 1: Contour plot of $\phi$

## 2 Updation of $\phi$

The equation obtained on solving $\nabla^2\phi = 0$ is :

$$\phi[i,j] = \frac{1}{4}(\phi[i-1,j] + \phi[i+1,j] + \phi[i,j-1] + \phi[i,j+1])$$

We can update the values of $\phi$ matrix in each iteration with suitable boundary conditions to get the approximate solution of the equation.
Python code for the above:

```
# Iterating for N_iteration times and recalculating phi at each point
# in the iteration
for i in range(N_iteration):
   oldphi=phi.copy()
   phi[1:-1,1:-1]=0.25*(phi[1:-1,2:]+phi[1:-1,0:-2]+phi[0:-2,1:-1]+phi[2:,1:-1])
```

```
# Boundary conditions
phi[1:-1,0]=phi[1:-1,1]
phi[1:-1,-1]=phi[1:-1,-2]
phi[-1,:]=phi[-2,:]

# Setting phi as 1V in the centre region
for j in range(len(X1)):
    phi[X1[j]][Y1[j]]=1

# Calculating errors
errors[i]=(abs(phi-oldphi)).max()
```

Error in each iteration is calculated and stored in an array.

## 3 Errors

The errors obtained from each iteration of each update of $\phi$ is plotted:

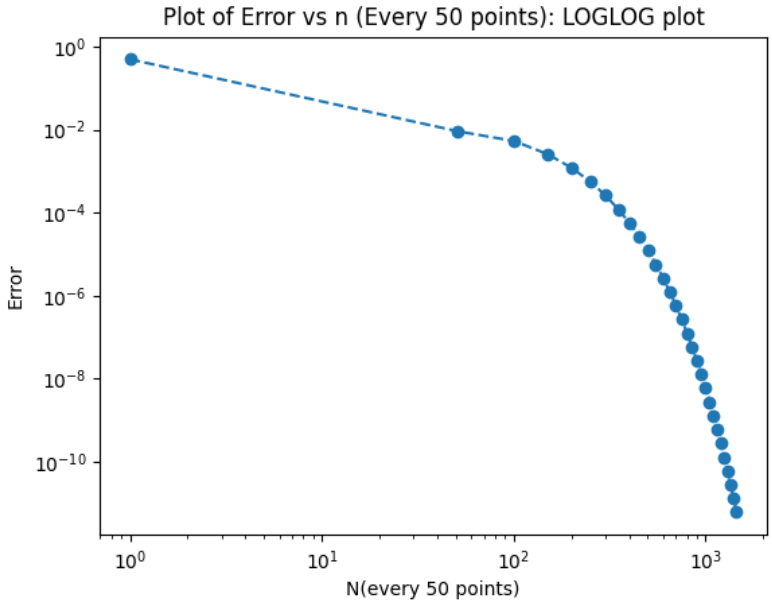


Figure 2: Plot of Error(every 50 points)- Loglog plot

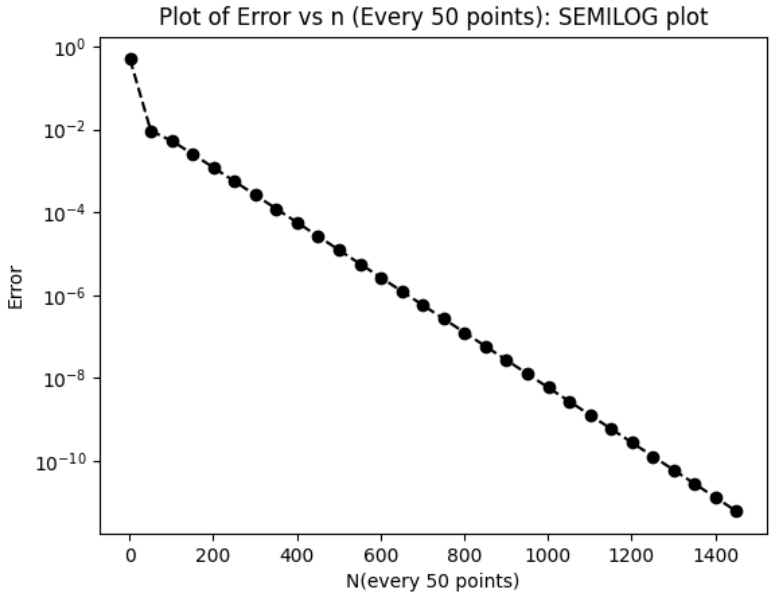


Figure 3: Plot of Error(every 50 points)- Semilog plot

These plots suggest that the error is decaying exponentially in higher iterations.

## 4 Obtaining linear fits to the errors

The fit is obtained for two different data samples. First sample comprises of all the iterations while the second sample consists of only larger iterations. The error is approximated to a exponential function $A * e^{B}$ and the best fit of $A$ and $B$ are calculated using the least squares method.
textttlinalg.lstsq() is used to obtain the best fit.
Python code for the above:

```
# Finding best fit for all values (fit1)

y_fit = np.log(errors)
x_fit = np.c_[n.T,n1.T]
B,logA = np.linalg.lstsq(x_fit,y_fit,rcond=None)[0]
A=np.exp(logA)

# Finding best fit for N= 500 to N_iterations (fit2)
```

```
y_fit1 = np.log(errors[500:])
x_fit1 = np.c_[n[500:].T,n1[500:].T]
#print(x_fit1)
B1,logA1 = np.linalg.lstsq(x_fit1,y_fit1,rcond=None)[0]
A1=np.exp(logA1)
```

The fits and actual error are plotted.

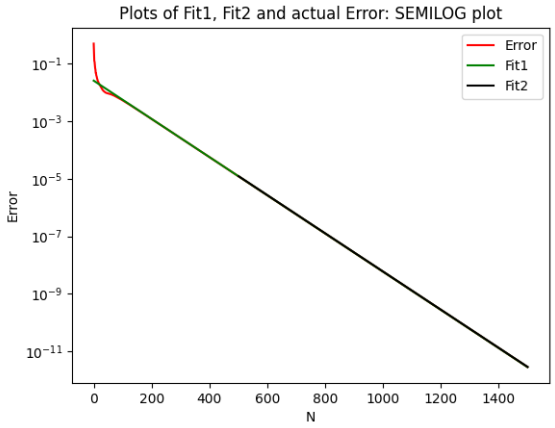


Figure 4: Plot of fits and actual error

The cumulative error is also determined and plotted.

$$Cumulative\ error\ = -\frac{A}{B}exp(B(N+0.5))$$

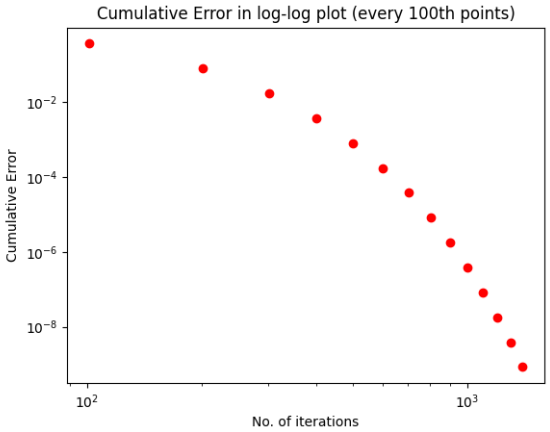


Figure 5: Cumulative error

## 5 Contour and 3D plots of $\phi$

$\phi$ after recalculation is plotted:

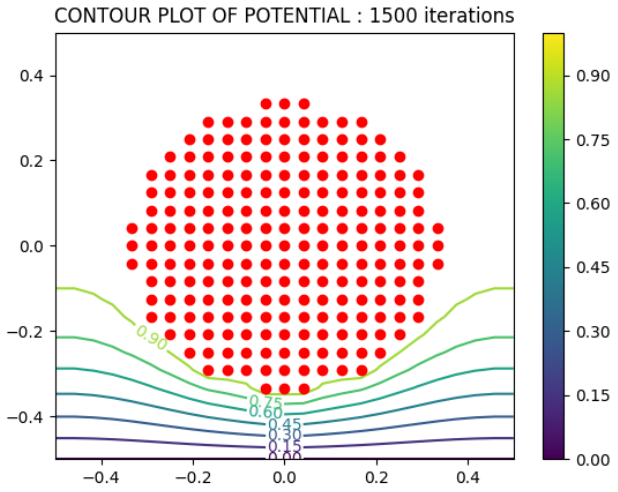


Figure 6: Contour plot of $\phi$

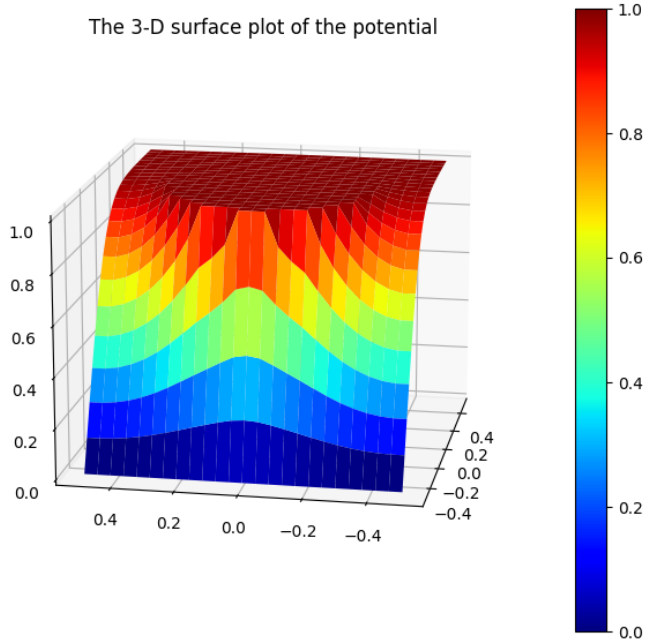


Figure 7: Surface plot of $\phi$

Python code for plotting the above:

```
pt.title(f"CONTOUR PLOT OF POTENTIAL : {N_iteration} iterations")
cntplot=pt.contour(x,y,phi)
pt.clabel(cntplot)
norm1= matplotlib.colors.Normalize(vmin=0, vmax=1)
sm1 = pt.cm.ScalarMappable(norm=norm1, cmap = cntplot.cmap)
sm1.set_array([])
pt.colorbar(sm1, ticks=cntplot.levels)
pt.plot(x[X1],y[Y1],'ro')
pt.show()


fig1=pt.figure()
ax=p3.Axes3D(fig1,auto_add_to_figure=False)
ax.set_title("The 3-D surface plot of the potential")
fig1.add_axes(ax)
#pt.title('The 3-D surface plot of the potential')
surf = ax.plot_surface(Y, X, phi.T, rstride=1, cstride=1, cmap=pt.cm.jet)
norm2= matplotlib.colors.Normalize(vmin=0, vmax=1)
sm2 = pt.cm.ScalarMappable(norm=norm2, cmap = surf.cmap)
sm2.set_array([])
pt.colorbar(sm2,shrink=0.9)
fig1.set_size_inches(6, 6)
pt.show()
```

## 6 Surface currents

Equations for current densities when converted to the discrete values change to the following:

$$J_x = -\frac{1}{2}(\phi[i+1,j] - \phi[i-1,j])$$

$$J_y = -\frac{1}{2}(\phi[i,j+1] - \phi[i,j-1])$$

Python code for finding and plotting $j_x$ and $j_y$:

```
Jx = 0.5*(phi[1:-1,0:-2]-phi[1:-1,2:])
Jy = 0.5*(phi[0:-2,1:-1]-phi[2:,1:-1])
pt.title("CURRENT DENSITY PLOT")
pt.quiver(x[1:-1],y[1:-1],Jx,Jy)
pt.plot(x[X1],y[Y1],'ro')
```

Plot of current density:

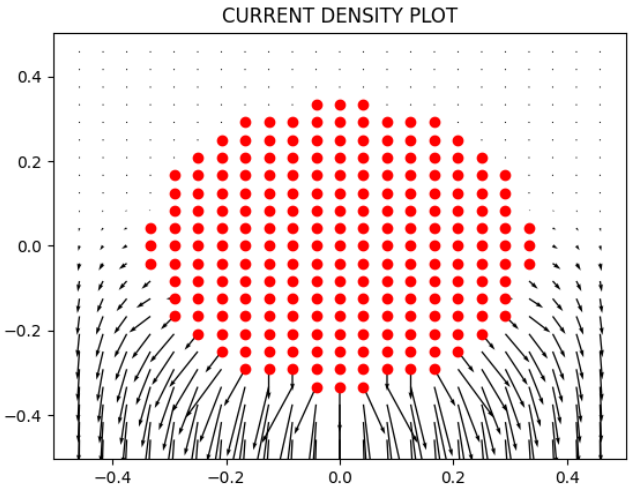


Figure 8: Plot of $j_x$ and $j_y$

## 7 Conclusion and Inference

- We have simplified Laplace equation for electric potential for our convenience.
- Taking more number of points in the matrix for $\phi$ will result in a more accurate calculation.
- The decrease in error in higher iterations is very low. The same can be observed in the plot of cumulative error.
- The vector plot of current density gives a good picture on the current flow directions.
- Most of the current flow occurs from the centre region to the side which is grounded.
- Based on the vector plot, we can say that the area near the side of the square which is grounded, gets heated the most.