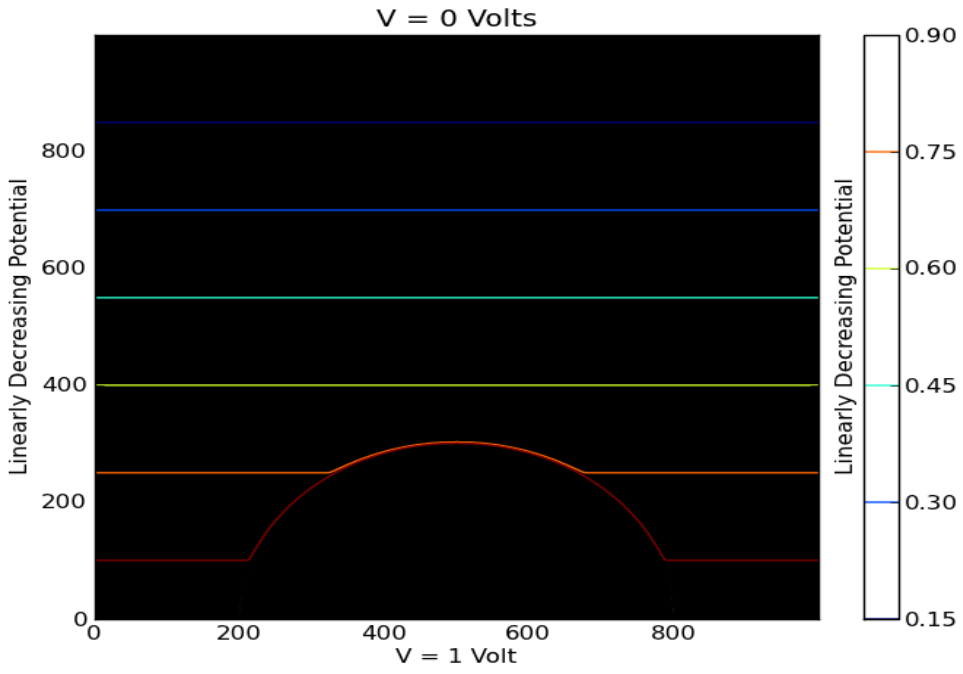
The Behavior of Electric Field Vectors and Potential lines Inside a Closed Surface

Saeed Mohammad

Abstract

We use a programming language called Python in order to plot and calculate both potential lines and electric field vectors inside a closed surface with specific boundary conditions. We use the knowledge we have about electrostatics and employ Poisson's equation for achieving our purpose. We will also examine how the lines behave after introducing a semicircular conducting surface inside our closed surface. The final figure we obtain is



Objective

To use Python in order to code a program that calculates and plots the electric potential and electric field at every point inside a closed surface.

Procedure and Discussion

We first set up a potential difference across a closed surface, which is a square in this case. We then fix the top boundary to 0 volt, the bottom boundary to 1 volt, and set the right and left boundaries to linearly decreasing potentials; therefore, we obtain a potential difference of 1 volt. Next, after setting up arbitrary values for the x- and y- axes, we command Python to calculate the potential at every point inside the square in the following manner:

"In order to find the potential at any point (x , y) inside the square, find the average value of the four neighboring points, which in 2-D, are the right, left, top, and bottom closest points. Iterate this definition as many times as commanded. Plot the contour lines of the founded values of potential. Show a reference colorbar."

This finds and plots the potential at every point inside the square, which looks like the following:

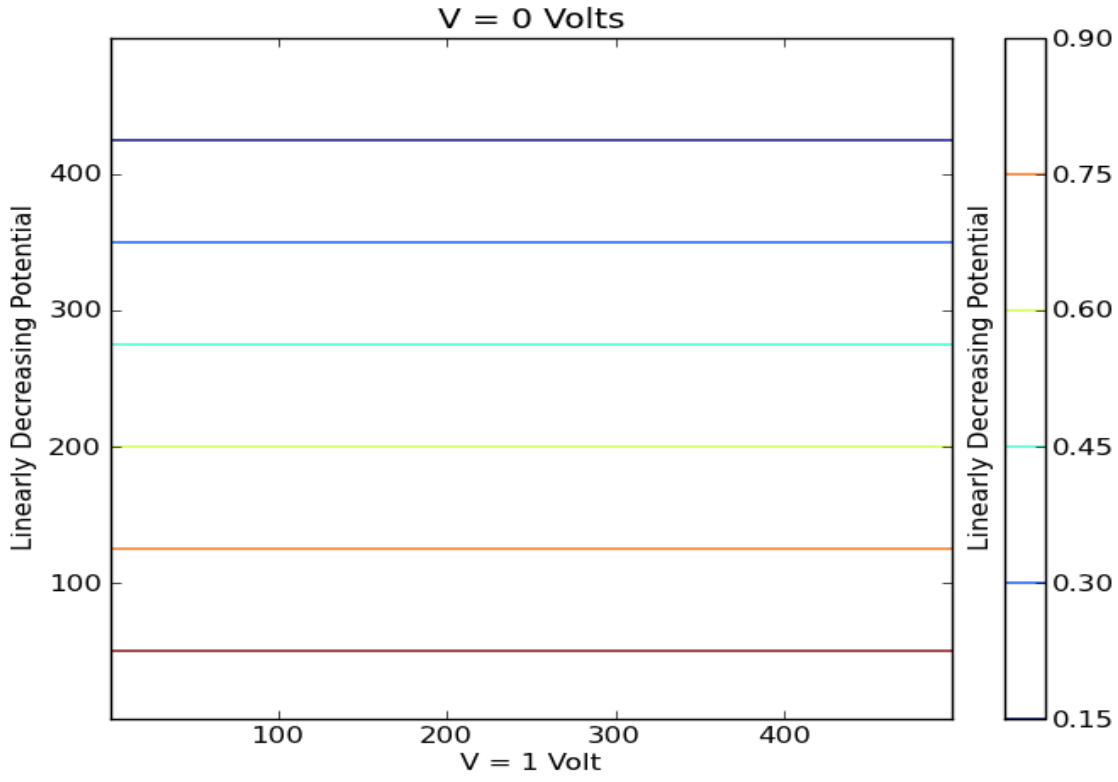


Figure 1—Electric Potential Lines

Next, we command Python to find and plot the 'negative gradient' of the potential, which is electric field at every point inside the square. We obtain the following:

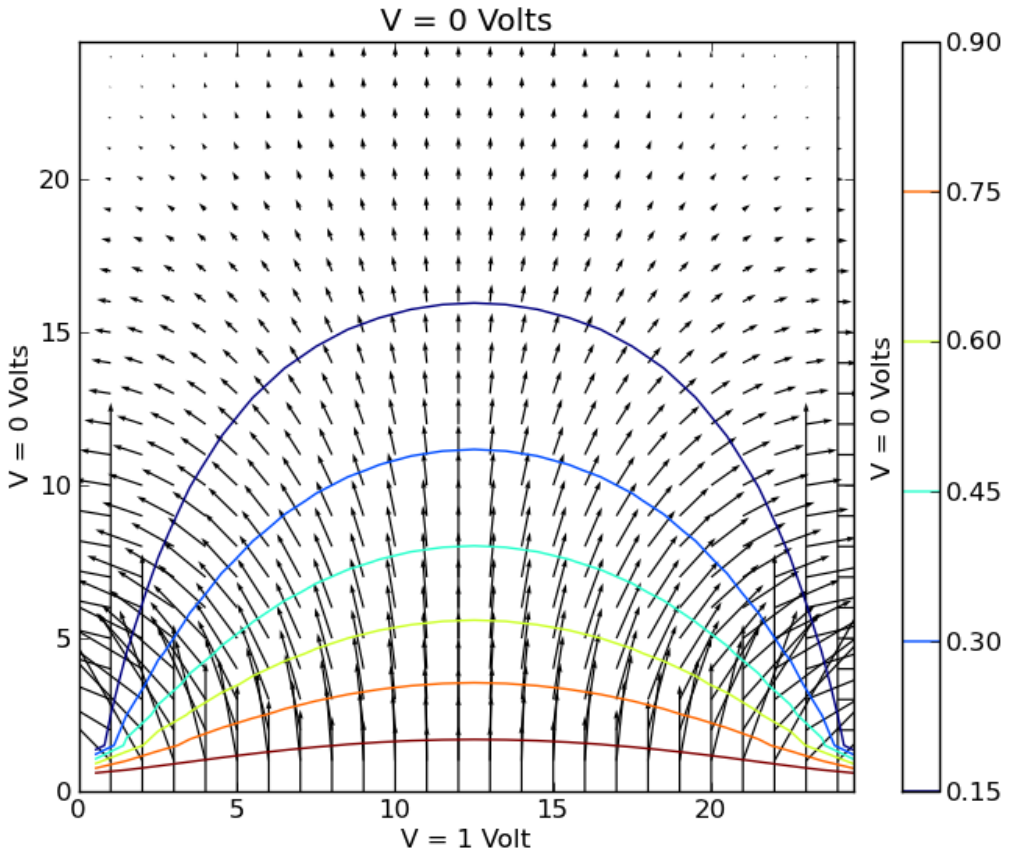
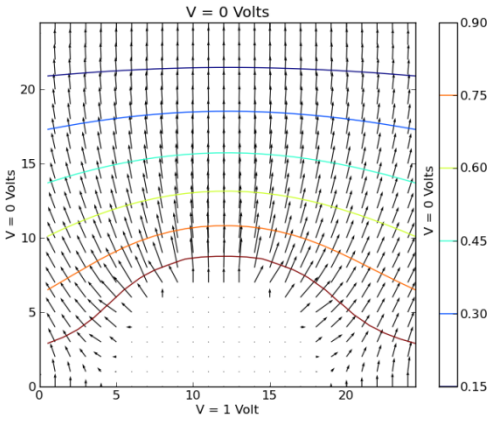


Figure 2—Electric Potential and Electric Field Lines

As can be seen from the figure, the electric field lines are perpendicular to the potential lines (as expected) and are 'flowing' towards a decreasing potential direction. Also, the electric field lines are more packed at the edges of the square, meaning that the electric field is stronger (has a higher value) at the bottom corners of the square (also as predicted).

Finally, we test our coding by visualizing the situation of a hill in a thunderstorm, how would the electric field lines and the potential lines behave in such a situation? To find out, we place a conducting semicircular surface of an arbitrary radius at the bottom of the square and observe how the potential lines and electric field lines behave. Also, the left and right boundaries are linearly decreasing potential functions in order to minimize the fringe effects at the edges of the closed square. The following figures are what we got:



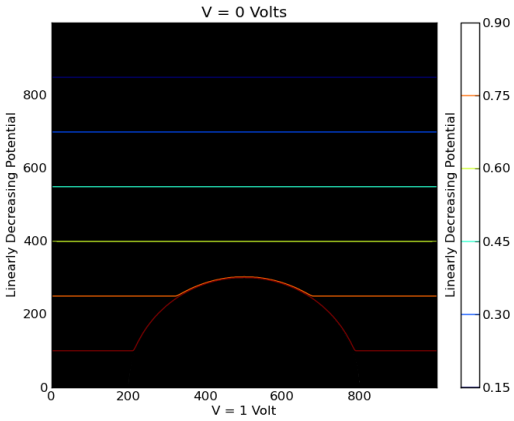


Figure 3a—A small square with few iterations

Figure 3b—A large square with many iterations

These figures make sense. No electric field lines pass inside the semi-circle (otherwise, current would flow in the semi-circle until no electric field lines remain). Also, the electric field is stronger when we move closer to the surface of the semicircle and decays as we go further.

Derivation

The physics that explains the figures and the lines' behaviors is as follows:

Gauss' law $E \cdot dA = Q_{enc} \varepsilon_0$ is equivalent to (by the divergence theorem) $\nabla \cdot E dV$

This is also equivalent to $\rho \varepsilon_0 dV$ where ρ is charge density.

From $\nabla \cdot E dV = \rho \varepsilon_0 dV$ we deduce that for this equation to hold true, this $\nabla \cdot E = \rho \varepsilon_0$ must be true.

We learned in electromagnetism that $E = -\nabla V$

Putting everything together, we get $-\nabla 2\, V = \rho \varepsilon_0$ which is Poisson's equation.

Now, because we are finding the potential in space, where the charge density is zero, Poisson's equation simplifies into $-\nabla 2\, V = 0$ which is Laplace's equation and means that the second partial derivative of the potential is equal to zero (in 2-D, $-\partial 2V \partial x2 - \partial 2V \partial y2 = 0$).

Errors:

- The number of iterations could be increased, which would make the lines look smoother.
- The area of the square could be increased as well; this would reduce the rigidness of the lines and spread them out more evenly.
- The fringe effects at the boundaries of the square might have slightly disfigured some of the lines.

Coding Method:

Note: Sentences beginning with # are comments for clarification; they do not affect the coding process. Italics sentences *explain* and bold sentences **command.**

 # numpy is a numerical array library for Python

import numpy as np

 # matplotlib.pyplot is a command style function that allows the user to plot and edit graphs

import matplotlib.pyplot as plt

 # math allows the user to use mathematical expressions such as sin, pi, etc...

import math

 # Set the x- and y- axes to a certain value. Define dx and dy.

NX = 1000

NY = 1000

dx = 1./NX; dy = 1./NY

 # Set the radius of the protruding ridge.

RADIUS = 0.3

 # Calculate the number of rows in the ridge.

nrow = int(RADIUS*NY)

print nrow

 # Define the width and size of the conducting semicircle.

ridgeWidth = np.zeros(nrow)

 # Create a loop for iteration.

```
for j in range(nrow):
   ridgeWidth[j] = int(NX*math.sqrt(RADIUS**2 - ((j+1)*dy)**2 +1e-6)+0.5)
print ridgeWidth
#Add a subplot and label each of the sides
fig = plt.figure()
v = fig.add_subplot(111)
v.set_ylabel('Linearly Decreasing Potential')
v.set_xlabel('V = 1 Volt')
v.set_title('V = 0 Volts')
v.text(1025,500,'Linearly Decreasing Potential',
      horizontalalignment='left',
      verticalalignment='center',
      rotation = 90)
# Make the potential(v) zero everywhere on the graph
v = np.zeros([NX,NY])
# Set the boundaries of your graph (voltages in this case) to a certain value
# Set the bottom boundary to 1
v[0,:] = 1.
# Set the top boundary to 0
v[-1,:] = 0.
# Set the left and right boundaries to a linearly decreasing potential.
v[:,:] = np.linspace(1.,0.,NY).reshape([NY,1])
# Iterate to solve Laplace equation (by creating a loop). range(n) repeats the loop n times
for istep in range(100000):
# (1:NX-1)means the starting point is 1 unit to the right of the left boundary and the ending point is 1 unit before the right boundary.
# (1:NY-1)means the starting point is 1 unit to above of the bottom boundary and the ending point is 1 unit below the top boundary.
```

```
# To find v on every point on the graph, we find the average of the four neighboring values
   v[1:NX-1,1:NY-1] = 0.25*(v[2:NX,1:NY-1] + v[0:NX-2,1:NY-1]
                    + v[1:NX-1,2:NY] +v[1:NX-1,0:NY-2])
   for j in range(nrow):
      v[j+1,NX/2-ridgeWidth[j]:NX/2+ridgeWidth[j]] = 1.


# Define the electric field vector components.
ex,ey = np.gradient(-v)
# print is just a tool to check our electric field values.
print (ex,ey)
# Plot the electric field vector using the quiver function. We also enlarge the size of our electric field vectors
plt.quiver(ey,ex,scale=1.)
# contour shows the contour lines. Python's origin is located at the top left corner of the screen; thus, we command Python to move its origin to the bottom left.
plt.contour(v,origin="lower")
#colorbar provides the 'colorbar' on the right
plt.colorbar()
# show simply shows the graph
plt.show()
```

Conclusion

Electrostatics tells us that electric field vectors and electric potential lines are always perpendicular to each other. It also tells us that wherever electric field vectors are jammed, then the electric field must have a high value there. In addition, electrostatics says that a conducting surface in an electric field will have no electric field flowing through it. Since, after coding the program in Python, none of the figures plotted by Python contradicted with these ideas, we then proved that the knowledge we have about electrostatics holds true in real life situations.