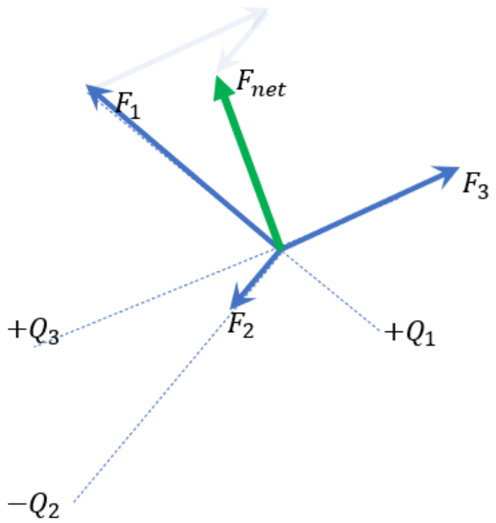
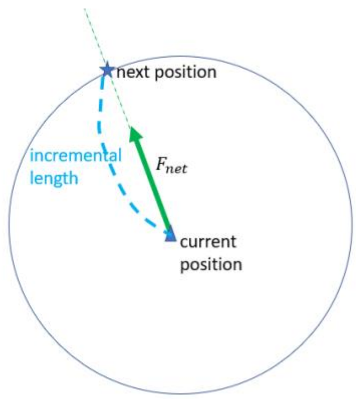
Plots of Electrical Force Lines and Electrical Potentials

In order to visualize the distribution of electrical force line over a plane, I come up with the following ways to depict it

1. Use traditional vector sum, adding horizontal and vertical component respectively, of individual field built from different charges. The resultant field is in the tangential direction of electrical force lines.



2. Use scalar sum of electrical potential from different charges.
3. The next point will be in the direction of electrical force, with distance equal to the incremental length from current position. Go to (1)

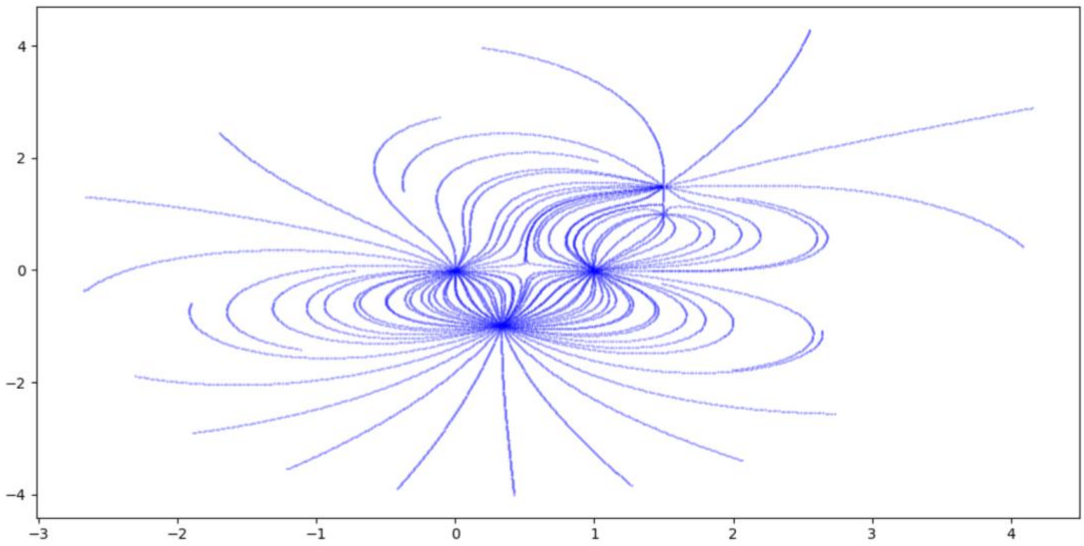


In my code, each electrical force line has length of 3 times the distance between source#1 and source#2. The electrical force line originating from a source is proportional to the charge of the source, according, according to Gauss' Law..

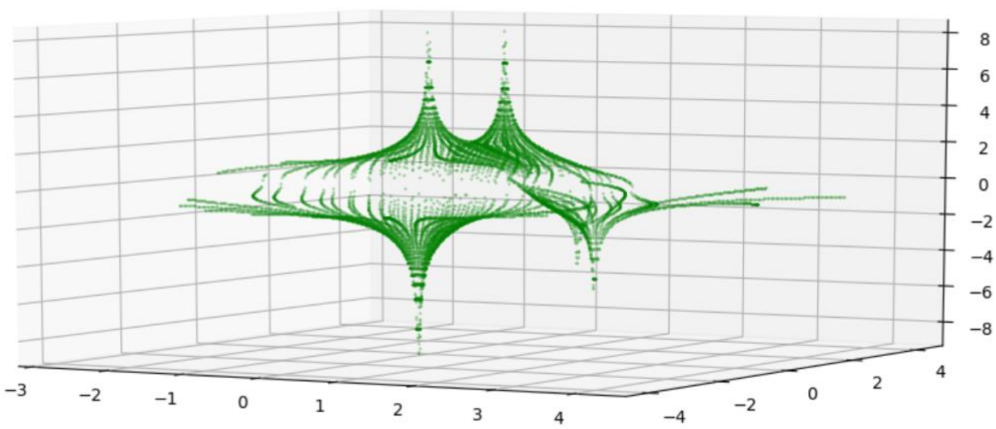
Here is the python source code you can run on your python environment. Make sure you are using python 3.x for python interpreter, particularly if you are with macOS which adopts its built-in python 2.7 by default

Simulation Result (example)

Field lines:



Potential:



Source Code in Python :

```python
 import numpy as np
import matplotlib.pyplot as plt
from  mpl_toolkits import mplot3d
pi=np.pi
xSource = [0.0, 1.0, 1.5, 1/3, 1.5]
ySource = [0.0,0.0, 1.5, -1.0, 1]
qSource = [6.0, 6.0,-3, -9.0, -1]
digit = 0
endOfInput = 0
while endOfInput == 0:

    try:
        print('Source #%d'%(digit))
        xS = float(input("X coordinator of Source: "))
        yS = float(input("Y coordinator of Source: "))
        qS = float(input("Charge of Source (Unit: e): "))

        if digit <= 1:
            xSource[digit] = xS
            ySource[digit] = yS
            qSource[digit] = qS
        digit+=1
    except:
        endOfInput = 1
        print('Value Invalid, will continue with default value')
        xS = 0.0
        yS = 0.0
        qS = 0.0
    if digit >= 2:
        xSource.append(xS)
        ySource.append(yS)
        qSource.append(qS)

'''
xSource = [0.0,1.0,2.0]
ySource = [0.0,0.0,1.0]
qSource = [1.0,-3.5,2.0]
'''
```

```python
k=1
x3D = []
y3D = []
potential3D = []

def dist(x1,y1,x2,y2):
    distance = np.sqrt(((x2-x1)**2)+((y2-y1)**2))
    return distance

def unitSin(x1,y1,x2,y2):
    xPortion = (x1-x2)/dist(x1,y1,x2,y2)
    return xPortion

def unitCos(x1,y1,x2,y2):
    yPortion = (y1-y2)/dist(x1,y1,x2,y2)
    return yPortion

def squareDist(x1,y1,x2,y2):
    distSqr = ((x2-x1)**2)+((y2-y1)**2)
    return distSqr

sourceSpacing = dist(xSource[0],ySource[0], xSource[1], ySource[1])
lineDensity = 4
rCircle = sourceSpacing/50
angleCircle = 0
sourceIndex = 0
print('Simulation ongoing!')
minimumSpacing = 1/100*sourceSpacing
while sourceIndex < len(qSource):

    forceLineIndex = 0

    while forceLineIndex < lineDensity*np.abs(qSource[sourceIndex]):

        angleAroundSource = forceLineIndex*2*pi/(lineDensity*np.abs(qSource[sourceIndex]))
        awayFromSource = 0
        xCurrent = xSource[sourceIndex]+rCircle*np.cos(angleAroundSource)
        yCurrent = ySource[sourceIndex]+rCircle*np.sin(angleAroundSource)
        while awayFromSource < 3*dist(xSource[0], ySource[0], xSource[1], ySource[1]):
```

```
            xField = 0
            yField = 0
            potential = 0
            everTooClose = 0
            for m in range(len(qSource)):
                if dist(xCurrent,yCurrent, xSource[m], ySource[m]) > minimumSpacing:
                    xField += qSource[m]/squareDist(xCurrent,yCurrent, xSource[m], ySource[m])
*unitSin(xCurrent,yCurrent, xSource[m], ySource[m])
                    yField += qSource[m]/squareDist(xCurrent,yCurrent, xSource[m], ySource[m])
*unitCos(xCurrent,yCurrent, xSource[m], ySource[m])
                    potential += qSource[m]/dist(xCurrent,yCurrent, xSource[m], ySource[m])
                else:
                    awayFromSource = 3*dist(xSource[0], ySource[0], xSource[1], ySource[1])
                    everTooClose = 1
            if everTooClose == 0:
                x3D.append(xCurrent)
                y3D.append(yCurrent)
                potential3D.append(potential)
                plt.plot(xCurrent, yCurrent, ".b", markersize = 0.5)
            if qSource[sourceIndex] > 0:
                xCurrent = xCurrent+rCircle*unitSin(xField,yField,0,0)
                yCurrent = yCurrent+rCircle*unitCos(xField,yField,0,0)
            else:
                xCurrent = xCurrent-rCircle*unitSin(xField,yField,0,0)
                yCurrent = yCurrent-rCircle*unitCos(xField,yField,0,0)
            awayFromSource += rCircle
        forceLineIndex += 1
    sourceIndex+=1
plt.show()




fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot3D(x3D,y3D,np.cbrt(potential3D),'.g', markersize = 0.5)
plt.show()
```