Q1

July 8, 2022

[1]:
```
#Q1 Where my own the integeration function is used
import numpy as np
import math
import matplotlib.pyplot as plt

# do integeration on func(x,y,arg1,..,), from x = a to b, y from gfun(x) to␣
↪hfun(x)
def dblquad(func, a, b, gfun, hfun, args=(), N = 100):
    x = a
    x_step = (b-a)/N
    result = 0
    while (x < b):
        y = gfun(x)
        y_max = hfun(x)
        y_step = (y_max - y) / N
        while (y < y_max):
            result += (func(x,y,*args) * y_step * x_step)
            y += y_step
        x += x_step
    return result

epsilon_0 = 8.8541878128*(10**-12)
k = 1 / (4* np.pi * epsilon_0)

# define a point class
class Point:
    def __init__(self, x: float, y: float, z: float):
        self.x = x
        self.y = y
        self.z = z

# calculate x-distance between two points
def disX(p1: Point, p2: Point):
    return p1.x-p2.x

# calculate y-distance between two points
def disY(p1: Point, p2: Point):
```

```python
    return p1.y-p2.y

# calculate z-distance between two points
def disZ(p1: Point, p2: Point):
    return p1.z-p2.z

# calculate distance between two points
# reqiure: p1 != p2
def dis(p1: Point, p2: Point):
    return math.sqrt((p1.x-p2.x)**2+(p1.y-p2.y)**2+(p1.z-p2.z)**2)

# get the field at point p2, generated by charge q as p1, ignore constants pi␣
↪and epsilon_0
def pointField(p1, p2, q):
    Ex = q*disX(p2, p1)/(dis(p1, p2)**3)
    Ey = q*disY(p2, p1)/(dis(p1, p2)**3)
    Ez = q*disY(p2, p1)/(dis(p1, p2)**3)
    return Ex, Ey, Ez

# the integrand of the result for x-component
def inte_x(y, x, p):
    q = Point(x,y,0)
    return disX(p, q)/(dis(q, p)**3)

# the x-component of E at p from the disk with raduis R, ignoring the constants
def Ex(R, p):
    return dblquad(inte_x, -R, R, lambda x: -math.sqrt(R**2-x**2), lambda x:␣
↪math.sqrt(R**2-x**2), args=(p,))

# the integrand of the result for y-component
def inte_y(y, x, p):
    q = Point(x,y,0)
    return disY(p, q)/(dis(q, p)**3)

# the y-component of E at p from the disk with raduis R, ignoring the constants
def Ey(R, p):
    return dblquad(inte_y, -R, R, lambda x: -math.sqrt(R**2-x**2), lambda x:␣
↪math.sqrt(R**2-x**2), args=(p,))

# the integrand of the result for z-component
def inte_z(y, x, p):
    q = Point(x,y,0)
    return disZ(p, q)/(dis(q, p)**3)

# the z-component of E at p from the disk with raduis R, ignoring the constants
def Ez(R, p):
```

```
    return dblquad(inte_z, -R, R, lambda x: -math.sqrt(R**2-x**2), lambda x:␣
↪math.sqrt(R**2-x**2), args=(p,))

# get the field at p generated by disk with radius R and density delta
# require: p is not on the disk
def diskField(p: Point, R, delta, slient = False):
    if (p.z == 0 and dis(p,Point(0,0,0)) <= R):
        print("Error: the point is on the disk")
        return 0
    C = delta * k
    E_x = Ex(R, p)
    E_y = Ey(R, p)
    E_z = Ez(R, p)
    E_size = math.sqrt(E_x**2 + E_y**2 + E_z**2) * C
    E_x *= C
    E_y *= C
    E_z *= C
    # print the result
    if not slient:
        print("The Radius is {}, the field at point ({},{},{}) is {}x, {}y,␣
↪{}z, with size {}".format(R, p.x, p.y, p.z, E_x, E_y, E_z, E_size))
    return E_x, E_y, E_z, E_size
```

[2]:
```
# Q3
# try some examples
# p = (1,1,0), R = 1, delta = 1
delta = 1
p = Point(1,1,0)
R = 1
diskField(p, R, delta)
# p = (1,0,1), R = 1, delta = 1
delta = 1
p = Point(1,0,1)
R = 1
diskField(p, R, delta)
# p = (0,1,1), R = 1, delta = 1
delta = 1
p = Point(0,1,1)
R = 1
diskField(p, R, delta)
# p = (0,0,1), R = 2, delta = 1
delta = 1
p = Point(0,0,1)
R = 2
diskField(p, R, delta)
# p = (1,1,1), R = 1, delta = 80
delta = 80
```

```
p = Point(1,1,1)
R = 1
diskField(p, R, delta)
# p = (0,0,-1), R = 2, delta = 1
delta = 1
p = Point(0,0,-1)
R = 2
diskField(p, R, delta)
```

```
The Radius is 1, the field at point (1,1,0) is 12808094487.245998x,
12806797867.758038y, 0.0z, with size 18112464106.79213
The Radius is 1, the field at point (1,0,1) is 7087341940.482487x,
-1581222.201402223y, 10091912290.666721z, with size 12331954912.506603
The Radius is 1, the field at point (0,1,1) is 29047323.34801386x,
7088512381.471768y, 10109440131.836927z, with size 12347009002.526371
The Radius is 2, the field at point (0,0,1) is 90705467.47484814x,
-1163613.6334037671y, 31253624935.843414z, with size 31253756581.668705
The Radius is 1, the field at point (1,1,1) is 385096880039.6241x,
384339531905.1661y, 497367444292.57574z, with size 737150498503.1641
The Radius is 2, the field at point (0,0,-1) is 90705467.47484814x,
-1163613.6334037671y, -31253624935.843414z, with size 31253756581.668705
```

[2]:
```
(90705467.47484814,
 -1163613.6334037671,
 -31253624935.843414,
 31253756581.668705)
```

[3]:
```
# Q4 Compare with the formula on central axis, with height h
# for the sake of simplicity, set the constant k and delta to 1
k = 1
delta = 1
def centralField(R, delta, h):
    E_z = 2*np.pi*k*delta*(1-h/math.sqrt(h**2+R**2))
    print("The Radius is {}, the field at point ({},{},{}) is {}, {}, {}".
 ↪format(R, 0, 0, h, 0, 0, E_z))
    return E_z

# h = 10, R = 1
h = 10
p = Point(0,0,h)
R = 1
__, __, __, E1 = diskField(p, R, delta)
Ez2 = centralField(R, delta, h)
print("the relative difference is {}".format(abs((E1-Ez2)/Ez2)))

# h = 9999, R = 1
h = 9999
```

```
p = Point(0,0,h)
R = 1
__, __, __, E1 = diskField(p, R, delta)
Ez2 = centralField(R, delta, h)
print("the relative difference is {}".format(abs((E1-Ez2)/Ez2)))
```

```
The Radius is 1, the field at point (0,0,10) is 1.3895526879775288e-05x,
-1.7825854351383657e-07y, 0.031299189011418836z, with size 0.0312991920964418
The Radius is 1, the field at point (0,0,10) is 0, 0, 0.031182253554923163
the relative difference is 0.0037501632559259574
The Radius is 1, the field at point (0,0,9999) is 1.4108711978736831e-14x,
-1.8099338513306525e-16y, 3.154095299104739e-08z, with size
3.154095299105055e-08
The Radius is 1, the field at point (0,0,9999) is 0, 0, 3.142221078642826e-08
the relative difference is 0.003778925850550751
```

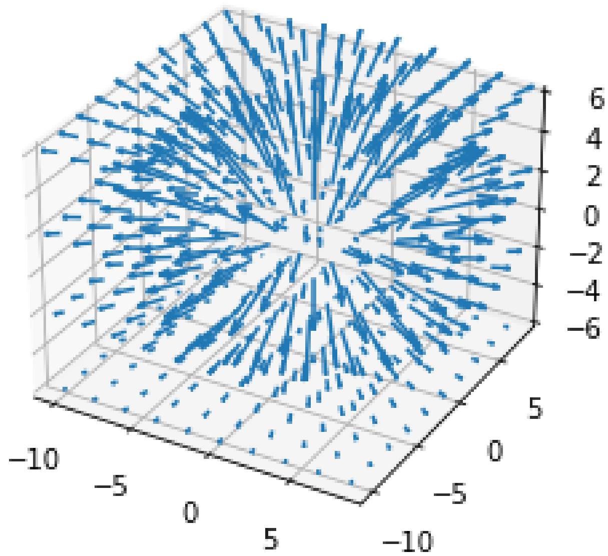
Note that when the point is further away, the error is larger, but still very small (less than 0.01)

[5]:
```
# plot symmetry
ax = plt.figure().add_subplot(projection='3d')
# Make the grid
x, y, z = np.meshgrid(np.arange(-10, 10, 2),
                      np.arange(-10, 10, 2),
                      np.arange(-10, 10 ,4))

# wrapper for diskField, but take constant delta=1, R =5, and x,y,z instead of␣
↪a point p
def disField1(x,y,z):
    p = Point(x,y,z)
    return diskField(p,5,1,True)

u,v,w = [], [], []
for i in np.arange(len(x)):
    u.append([])
    v.append([])
    w.append([])
    for j in np.arange(len(x[0])):
        u[-1].append([])
        v[-1].append([])
        w[-1].append([])
        for k in np.arange(len(x[0][0])):
            E_x, E_y, E_z, __ = disField1(x[i][j][k],y[i][j][k],z[i][j][k])
            u[-1][-1].append(E_x)
            v[-1][-1].append(E_y)
            # Ez is way too large in comparison to the other two, divide by␣
↪constant just to show symmetry
            w[-1][-1].append(E_z/2)
```

```
ax.quiver(x, y, z, u, v, w, length=0.7, normalize=False)
plt.show()
```



[ ]: