Q2

July 8, 2022

[1]:
```
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=(), x_step = 0.01, y_step = 0.01):
    x = a
    result = 0
    while (x < b):
        y = gfun(x)
        y_max = hfun(x)
        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):
    return p1.y-p2.y

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

```python
    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)

# 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 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 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 x-component of E at p from the rectangle of (a,b), ignoring the constants
def Ex(a, b, p):
    return dblquad(inte_x, -a/2, a/2, lambda x: -b/2, lambda x: b/2, args=(p,))

# the y-component of E at p from the rectangle of (a,b), ignoring the constants
def Ey(a, b, p):
    return dblquad(inte_y, -a/2, a/2, lambda x: -b/2, lambda x: b/2, args=(p,))

# the z-component of E at p from the rectangle of (a,b), ignoring the constants
def Ez(a, b, p):
    return dblquad(inte_z, -a/2, a/2, lambda x: -b/2, lambda x: b/2, args=(p,))

# get the field at p generated from the rectangle of (a,b) and density delta
# require: p is not on the rectangle, a,b >0
def rectangleField(p: Point, a, b, delta, slient = False):
    if (p.z == 0 and p.x <= a/2 and p.x >= -a/2 and p.y <= b/2 and p.y >= -b/2):
        print("Error: the point is on the rectangle")
        return 0
    C = delta * k
    E_x = Ex(a, b, p)
    E_y = Ey(a, b, p)
    E_z = Ez(a, b, 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 rectangle is {}*{}, the field at point ({},{},{}) is {}x,␣
↪{}y, {}z, with size {}".format(a, b, 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 = (4,3,0), a = 1, b = 1, delta = 1
delta = 1
p = Point(4,3,0)
a = 1
b = 1
rectangleField(p, a, b, delta)

# p = (4,3,3), a = 1, b = 1, delta = 1
delta = 1
p = Point(4,3,3)
a = 1
b = 1
rectangleField(p, a, b, delta)

# p = (4,0,3), a = 1, b = 1, delta = 1
delta = 1
p = Point(4,0,3)
a = 1
b = 1
rectangleField(p, a, b, delta)
```

```
The rectangle is 1*1, the field at point (4,3,0) is 288221765.9275177x,
216242227.14645335y, 0.0z, with size 360322754.1463835
The rectangle is 1*1, the field at point (4,3,3) is 180788211.97337925x,
135644105.54040807y, 136422961.3546347z, with size 263997964.65343636
The rectangle is 1*1, the field at point (4,0,3) is 286106584.21990496x,
357134.3337067172y, 216458060.3973014z, with size 358763427.60916376
```

[2]:
```
(286106584.21990496, 357134.3337067172, 216458060.3973014, 358763427.60916376)
```

[4]:
```
# Q4 Compare with the formula on central axis, with height h
# for the sake of simplicity, set the constant delta to 1
delta = 1
def planeField(delta, h):
    E_z = delta/(2*epsilon_0)
    print("The field at point ({},{},{}) is {}, {}, {}".format(0, 0, h, 0, 0,␣
↪E_z))
    return E_z
```

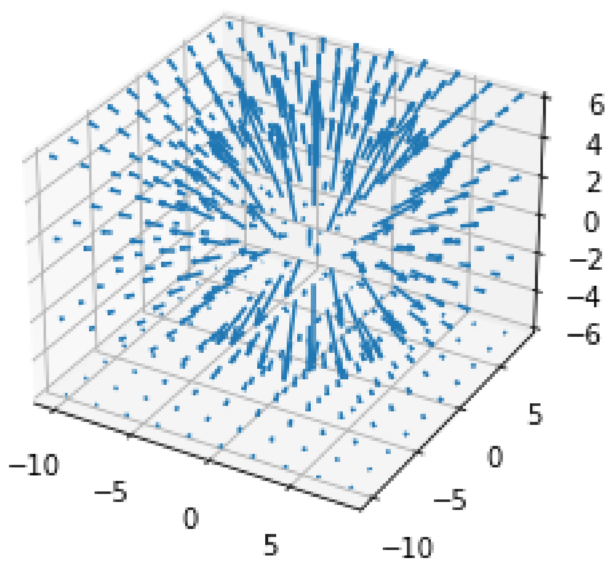
```python
# large plane
h = 0.01
p = Point(0,0,h)
a = 10
b = 10
__, __, __, E1 = rectangleField(p, a, b, delta)
Ez2 = planeField(delta, h)
print("the relative difference is {}".format(abs((E1-Ez2)/Ez2)))

# smaller plane
h = 0.01
p = Point(0,0,h)
a = 1
b = 1
__, __, __, E1 = rectangleField(p, a, b, delta)
Ez2 = planeField(delta, h)
print("the relative difference is {}".format(abs((E1-Ez2)/Ez2)))
```

```
The rectangle is 10*10, the field at point (0,0,0.01) is 0.009480480882536226x,
0.009491984174046104y, 56823095526.54007z, with size 56823095526.54007
The field at point (0,0,0.01) is 0, 0, 56470453368.65096
the relative difference is 0.006244719793322444
The rectangle is 1*1, the field at point (0,0,0.01) is 254072952.2688889x,
254072952.26890674y, 55907935637.810875z, with size 55909090257.41479
The field at point (0,0,0.01) is 0, 0, 56470453368.65096
the relative difference is 0.009940828836125671
```

Note that when the rectangle is smaller, the difference from an infinite plane is larger, but still very small (<0.01), as long as the height is much smaller than the area

[7]:
```python
# 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, a = 5, b=7, and x,y,z␣
 ↪instead of a point p
def recField1(x,y,z):
    p = Point(x,y,z)
    return rectangleField(p,5,7,1,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])):
            # after investigation on the meshgrid function, it very wierdly is␣
↪in the form y-x-z loop
            E_x, E_y, E_z, __ = recField1(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)


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

plt.show()
```



[ ]: