FYS1120 Oblig 1

Evan Matel
(Dated: September 23, 2021)

I. OPPGAVE 1

Our project was to simulate two positively charged rings, each with a charge of 1, both centered around the origin, one in the xy-plane and one in the xz-plane (as seen in figure 1), and then plot the electric potential and electric field.

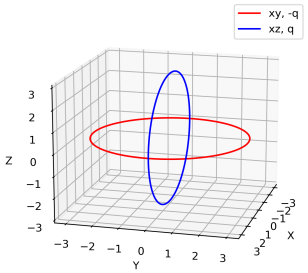

Figure 1

We split up the rings into small pieces and found the electrical potential of every individual piece, and summed them up afterwards to get our values. We then plotted the electrical potential under all our graphs using numpy's gradient function.

Next we used the formula:

$$\vec{E} = -\nabla V$$

to find the electric field and then used numpy's streamline function to plot that. We plotted both the potential and electric field as seen from all three planes. Since both the rings are equally charged, they look identical.

II. OPPGAVE 2

We made a simplified version of the system and approximated the electric field. Using superposition, since the rings with charge $1C$ were centered around the origin, we chose to treat the rings as one point at the origin with a charge of $2C$. We used coulombs law,

$$\vec{E} = \frac{Q}{4\pi\epsilon_0}\frac{\vec{r}}{|r|^3}$$

on a limited set of points and plotted the results. We compared these plots to our numerically calculated electrical fields.

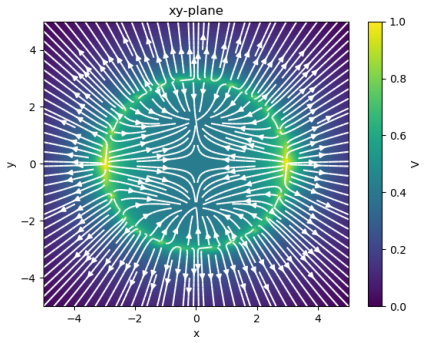

Figure 2: The ring in the xy-plane.

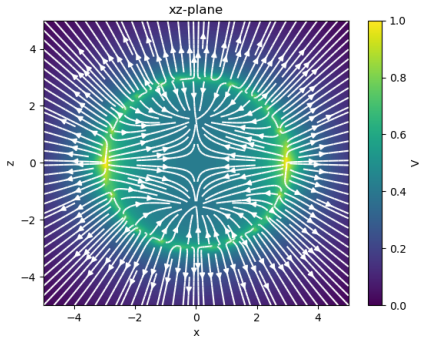

Figure 3: The ring in the xz-plane.

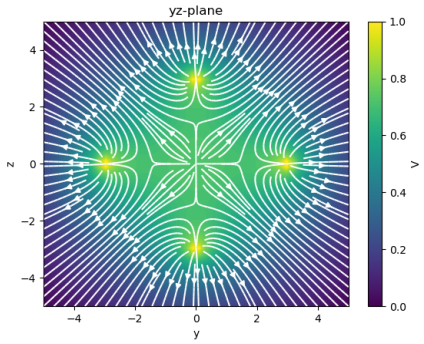

Figure 4: A cross section of the rings in the yz-plane.

As one can see in figures 5-8, the graphs of the approximations look almost identical to the "zoomed out" numerically generated plots. If one looks really closely at the quiver plots (figures 5-6), specifically the medium sized arrows along side the arrows that shoot out of the plot, one could see that the two plots don't align exactly.

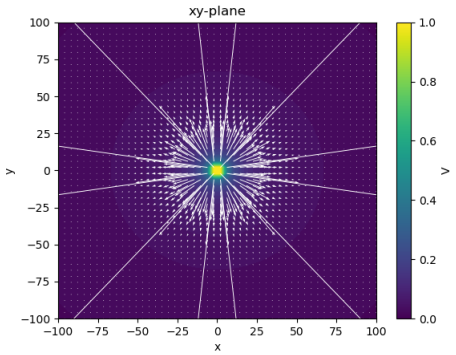

Figure 5: The ring in the xy-plane visualised by numpy's quiver function, seen from far away.

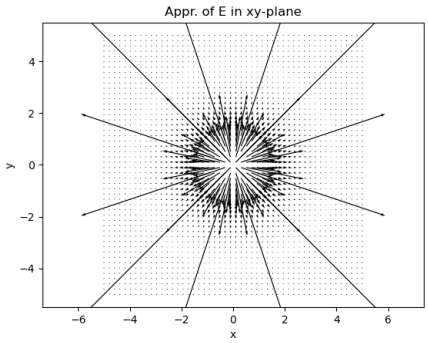

Figure 6: The point charge approximation visualised by numpy's quiver function.

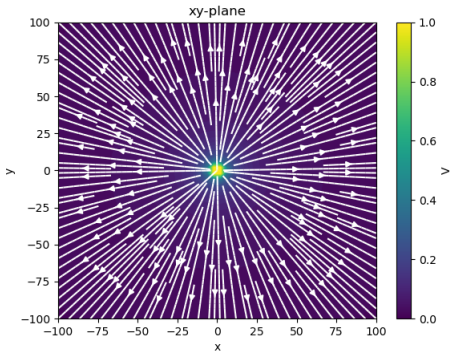

Figure 7: The ring in the xy-plane visualised by numpy's streamline function, seen from far away.

The same can be said about the streamline plots(figures 7-8), some arrows on the outer rim don't align exactly, but if one weren't to spend that much time looking at them, one could say they are two different plots of the same data.

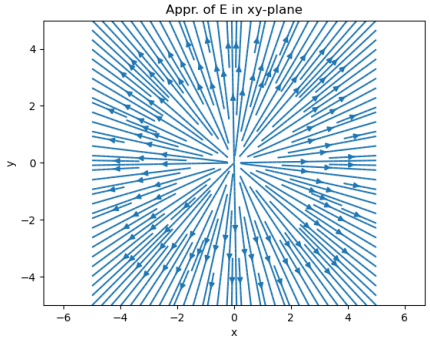

Figure 8: The point charge approximation visualised by numpy's streamline function.

III. CODE

```
import numpy as n
import matplotlib.pyplot as plt


def potential(r, Q, R):  #  potensial funksjon
    v = 0
    for i in range(len(R)):
        Ri = r - R[i]   # posisjonensvektoren
        Qi = Q[i]    # ladning
        Rinorm = n.linalg.norm(Ri) # lengden av posisjonensvektoren
        v += Qi/Rinorm
    return v


r = 3 #radius
q = 2 # ladning

R = []
Q = []

M = 100
Dtheta = (2 * n.pi) / M

for i in range(M):   # lager ringene her
    theta = i * Dtheta
    x = r * n.cos(theta)
    y = r * n.sin(theta)
    z = y
    rxy = n.array([x, y, 0])
    rxz = n.array([x, 0, z])

    R.append(rxy)
    R.append(rxz)

    Qi = q / M
    Q.append(Qi)
    Q.append(Qi)



N = 50  # base verdier N=50 og Lx=5
Lx = 5

x = n.linspace(-Lx, Lx, N)
y = n.linspace(-Lx, Lx, N)
z = n.linspace(-Lx, Lx, N)


ry, rz = n.meshgrid(y, z)
rx, rY = n.meshgrid(x, y)
RX, RZ = n.meshgrid(x, z)


Vyz = n.zeros((N, N))
Vxy = n.zeros((N, N))
Vxz = n.zeros((N, N))

for i in range(len(ry.flat)):
    ryz = n.array([0, ry.flat[i], rz.flat[i]])
    rxy = n.array([rx.flat[i], rY.flat[i], 0])
    rxz = n.array([RX.flat[i], 0, RZ.flat[i]])

    Eyz = potential(ryz, Q, R)
    Exy = potential(rxy, Q, R)
    Exz = potential(rxz, Q, R)

    Vyz.flat[i] = Eyz
    Vxy.flat[i] = Exy
    Vxz.flat[i] = Exz


Ey, Ex = n.gradient(-Vyz)

Ey2, Ex2 = n.gradient(-Vxy)

Ey3, Ex3 = n.gradient(-Vxz)



# plotter ringene i 3d

X = []
Y = []
Z = []
for i in range(M+1):
    theta = i * Dtheta
    x = r * n.cos(theta)
    y = r * n.sin(theta)

    X.append(x)
    Y.append(y)
    Z.append(0)

fig = plt.figure()
ax = plt.axes(projection='3d')

ax.plot(X, Y, Z, color='red', label='xy')
ax.plot(X, Z, Y, color='blue', label='xz')


ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.legend()
ax.view_init(15, 15)
plt.savefig('ringer')
plt.show()




 # plotter et tversnitt i det yz-planet

plt.contourf(ry, rz, Vyz, 80)
plt.quiver(ry, rz, Ex, Ey, color='white')
plt.colorbar(label='V')
plt.xlabel('y')
plt.ylabel('z')
plt.title('yz-plane')
plt.figure(figsize=(8, 8))
plt.savefig('yz-quiver')
plt.show()

plt.contourf(ry, rz, Vyz, 80)
plt.streamplot(ry, rz, Ex, Ey, 2, color='white')
plt.colorbar(label='V')
plt.xlabel('y')
plt.ylabel('z')
plt.title('yz-plane')
plt.axis()
plt.savefig('yz-stream')
plt.show()

# plotter et tversnitt i det xy-planet

plt.contourf(rx, rY, Vxy, 80)
plt.quiver(rx, rY, Ex2, Ey2, color='white')
plt.colorbar(label='V')
plt.xlabel('x')
plt.ylabel('y')
plt.title('xy-plane')
plt.figure(figsize=(8, 8))
plt.savefig('xy-quiver')
plt.show()

plt.contourf(rx, rY, Vxy, 80)
plt.streamplot(rx, rY, Ex2, Ey2, 2, color='white')
plt.colorbar(label='V')
plt.xlabel('x')
plt.ylabel('y')
plt.title('xy-plane')
plt.axis()
plt.savefig('xy-stream')
plt.show()

# plotter et tversnitt i det xz-planet

plt.contourf(RX, RZ, Vxz, 80)
plt.quiver(RX, RZ, Ex3, Ey3, color='white')
plt.colorbar(label='V')
plt.xlabel('x')
plt.ylabel('z')
```

```
plt.title('xz-plane')
plt.figure(figsize=(8, 8))
plt.savefig('xz-quiver')
plt.show()

plt.contourf(RX, RZ, Vxz, 80)
plt.streamplot(RX, RZ, Ex3, Ey3, 2, color='white')
plt.colorbar(label='V')
plt.xlabel('x')
plt.ylabel('z')
plt.title('xz-plane')
plt.axis()
plt.savefig('xz-stream')
plt.show()




def AnalyticCoulombs(Q, ref):
    e0 = 8.854187817e-12
    a = 1 / (4 * n.pi * e0)
    E = a * Q * ref / n.linalg.norm(ref) ** 3
    return E


x = n.linspace(-Lx, Lx, N)
y = n.linspace(-Lx, Lx, N)
rx, ry = n.meshgrid(x, y)
```

```
AnalEx = n.zeros((N, N))
AnalEy = n.zeros((N, N))

for i in range(len(rx.flat)):
    r = n.array([rx.flat[i], ry.flat[i]])
    AnalEx.flat[i], AnalEy.flat[i] = AnalyticCoulombs(q, r)

plt.quiver(rx, ry, AnalEx, AnalEy)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Appr. of E in xy-plane')
plt.axis('equal')
plt.savefig('xyappr-quiver')
plt.show()

plt.streamplot(rx, ry, AnalEx, AnalEy, 2)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Appr. of E in xy-plane')
plt.axis('equal')
plt.savefig('xyappr-stream')
plt.show()

"""
Kjoreeksempel

Process finished with exit code 0
"""
```