PH2102 Bonus Homework Note

Lim Jiun Yeu
U1640220C
School of Physical and Mathematical Sciences (SPMS)
limj0187@e.ntu.edu.sg

1 Spyder

The language used for this homwork is Python. The compiler used for this Python code is Spyder.

In order to see the animation, please open the code in Sspyder program, go to: Tools – Preferences – IPython console – Graphics – Graphics Backend – Automatic. This will cause it to open the animation in a separate window instead of running it in the console, where the animation will not play. Run the code in the IPython console to see the animation.

2 Explaination of Code

The variables of the particles used includes the mass, the charge, the magnetotron motion, the cyclotron motion and the axial oscillations.

Listing 1: Variables

```
mass = 0
charge = 0
velocity = 0
magnetotron_radius = 0
magnetotron_theta = 0
magnetotron_omega = 0
z = 0
z_omega = 0
cyclotron_radius = 0
cyclotron_theta = 0
cyclotron_omega = 0
```

This part is just to initialise the variables that will be used later.

Listing 2: User defined variables

```
mass = 1
```

```
charge = 1
magnetic_field = 1
electric_potential = 10
trap_dimension = 1
velocity = 1
```

This part is the variables given by the user, they can be changed for different results.

Listing 3: Calculated variables from the user given variables

```
z_omega = (charge*electric_potential)/(mass*trap_dimension)
magnetotron_radius = (mass*(velocity))/(magnetic_field*charge)
magnetotron_omega = (velocity) / magnetotron_radius
cyclotron_omega = (z_omega**2)/(2*magnetotron_omega)
cyclotron_radius = (velocity)/cyclotron_omega
```

z.omega is the angular frequency of the oscillation in the z direction due to the electric field. The calculations for its angular frequency come from the formula given in by [1].

$$\omega_z^2 = \frac{eV_0}{md^2} \tag{1}$$

magnetotron$_r$adius is calculated by using the formula for uniform circular motion.

$$Bqv = \frac{mv^2}{r} \tag{2}$$

$$r = \frac{mv}{Bq} \tag{3}$$

magnetotron_omega is calculated by using the formula for the angular frequency,

$$\omega = \frac{v}{r} \tag{4}$$

cyclotron_omega is found by a formula, specifically equation 2.82, in [1]. That is given as,

$$\omega_c = \frac{\omega_z^2}{2\omega_m} \tag{5}$$

lastly, *cyclotron_radius* is found by the same way as the *magnetotron_radius*.

$$\omega = \frac{v}{r} \tag{6}$$

Listing 4: Functions

```
def x_position(radius,theta,cyclotron_radius,cyclotron_theta):
    return
    radius*math.cos(theta)+(cyclotron_radius*math.cos(cyclotron_theta))
def y_position(radius,theta,cyclotron_radius,cyclotron_theta):
    return
    radius*math.sin(theta)+(cyclotron_radius*math.sin(cyclotron_theta))
def z_position(amplitude, t):
    return
    amplitude*math.cos(z_omega*t)
```

The Functions in Listing 4 calculate where the particle for the animation. They make change the cylindrical coordinates into cartesian coordinates for the animation code to use.

$$xposition = r_m \times cos(\theta_m) + r_c \times cos(\theta_c) \tag{7}$$

$$yposistion = r_m \times sin(\theta_m) + r_c \times sin(\theta_c) \tag{8}$$

The x and y position of the particle takes into account the contributions from both the magnetotron as well as the cyclotron. since both of the motions are in cylindrical coordinates, *sin* and *cos* are used as to convert the cylindrical coordinates to cartesian coordinates.

$$zpostion = amplitude \times cos(w_z t) \tag{9}$$

The z-axis is only affected by the axial motion. The amplitude is user defined.

3 Animation Code

Listing 5: Animation Code

```
fig = plt.figure()
ax = plt.subplot(111,projection = '3d')

def animate(i):
    t = i*(5*10**-1)
    xs = x_position(magnetotron_radius,magnetotron_theta
    +(magnetotron_omega*t),cyclotron_radius ,cyclotron_theta
    +(cyclotron_omega*t))
    ys = y_position(magnetotron_radius,magnetotron_theta
    +(magnetotron_omega*t),cyclotron_radius ,cyclotron_theta
    +(cyclotron_omega*t))
    zs = z_position(1,t)
    return ax.scatter(xs,ys,zs)
anim = animation.FuncAnimation(fig ,animate ,frames=1000,save_count=1000)
plt.show()
#this portion is for saving the animation into an mp4 file.
#writer = animation.FFMpegWriter()
#anim.save('mymovie.mp4', writer)
```

For the animation code, first, the figure is initialised as fig, and a 3d projection of a plot is initialised as ax.

then the function that we will use to animate, called animate, makes use of the previous 3 functions for the x,y and z position to return a plot of the position of the particle.

$$t = i \times 5 \times 10^-1 \tag{10}$$

This equation determines the interval between each time that the particle is drawn on the plot. Changing this value will, in layman terms, change how fast the particle moves.

The *anim* portion uses the *FuncAnimation* function, that will call the *animate* function repeatedly to continuously draw new points on the figure in order to create the animation. The 'frames = 1000' is the number of times that the function will be called. Change to 'None' if you want the animation to continue non-stop. 'save_count = 1000' is the number of frames that will be saved if the same function is used.

Lastly, to save the animation into an MP4 file, a FFMPEG writer is used. to save, please make sure that FFMPEG is installed on your computer. Then, the anim.save function will save it as an MP4 file.

4 Execution

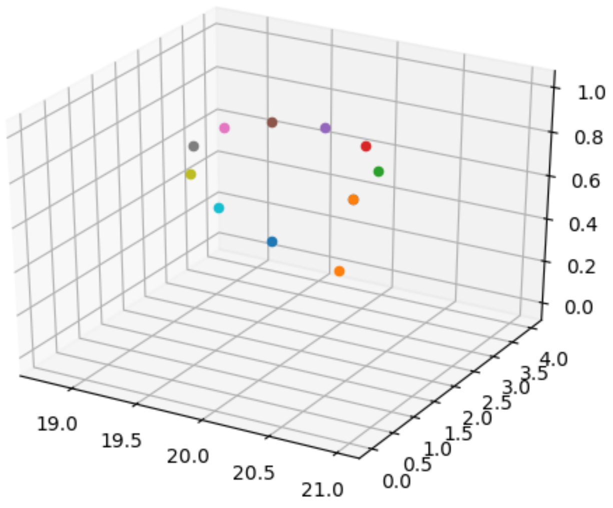

Figure 1: At t ≈ 0.1s

At the start of the animation, we can see the particle moving along the x-y axis in a circle due to the magnetic force. The motion soon starts to move down along the z-axis due to the quadrupole electric force.

From Figure 2, after a few seconds, we can see that the particle has begun to oscillate in the z direction.

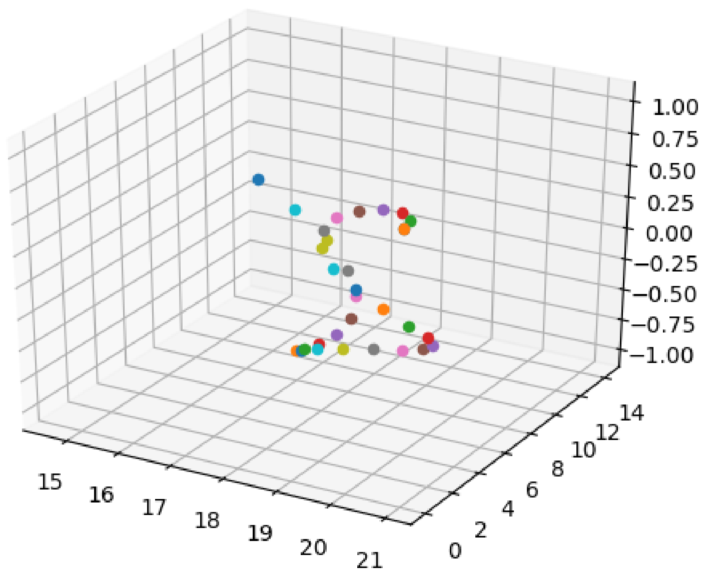

Figure 2: At t ≈ 3s

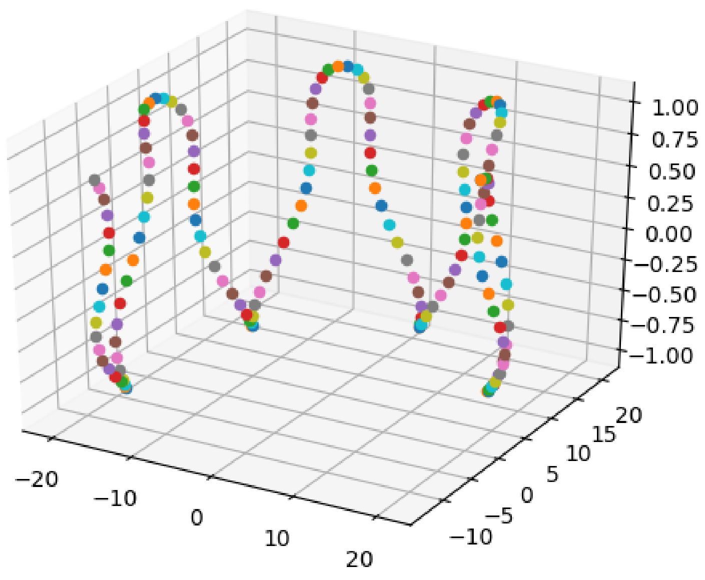

Figure 3: At t ≈10s

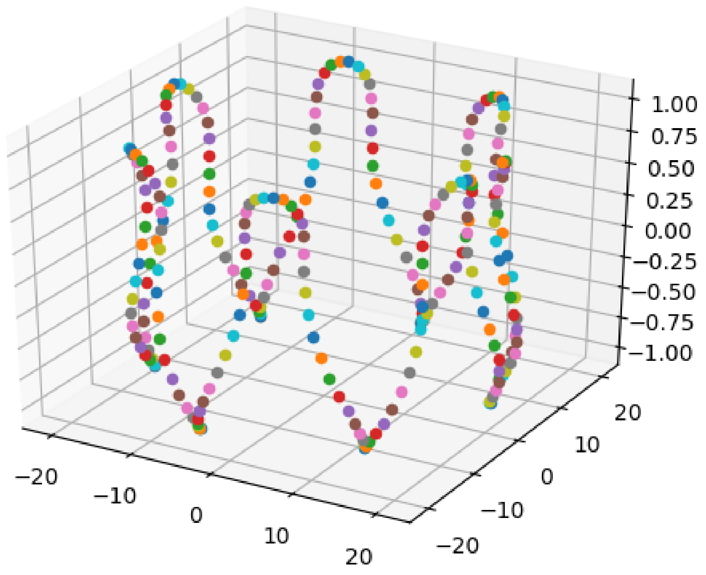

Figure 4: At t ≈ 20s

from Figure 3 and 4, the shape of the oscillations can be easily seen and the particle can be seen to have a circular motion.

References

[1] Brown LS, Gabrielse G. Geonium theory: Physics of a single electron or ion in a Penning trap. 1986;58(1):233–311.