Bonus Homework

Adrian Tang (U1640241G)
PH 2102 - Electromagnetism

April 22, 2018

1 Introduction

A Penning trap is a magneto-electric trap for confining charged particles by using a homogeneous axial magnetic field and an inhomogeneous quadrupole electric field. This kind of trap is particularly well suited to precision measurements of properties of ions and stable subatomic particles[1].

2 Theory

In a Penning trap, a uniform magnetic field is applied in the z-direction,

$$\vec{B} = B_0 \hat{z}, \tag{1}$$

along with a quadrupole electric potential,

$$V(\hat{r}) = \frac{V_0}{2d^2}\left(z^2 - \frac{s^2}{2}\right), \tag{2}$$

where z and s are cylindrical coordinates and d is a characteristic trap dimension. A particle with mass m and charge q moving at velocity $\boldsymbol{v}$ through a magnetic field $\vec{B}$ and electric $\vec{E}$ experiences a Lorentzian force:

$$\vec{\boldsymbol{F}} = q\vec{\boldsymbol{E}} + q\vec{\boldsymbol{v}} \times \vec{\boldsymbol{B}} \tag{3}$$

To quantify the Penning trap trajectories, we have to solve the equations of motion:

$$\ddot{s} = \frac{q}{m}(\vec{\boldsymbol{E}} + \dot{s} \times \vec{\boldsymbol{B}}) \tag{4}$$

Given that

$$\vec{\boldsymbol{E}} = -\vec{\nabla} V, \tag{5}$$

we can rerwite Equation 4 in cartesian coordinates as

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = \frac{qV_0}{2md^2}\begin{pmatrix} x \\ y \\ -2z \end{pmatrix} + \frac{qB_0}{m}\begin{pmatrix} \dot{y} \\ -\dot{x} \\ 0 \end{pmatrix} \tag{6}$$

The z-component of this differential equation is independent of the radial components, and is in the form of an undamped harmonic oscillator with solution:

$$z(t) = d\,\mathrm{e}^{i\omega_z t} \quad \text{with} \quad \omega_z = \sqrt{\frac{qV_0}{md^2}} \tag{7}$$

where ω_z is the **axial frequency**. The radial equations of motions can then be simplified to:

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \end{pmatrix} = \omega_c \begin{pmatrix} \dot{y} \\ -\dot{x} \end{pmatrix} + \frac{1}{2}\omega_z^2 \begin{pmatrix} x \\ y \end{pmatrix} \quad \text{with} \quad \omega_c = \frac{qB_0}{m} \tag{8}$$

where ω_c is the **cyclotron frequency**. These equations then have a solution in the form:

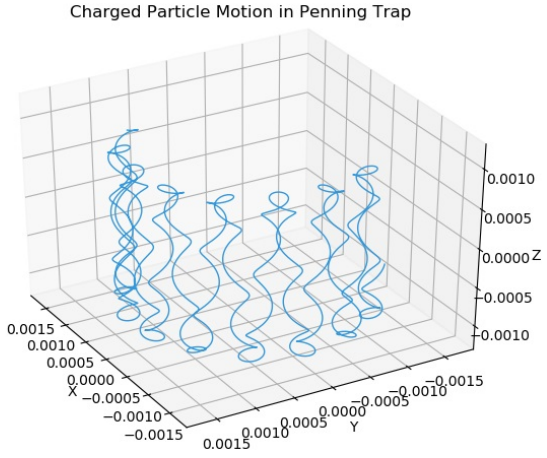
$$s(t) = s_0\, \mathrm{e}^{-iwt} \qquad \text{with} \qquad \omega_\pm = \frac{1}{2}\left(\omega_c \pm \sqrt{\omega_c^2 - 2\omega_z^2}\right) \tag{9}$$

The faster frequency ω_+ is the **reduced cyclotron frequency** while ω_- is the **magnetron frequency**. For a stable trajectory, the frequencies must be real valued. This requires

$$B_0 > \sqrt{\frac{2mV_0}{qd^2}} \tag{10}$$

3 Animtation of a particle in a Penning trap

Figure 1: Sample view of animation



The trajectory of a particle in an ideal Penning trap can be computed from the equations of motions found above. We will use Python 3.6 to create the animation of this trajectory. The matplotlib library will be used to do the graphing and animation. The numpy library is also used to facilitate manipulation of arrays in python. The animation can be viewed by simply running the attached python code after installing the required libraries. The required libraries are:

```
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as p3
import matplotlib.animation as animation
```

We require the initial conditions of our particle, as well as the specifications of the Penning trap. These can be changed to any suitable set of conditions. By default, these have been set to be the motion of an Rb-85 ion inside the Penning trap of TITAN at TRIUMF[4]. Defining our constants:

```
# Defining constants
x0_position = 0.001  # Initial x position (m)
y0_position = 0.001  # Initial y position (m)
z0_position = 0.001  # Initial z position (m)
x0_velocity = 300.0  # Initial x velocity (m/s)
y0_velocity = 400.0  # Initial y velocity (m/s)
z0_velocity = 50.0  # Initial z velocity (m/s)
q = 1.602176565 * 10**-19  # Charge of particle (C)
m = 1.411458082 * 10**-25  # Cass of particle (kg)
B0 = 3.7  # Magnitude of the uniform magnetic field in z-direction (T)
V0 = 35.75  # Voltage between ring electrodes and end caps (V)
d = 0.01121  # Characteristic trap dimension (m)
```

Then, based on the initial conditions, we can calculate the frequencies using:

```
# Frequencies
omega_c = q * B0 * (1/m)  # Cyclotron frequency (rad/s)
omega_z = np.sqrt(q * V0 * (1/m) * (1/d**2))  # axial frequency (rad/s)
omega_plus = 0.5 * (omega_c + np.sqrt(omega_c**2 - (2 * (omega_z**2))))  # Reduced cyclotron frequency (rad/s)
omega_minus = 0.5 * (omega_c - np.sqrt(omega_c**2 - (2 * (omega_z**2))))  # Magnetron frequency (rad/s)
```

The amplitude of oscillation are given by:

```
# Amplitudes
r_plus = np.sqrt((((q/abs(q))*y0_velocity + omega_minus*x0_position)**2 + (x0_velocity - (q/abs(q))*omega_minus*y0_position)**2) /((omega_minus - omega_plus)**2))  # Amplitude of cyclotron motion (m)
r_minus = np.sqrt((((q/abs(q))*y0_velocity + omega_plus*x0_position)**2 + (x0_velocity - (q/abs(q))*omega_plus*y0_position)**2) /((omega_plus - omega_minus)**2))  # Amplitude of magnetron drift around trap centre (m)
r_z = np.sqrt(z0_position**2 + (z0_velocity**2)/(omega_z**2))  # Amplitude of axial motion
```

The phase constants (due to initial conditions) are:

```
# Phase constants
phi_plus = np.arccos(((q/abs(q))*y0_velocity + omega_minus*x0_position) / (r_plus*(omega_minus - omega_plus)))  # Phase constant for reduced cyclotron frequency
phi_minus = np.arccos(((q/abs(q))*y0_velocity + omega_plus*x0_position) / (r_minus*(omega_plus - omega_minus)))  # Phase constant for magnetron frequency
phi_z = np.arccos(z0_position/r_z)  # Phase constant for axial frequency
```

Using the variables defined above, we can determine the x, y and z coordinate of the particle at a given time by defining the 3 following functions:

```
def x(time):
    """returns x-coordinate position of particle
       Keyword arguments:
           time -- the time at which the x-coordinate of the particle is desired
    """
    result = r_plus * np.cos(omega_plus * time + phi_plus) + r_minus * np.cos(omega_minus * time + phi_minus)
    return result


def y(time):
    """returns y-coordinate position of particle
           Keyword arguments:
               time -- the time at which the y-coordinate of the particle is desired
    """
    result = r_plus * np.sin(omega_plus * time + phi_plus) + r_minus * np.sin(omega_minus * time + phi_minus)
    return result


def z(time):
    """returns z-coordinate position of particle
           Keyword arguments:
               time -- the time at which the z-coordinate of the particle is desired
    """
    result = r_z * np.cos(omega_z * time + phi_z)
    return result
```

With these functions, we generate an array consisting of the particle's position at specified time intervals defined by the variable "step". By default, the step is the period of one reduced cyclotron oscillation divided by 45 to ensure smooth animation. The "cycles" variable is the number of full magnetron cycles the animation will show before resetting. This can be increased or decreased as desired, although simulating a large number of cycles will take longer for the data to be computed. This data is then stored in an array which we will plot to form our animation.

```
# Generate data
step = 2 * np.pi * 1/omega_plus * 1/45  # Ensures that the time divisions are sufficiently small to generate trajectory of cyclotron motion
cycles = 10  # Number of magnetron cycles
```

```
n = int(np.ceil(2 * np.pi * 1/omega_minus * cycles * 1/step))  # Number of data points
data = np.empty((3, n))
for i in range(0, n):
    t = i * step  # time
    data[0, i] = x(t)
    data[1, i] = y(t)
    data[2, i] = z(t)
data = [data]
```

Then, we need to define an additional function which is required by matplotlib to create the animation.

```
def update_lines(n, dataLines, lines):
    for line, dat in zip(lines, dataLines):
        line.set_data(dat[0:2, :n])
        line.set_3d_properties(dat[2, :n])
    return lines
```

Finally, we configure the figure and animate our data.

```
# Attaching 3D axis to the figure
fig = plt.figure()
ax = p3.Axes3D(fig)

# Viewing position
ax.elev = 35.
ax.azim = 150.
ax.dist = 11.

lines = [ax.plot(dat[0, 0:1], dat[1, 0:1], dat[2, 0:1], color='#3498DB',  linewidth=1)[0] for dat in
↪   data]

# Setting the axes properties
ax.set_xlim3d([-1.25 * r_minus, 1.25 * r_minus])
ax.set_xlabel('X')

ax.set_ylim3d([-1.25 * r_minus, 1.25 * r_minus])
ax.set_ylabel('Y')

ax.set_zlim3d([-1.25 * r_z, 1.25 * r_z])
ax.set_zlabel('Z')

ax.set_title('Charged Particle Motion in Penning Trap')

# Creating the Animation object
line_ani = animation.FuncAnimation(fig, update_lines, n, fargs=(data, lines),
                                   interval=1, blit=True)

plt.show()
```

The viewing position can be changed by modifying the variables “ax.elev”, “ax.azim” and “ax.dist” in lines 6, 7 and 8 respectively. The speed of the animation can be increased or decreased by changing the “interval” argument in line 25. If required, the animation can be saved as an mp4 file by replacing plt.show() in line 28 with the following:

```
Writer = animation.writers['ffmpeg']
writer = Writer(fps=60, metadata=dict(artist='Me'), bitrate=1800)
line_ani.save('Charged Particle Motion in Penning Trap.mp4', writer=writer)
```

However, this process is typically very time consuming due to the large number of frames required to obtain a smooth animation.

4 Attributions

The paper on “Cyclotron frequency in a Penning trap” by Blaum group of Physikalische Insitute was referrenced in solving the equations in the theory portion of this homework[3]. The code was adapted from the example of a 3D animation plot on the matplotlib website[2]. The initial conditions and equations in the code were adapted from “A visualization of an ideal Penning Trap” by Dilyn Fullerton, Shiv Mittal, and Michael Stone.[4]

References

[1] Wikipedia: Penning trap
https://en.wikipedia.org/wiki/Penning_trap

[2] animation example code: simple_3danim.py
https://matplotlib.org/examples/animation/simple_3danim.html

[3] Cyclotron frequency in a Penning trap
https://www.physi.uni-heidelberg.de/Einrichtungen/FP/anleitungen/F47.pdf

[4] A visualization of an ideal Penning Trap
https://github.com/dilynfullerton/penning