Fundamentals of Plasma Physics, Nuclear Fusion and Lasers

ATHENS PROGRAMME

ipfn INSTITUTO DE PLASMAS E FUSÃO NUCLEAR

Single Particle Motion

Uniform and constant electromagnetic fields

Nuno R. Pinhão

2015, March

In this notebook we analyse the movement of individual particles under constant electric and magnetic fields integrating the equations of motion. We introduce the concepts of cyclotron frequency, Larmor radius and drift velocity. We finish generalizing the drift velocity for a general force.

1 Introduction

In this notebook we will use the Lorentz force equation,

$$\vec{F} = q\left(\vec{E} + \vec{v} \times \vec{B}\right) \qquad \text{(Lorentz force)}$$

and the Euler's formula

$$e^{i\theta} = \cos\theta + i\sin\theta. \qquad \text{(Euler formula)}$$

The subject of this notebook is covered in the bibliography in the following chapters:

- Chen[1]: chapter Two, section 2.2
- Nicholson[2]: chapter 2, section 2.2
- Bittencourt[3]: chapter 2
- Goldston[4]: chapter 2

The examples are prepared with the help of two scientific software packages, *Numpy/Scipy*[5] and *IPython*[6].

2 Movement without $\vec{E}$ field

Let's keep it simple: $\vec{B}(\vec{r},t) = B_0\vec{u}_\parallel$; $\quad \vec{v}_0 = v_{\perp,0}\vec{u}_\perp + v_{z,0}\vec{u}_\parallel$ and we take $\vec{u}_\parallel \equiv \vec{u}_z$.

- The Lorentz force is $\vec{F} = q\vec{v} \times \vec{B}$ and, as $\vec{F} \perp \vec{v} \Rightarrow$ constant kinetic energy, W.

Writing the Lorentz force in cartesian coordinates, we have

$$F_x = q\,(v_y B_z - v_z B_y) = q v_y B_0 \tag{1}$$
$$F_y = q\,(v_z B_x - v_x B_z) = -q v_x B_0 \tag{2}$$
$$F_z = q\,(v_x B_y - v_y B_x) = 0 \tag{3}$$

or,

$$\dot{v}_x = \frac{q}{m} v_y B_0 \tag{4}$$
$$\dot{v}_y = -\frac{q}{m} v_x B_0 \tag{5}$$
$$\dot{v}_z = 0 \tag{6}$$

Taking the derivative of any of the first two equations we obtain:

$$\ddot{v}_{x,y} + \left(\frac{q}{m} B_0\right)^2 v_{x,y} = 0$$

This is the homogeneous equation for a *harmonic oscilator*, with frequency $\boxed{\omega_c \equiv \frac{|q|B}{m}}$. We call it the **cyclotron frequency**.
Integrating again we obtain

$$x = x_0 - i(v_\perp/\omega_c)\,[\exp(i\omega_c t + i\delta) - \exp(i\delta)] \tag{7}$$
$$y = y_0 \pm (v_\perp/\omega_c)\,[\exp(i\omega_c t + i\delta) - \exp(i\delta)] \tag{8}$$
$$z = z_0 + v_{z,0} t \tag{9}$$

The quantity $\boxed{r_L \equiv \frac{v_\perp}{\omega_c}}$ is the **Larmor radius** or **gyro-radius**.

2.0.1 In conclusion:

- Uniform circular motion in $\perp$;
- $v_\parallel$ is constant $\Rightarrow$ uniform motion in $\parallel$;
- The frequency of the circular motion depends on the q/m ratio and the magnetic field intensity, B;
- The radius of the trajectory is the ratio between the module of the velocity in the plane perpendicular to the magnetic field, $v_\perp$, and the rotation frequency, ω_c.

2.1 Practice:

Let's represent the movement of two imaginary particles, with (q, m) values respectively $(-1, 1)$ and $(1, 10)$. For that we convert the Lorentz force equation in a system of first order differential equations,

$$\dot{\vec{r}}(t) = \vec{v}(t) \tag{10}$$
$$\dot{\vec{v}}(t) = \frac{q}{m}(\vec{E} + \vec{v}(t) \times \vec{B}) \tag{ode}$$

and we integrate this system to obtain the trajectories.
We start by importing some libraries...

```
In [1]: %matplotlib inline
        import numpy as np
```

…we define some values common to all simulations,

```
In [2]: global q, me, Mp, Bz                        # We share these values
        q = 1; me = 1; Mp = 10*me                   # Module of charge and masses
        B0 = np.array([0,0,1])                      #  Magnetic field
```

and write the (ode) system above in a function:

```
In [3]: def cteEB(Q, t, qbym, E0, B0):
            """Equations of movement for constant electric and magnetic fields.

            Positional arguments:
            Q -- 6-dimension array with values of position and velocity (x,y,z,vx,vy,v
            t -- time value (not used here but passed by odeint)
            qbym -- q/m
            E0, B0 -- arrays with electric and magnetic field components

            Return value:
            Array with dr/dt and dv/dt values."""

            v = Q[3:]

            drdt = v                                # Velocity
            dvdt = qbym*(E0 + np.cross(v,B0))       # Acceleration

            return np.concatenate((v,dvdt))
```

All we need now is to define initial values, and solve this system in time to obtain the trajectories. We use the *odeint* routine for the integration of first-order vector equations, from the *Scipy* package. [Technical note: This routine is a call to *lsoda* from the FORTRAN library *odepack*.]

```
In [4]: def computeTrajectories(func, E0=np.zeros(3), **keywords):
            """Movement of electron and ion under a constant magnetic field.

            Positional arguments:
            func -- the name of the function computing dy/dt at time t0
            Keyword arguments:
            E0 -- Constant component of the electric field
            All other keyword arguments are collected in a 'keywords' dictionary
            and specific to each func."""

            from scipy.integrate import odeint
            global q, me, Mp, B0

            # Initial conditions
            r0 = np.zeros(3)                    # Initial position
            if "vi" in keywords.keys():         # Initial velocity
                v0 = keywords["vi"]
            else:
                v0 = np.array([0,0,0])
```

```
    Q0 = np.concatenate((r0,v0))  # Initial values

    tf = 350; NPts = 10*tf
    t = np.linspace(0,tf,NPts)    # Time values

    # Integration of the equations of movement
    Qe = odeint(func, Q0, t, args=(-q/me,E0,B0))  # "electron" trajectory
    Qp = odeint(func, Q0, t, args=(q/Mp,E0,B0))   # "ion" trajectory

    return Qe, Qp
```

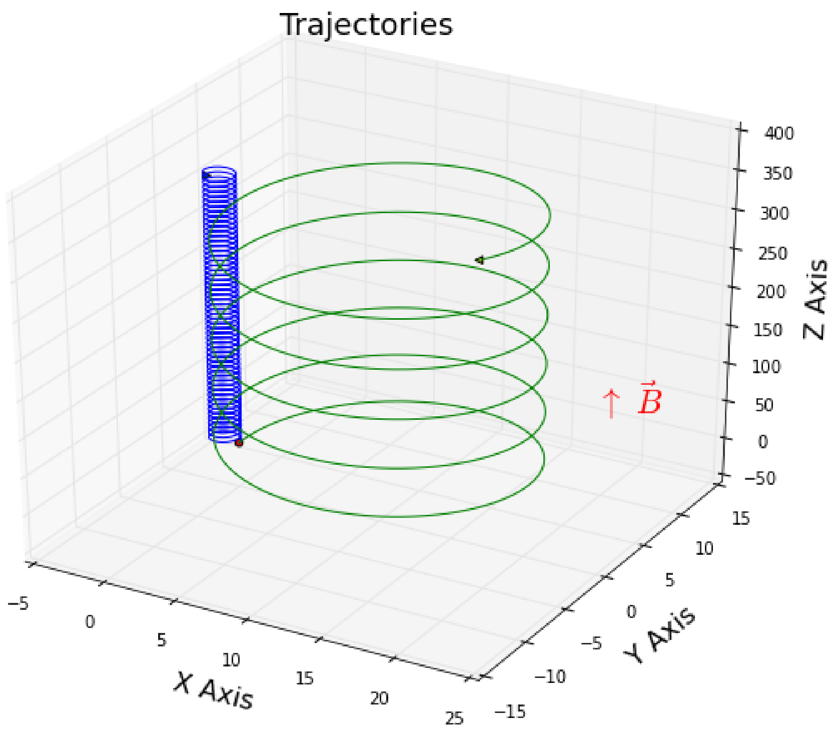
We also define functions to compute the cyclotron frequency, Larmor radius and to visualize the trajectories, marking the starting and final points, using a 3D plotting package:

```
In [5]: wc = lambda m: q*B0[2]/m                            # cyclotron frequency
        rL = lambda m: np.sqrt(v0[0]**2+v0[1]**2)/wc(m)  # Larmor radius

        def plotTrajectories(re,rp):
            """Plot the trajectories and Larmor radius"""
            import matplotlib.pyplot as plt
            from mpl_toolkits.mplot3d import Axes3D

            fig = plt.figure(figsize=(10,8))
            ax = fig.gca(projection='3d')
            # Legibility
            ax.set_title("Trajectories",fontsize=18)
            ax.set_xlabel("X Axis",fontsize=16)
            ax.set_ylabel("Y Axis",fontsize=16)
            ax.set_zlabel("Z Axis",fontsize=16)
            ax.text(17,15,0, "$\\uparrow\\, \\vec{B}$", color="red",fontsize=20)
            ax.scatter(re[0,0],re[0,1],re[0,2],c='red') # Starting point
            ax.plot(re[:,0],re[:,1],re[:,2])            # Electron trajectory
            ax.plot(rp[:,0],rp[:,1],rp[:,2])            # Ion trajectory
            # Final points
            ax.scatter(re[-1,0],re[-1,1],re[-1,2],c='green', marker='>')
            ax.scatter(rp[-1,0],rp[-1,1],rp[-1,2],c='yellow', marker='<')
```

And we are ready to test our code:

```
In [7]: vz = 1                                   # Initial z-velocity
        v0 = np.array([0,1,vz])                  # Initial particle velocity

        # we can already compute the cyclotron frequencies and Larmor radius
        print('Cyclotron frequencies    Larmor radius\n  we= {}, wp= {}   \
          rLe= {}, rLp= {}'.format(wc(me),wc(Mp),rL(me),rL(Mp)))

        # And now the trajectories...
        re, rp = computeTrajectories(cteEB, vi=v0)
        plotTrajectories(re,rp)
```

```
Cyclotron frequencies    Larmor radius
  we= 1.0, wp= 0.1    rLe= 1.0, rLp= 10.0
```



3 $\vec{E} \neq \vec{0}$, constant

Let's take $\vec{E} = E_{\perp,0}\vec{u}_\perp + E_{\parallel,0}\vec{u}_z$. From the Lorentz force equation, we obtain

$$\dot{v}_x = \frac{q}{m}(E_x + v_y B_0) \tag{11}$$

$$\dot{v}_y = \frac{q}{m}(E_y - v_x B_0) \tag{12}$$

$$\dot{v}_z = \frac{q}{m}E_z \tag{13}$$

In the direction parallel to $\vec{B}$, we just have the free-fall in the electric field:

$$v_\parallel = \frac{q}{m}E_\parallel t + v_{\parallel,0}.$$

In the plane $\perp$ to $\vec{B}$, making the substitution $v'_x = v_x - E_y/B_0, \quad v'_y = v_y + E_x/B_0$, we obtain for v'_x, v'_y the same equations as in the previous case! Or if we want to proceed in a more formal way, we take again the derivative of any of the first two equations obtaining the *inhomogeneous equation for a harmonic oscillator*,

$$\ddot{v}_{x,y} + \omega_L^2 v_{x,y} = \omega_L^2 \frac{E_{y,x}}{B},$$

the solution of which is the solution for the *homogeneous* case plus a particular solution $\Rightarrow \vec{v}_\perp$ has a Larmor movement with a drift: $v_\perp = v_\bigcirc + v_d$.

How to compute v_d?

If we average the Lorentz force over many gyroperiods, the average acceleration is *zero* and the only velocity component left is v_d:

$$0 = \frac{q}{m}(\vec{E} + \vec{v}_d \times \vec{B}).$$

Taking the cross product with $\vec{B}$ and using the vector formula in the appendix, we finally obtain

$$\vec{v}_d = \frac{\vec{E} \times \vec{B}}{B^2}$$

3.1 Summary

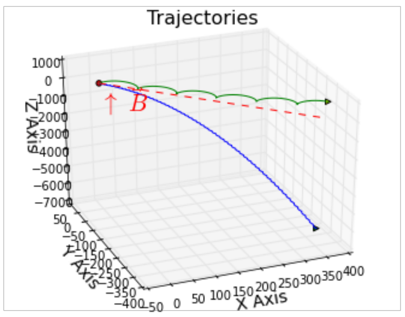
- The drift is $\perp$ both to $\vec{E}$ and $\vec{B}$;
- On the same direction for electrons and ions $\Rightarrow$ no net current!;
- The drift is independent of m, q and $v_\perp$.
  - Exercise: What is the $\vec{E}_\perp$ field in an inertial frame moving with $\vec{v}_d$? (*Hint*: use the Lorentz transformation for an electromagnetic field.)

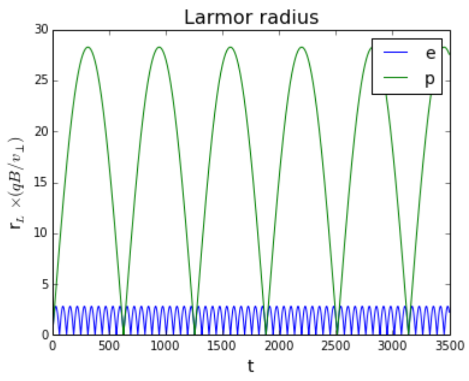
3.2 Practice:

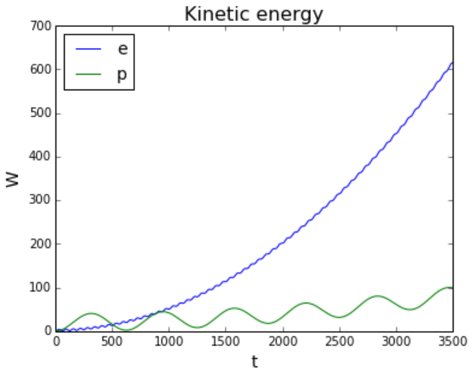
We can easily extend the previous example to include a constant electric field. To confirm that the guiding center moves along the direction of v_d, we draw a red line along v_d. It is also interesting to see what happens with the kinetic energy and the Larmor radius and we add two more figures for these values.

We include one more library to allow us to interact with the script and we leave the rendering of the figure to an external script.

```
In [8]: from IPython.html.widgets import interact
        import plotEB

        modv = lambda v: np.sqrt((v[:,0])**2+(v[:,1])**2)  # v-perpendicular
        v2 = lambda v: v[:,0]**2+v[:,1]**2+v[:,2]**2

        def crossEB(Exy=0, Ez=0, angle=320):
            global q, me, Mp, B0
            E0 = np.array([Exy,Exy,Ez])

            re, rp = computeTrajectories(cteEB, E0)      # We use the same routine!
            vd = np.cross(E0,B0)/np.dot(B0,B0)           # vd = (EXB)/(B|B)
            t = np.arange(re.shape[0])/10
            rd = np.array([t,vd[0]*t,vd[1]*t]).T         # drift trajectory
            # Larmor radius
            rLe = me/q*modv(re[:,3:])/B0[2]; rLp = Mp/q*modv(rp[:,3:])/B0[2]
            We = me/2*v2(re[:,3:]); Wp = Mp/2*v2(rp[:,3:]) # Kinetic energy
            # Plot the trajectories and Larmor radius
            plotEB.plot3d(re, rp, rd, rLe, rLp, We, Wp, angle)

        dummy = interact(crossEB, Exy=(0,1), Ez=(0,0.2), angle=(180,360))
```







4 General force, $\vec{F}$

The result above can be generalized for any constant and uniform force such that $\vec{F} \cdot \vec{B} = 0$:

$$\vec{v}_d = \frac{\vec{F} \times \vec{B}}{qB^2}$$

In particular, for the gravitational field, we have a drift $v_g = \frac{m}{q}\frac{\vec{g}\times\vec{B}}{B^2}$. In this case the direction of drift depends on the signal of $q \Rightarrow$ for positive and negative charges, we have a current!

The velocity of any particle can be decomposed into three components:

$$\vec{v} = \vec{v}_{\|} + \vec{v}_d + \vec{v}_L$$

5 Appendix - Useful vector formulae

$$\vec{A} \times (\vec{B} \times \vec{C}) = (\vec{A} \cdot \vec{C})\vec{B} - (\vec{A} \cdot \vec{B})\vec{C}$$

References

[1] Francis Chen. *Single-particle Motions*, chapter 2, pages 19–52. Plenum, 1974.

[2] Dwight R. Nicholson. *Single Particle Motion*, chapter 2, pages 17–36. Wiley series in plasma physics. John Wiley & Sons, 1983.

[3] J. A. Bittencourt. *Charged particle motion in constant and uniform electromagnetic fields*, chapter 2, 3, pages 33–89. Springer, 2004.

[4] R. L. Goldston and P. H. Rutherford. *Single-particle Motion*, pages 21–68. IOP Publishing, Bristol and Philadelphia, 1995.

[5] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online].

[6] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.