Motion of Charged Particle in Electromagnetic Field

Bhavya Sharma
Class $12^{th} - B$
Delhi Public School, Koba-Adalaj Road
Ambapur, Gandhinagar, Gujarat, India -382421

under supervision of

Dr. Igor Kaganovich
Princeton Plasma Physics Laboratory
Princeton University, NJ-08540, USA
October 5, 2022

Abstract

Motion of a charged particle in classical sense is solely governed by Lorentz force [$\mathbf{F} = \mathbf{E} + q(\mathbf{v} \times \mathbf{B})$]. While this differential equation of motion can be solved theoretically for simple systems, it becomes harder and challenging to solve this for complex setup of fields. We attempt to computationally obtain the trajectory of this particle in a given electric and magnetic field by approximately solving this ODE using numerical methods. In this project we employed Euler, 2nd order Runge-Kutta and 4th order Runge-Kutta. We studied the accuracy of these methods and the trend of errors generated in respective motions. We found that Runge-Kutta methods are far better than Euler method, albeit computationally expensive. Also that we cannot run this simulation for longer times using these methods because of error accumulation. We need a better method which does not accumulate much error.

1 Numerical Methods

1.1 Euler Method

Consider the first-order differential equation

$$\dot{x} = f(t, x)$$

Using Taylor Expansion,

$$\begin{aligned} x_{n+1} &= x(t_n + \Delta t) \\ &= x(t_n) + \Delta t \dot{x}(t_n) + O(\Delta t^2) \\ &= x(t_n) + \Delta t f(t_n, x_n) + O(\Delta t^2) \end{aligned} \tag{1.1}$$

Ignoring second and higher orders of Δt

$$\boxed{x_{n+1} = x(t_n) + \Delta t f(t_n, x_n)}$$

Local error: $O(\Delta t^2)$
Global error (when integrated over time T): $O(\Delta t)$

For the case of Lorentz force,

$$\begin{aligned} \mathbf{a}_n &= \frac{q}{m}(\mathbf{E} + \mathbf{v}_n \times \mathbf{B}) \\ \mathbf{v}_{n+1} &= \mathbf{v}_n + \Delta t \mathbf{a}_n \\ \mathbf{r}_{n+1} &= \mathbf{r}_n + \Delta t \mathbf{v}_n \end{aligned} \tag{1.2}$$

where, $\mathbf{a}_n, \mathbf{v}_n, \mathbf{r}_n$ are acceleration, velocity and position respectively of the particle at n^{th} time step.

1.2 2nd order Runge-Kutta

$$\dot{x} = f(t, x)$$

Original algorithm,

$$\begin{aligned} k_1 &= \Delta t f(t_n, x_n) \\ k_2 &= \Delta t f(t_n + \Delta t, x_n + k_1) \\ x_{n+1} &= x_n + \frac{1}{2}(k_1 + k_2) \end{aligned} \tag{1.3}$$

For the case of Lorentz force,

$$\begin{aligned} k_1 &= \Delta t \frac{q}{m}(\mathbf{E} + \mathbf{v}_n \times \mathbf{B}) \\ k_2 &= \Delta t \frac{q}{m}(\mathbf{E} + (\mathbf{v}_n + k_1) \times \mathbf{B}) \\ \mathbf{v}_{n+1} &= \mathbf{v}_n + \frac{1}{2}(k_1 + k_2) \\ kk1 &= \Delta t \mathbf{v}_n \\ kk2 &= \Delta t(\mathbf{v}_n + k_1) \\ \mathbf{r}_{n+1} &= \mathbf{r}_n + \frac{1}{2}(kk1 + kk2) \end{aligned} \tag{1.4}$$

1.3 4TH ORDER RUNGE-KUTTA

$$\dot{x} = f(t, x)$$

Original algorithm,

$$\begin{aligned} k_1 &= \Delta t f(t_n, x_n) \\ k_2 &= \Delta t f(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_1) \\ k_3 &= \Delta t f(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_2) \\ k_4 &= \Delta t f(t_n + \Delta t, x_n + k_3) \\ x_{n+1} &= x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{1.5}$$

For the case of Lorentz force,

$$\begin{aligned} k_1 &= \Delta t \frac{q}{m}(\mathbf{E} + \mathbf{v}_n \times \mathbf{B}) \\ k_2 &= \Delta t \frac{q}{m}(\mathbf{E} + (\mathbf{v}_n + \frac{1}{2}k_1) \times \mathbf{B}) \\ k_3 &= \Delta t \frac{q}{m}(\mathbf{E} + (\mathbf{v}_n + \frac{1}{2}k_2) \times \mathbf{B}) \\ k_4 &= \Delta t \frac{q}{m}(\mathbf{E} + (\mathbf{v}_n + k_3) \times \mathbf{B}) \\ \mathbf{v}_{n+1} &= \mathbf{v}_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ kk1 &= \Delta t \mathbf{v}_n \\ kk2 &= \Delta t(\mathbf{v}_n + \frac{1}{2}k_1) \\ kk3 &= \Delta t(\mathbf{v}_n + \frac{1}{2}k_2) \\ kk4 &= \Delta t(\mathbf{v}_n + k_3) \\ \mathbf{r}_{n+1} &= \mathbf{r}_n + \frac{1}{6}(kk_1 + 2kk_2 + 2kk_3 + kk_4) \end{aligned} \tag{1.6}$$

1.4 Python code for these methods

Following is the Python code used for these methods.

```
class create_particle:
    def __init__(self, v, start, q, m, h):
        self.v = v
        self.pos = start
        self.q = q
        self.a = np.array([0.,0.,0.])
        self.m = m
        self.h = h

    def euler_move(self):
        self.a = self.q * (E+np.cross(self.v, B)) / self.m
        self.pos += self.v * self.h
        self.v += self.a * self.h

    def RK2_move(self):
        self.a = self.q * (E+np.cross(self.v, B)) / self.m
        k1 = self.a * self.h
        kk1 = self.v * self.h
        k2 = self.h* self.q *(E+np.cross(self.v + k1, B)) / self.m
        kk2 = (self.v+k1) * self.h
        self.pos +=  (kk1+kk2)/2
        self.v += (k1+k2)/2

    def RK4_move(self):
        self.a = self.q *(E+np.cross(self.v, B)) / self.m
        k1 = self.a * self.h
        kk1 = self.v * self.h
        k2 = self.h * self.q *(E+ np.cross(self.v + k1 / 2, B)) / self.m
        kk2 = (self.v + k1/2)* self.h
        k3 = self.h * self.q *(E+ np.cross(self.v + k2 / 2, B)) / self.m
        kk3 = (self.v + k2/2)* self.h
        k4 = self.h * self.q * (E+np.cross(self.v + k3, B)) / self.m
        kk4 = (self.v + k3) * self.h
        self.pos += (kk1 + 2*kk2 + 2*kk3 + kk4) /6
        self.v += (k1 + 2*k2 + 2*k3 + k4)/6
```

Listing 1: Python class for particle

1.5 Analysis of Numerical Methods

We compare the accuracy of the following models in a simple setup of constant and uniform magnetic field. We obtain graphs for multiple time steps.

Initial Conditions:
E = (0,0,0)
B = (0,0,1)
v = (1,0,0)
r = (0,0,0)

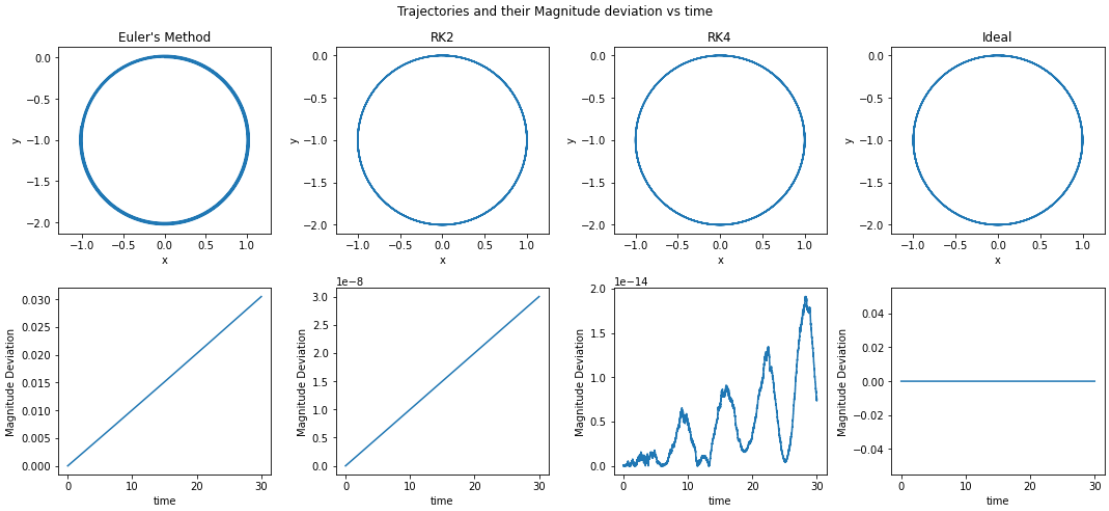


Figure 1.1: $\Delta t = 0.002$s

q = 1
m = 1

Ideally the trajectory should be a perfect circle. We plot trajectory using each of these methods and magnitude deviation in trajectory with time ($|\mathbf{r}_n - \mathbf{r}_{\text{ideal}}|$)

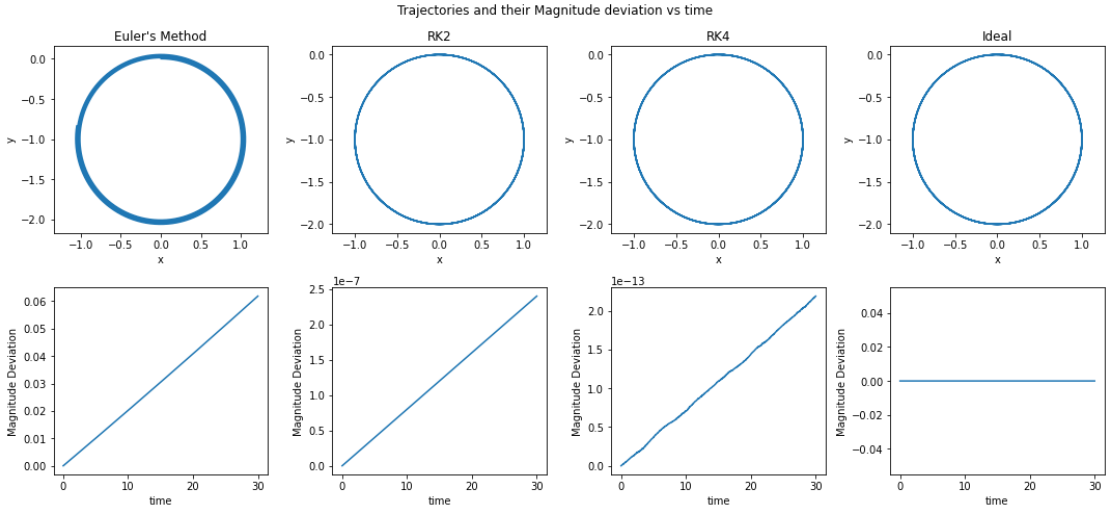


Figure 1.2: $\Delta t = 0.004$s

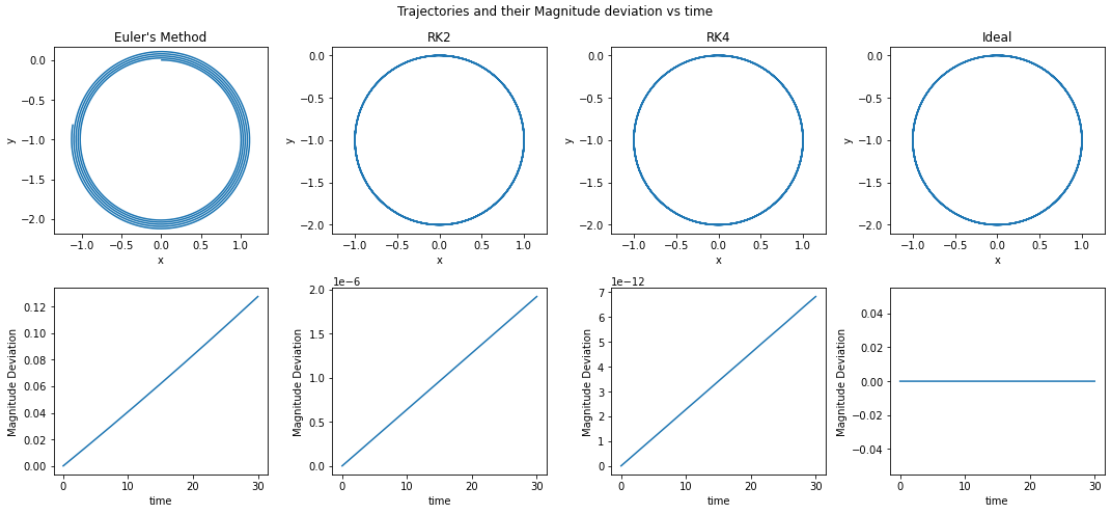


Figure 1.3: $\Delta t = 0.008$s

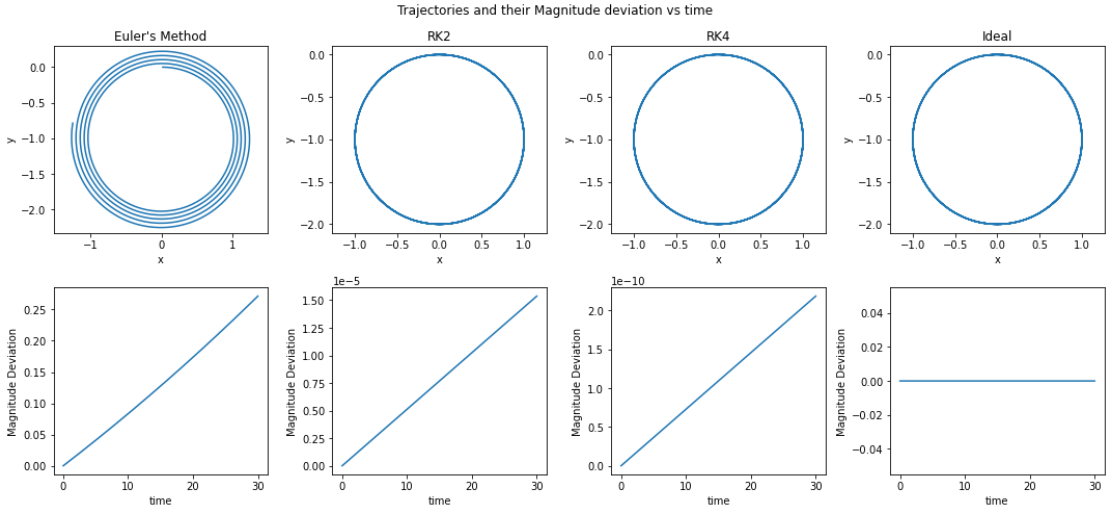


Figure 1.4: $\Delta t = 0.016$s

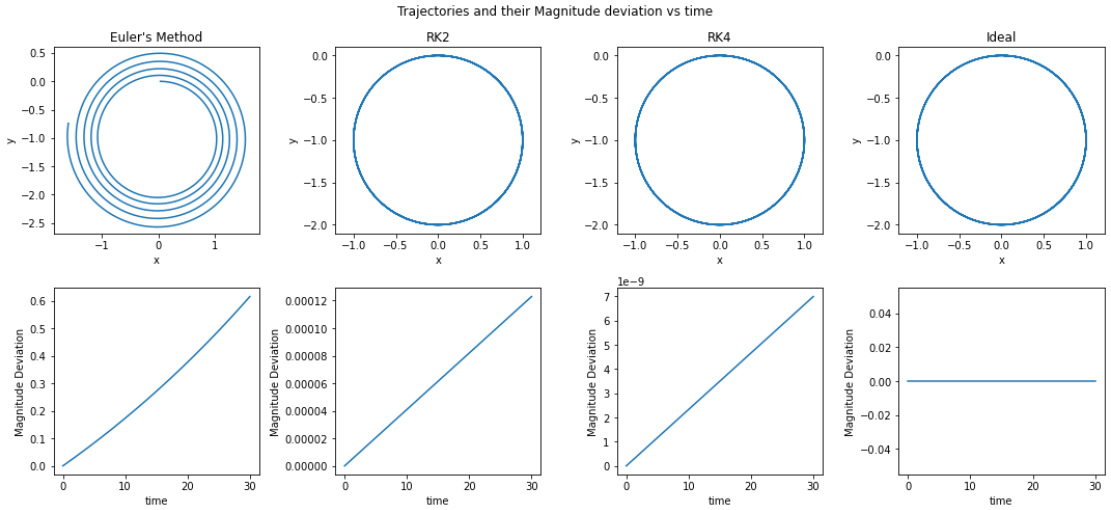


Figure 1.5: $\Delta t = 0.032$s

OBSERVATIONS

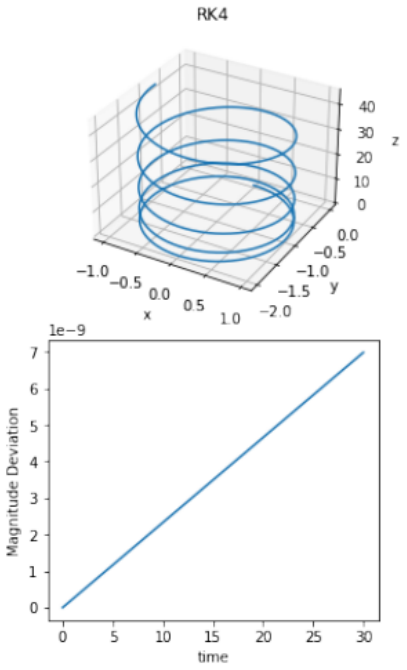
- Error increases with increasing time step (Δt) in all three methods.
- With sufficiently small time step, and over 30s duration, Euler's method gives deviation of the order 10^{-2}, 2nd order Runge kutta gives 10^{-8} - 10^{-6} while 4th order Runge kutta gives 10^{-14} - 10^{-12}, where our working values are of the order 1. Hence we conclude that 4th order Runge kutta gives much better accuracy than 2nd order Runge kutta. While both are far better than Euler's.
- In all three methods error accumulate linearly with time. This is a critical property which limits us on the duration of simulation.

FURTHER POSSIBILITIES We need a method which does not accumulate enough error within our working time. Boris Push method fulfills this need.

2 PARTICLE DYNAMICS

We know that a moving charged particle is affected by magnetic and electric forces. Various combination of uniform and non-uniform fields give rise to ordered and controlled motion of particles which can be utilised in the controlling plasma.
Using 4th order Runge-Kutta, we simulate the motion of a particle in some special setup of electromagnetic fields.

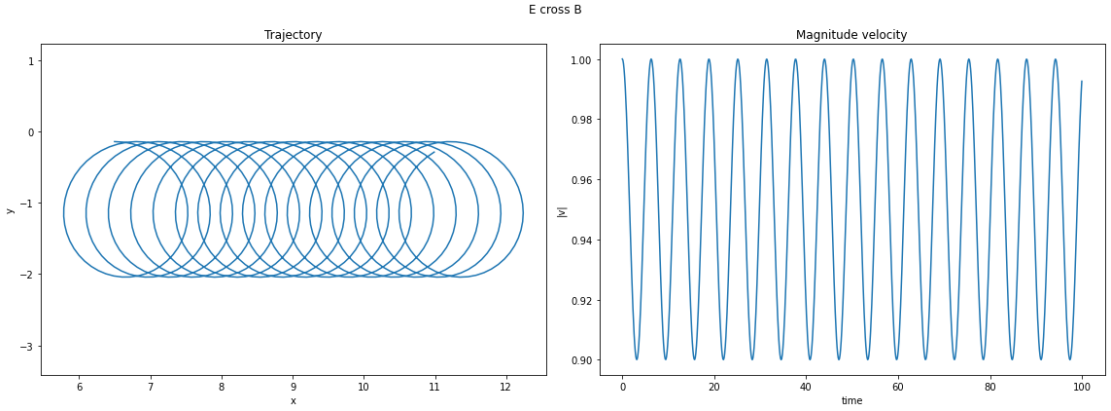
2.1 **E** || **B**



Initial Conditions:
E = (0,0,0.1)
B = (0,0,1)
v = (1,0,0)
q = 1
m = 1
time-step: h = 0.032s

As expected, particle is accelerated along the electric field in this case.

2.2 E × B DRIFT



Initial Conditions:

E = (0,0.05,0)

B = (0,0,1)

v = (1,0,0)

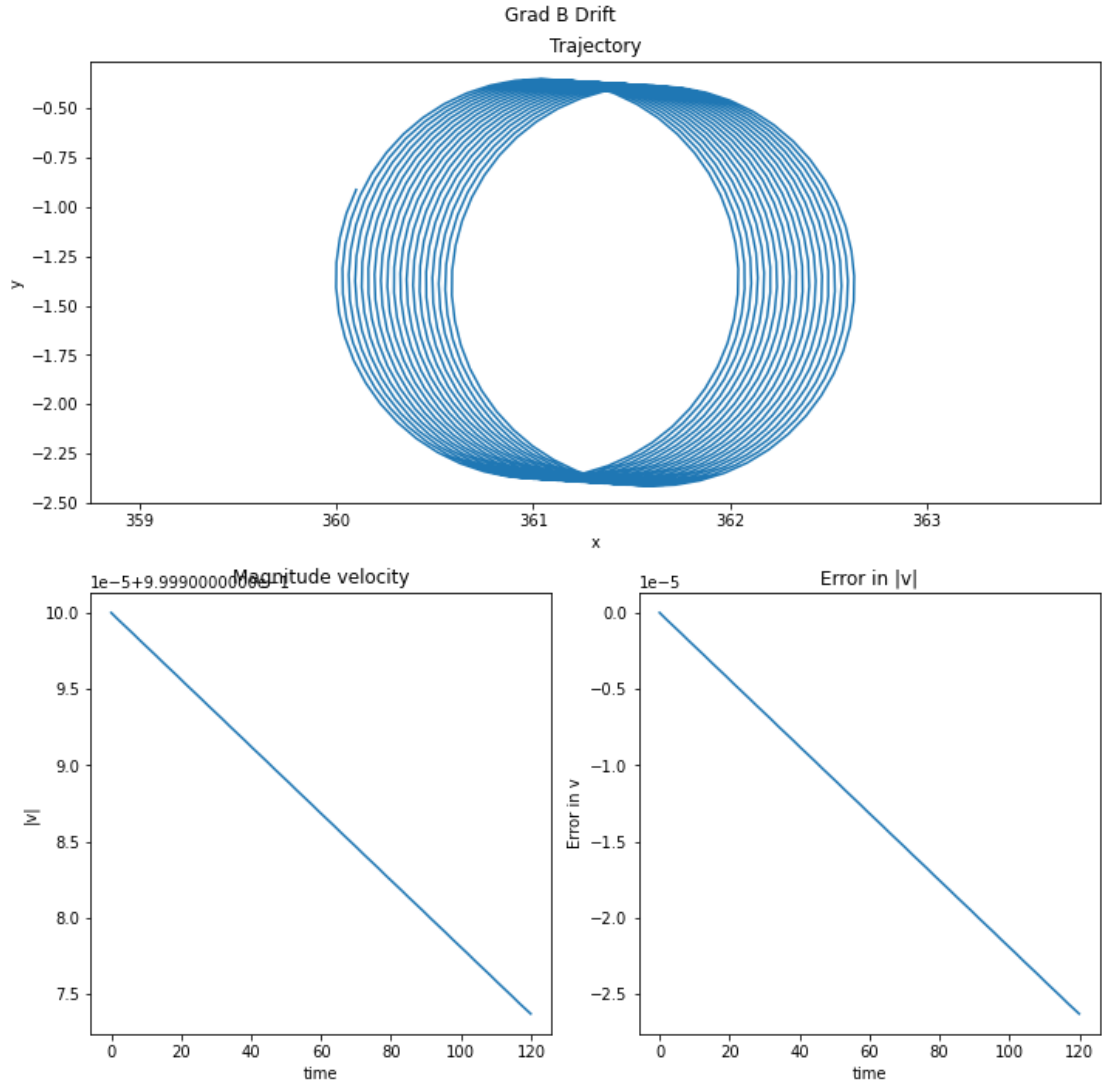
q = 1

m = 1

time-step: h = 0.128s

Under such a setup with both Electric and Magnetic Field, a drift is generated in the **E** × **B** direction. This can be easily visualized in above figure.

2.3 $\mathbf{B} \perp \nabla \mathbf{B}$: $\nabla \mathbf{B}$ Drift



Initial Conditions:
E = (0,0,0)
B = (0,0,1+0.01y)
v = (1,0,0)
q = 1
m = 1
time-step: h = 0.128s

Adding a gradient in Magnetic Field perpendicular to the field produces a drift in $\mathbf{B} \times \nabla \mathbf{B}$ direction.

3 Conclusion

Numerical calculation provides us an advantage of obtaining results in any complex combination of equations however with an added error. We explored three different numerical methods namely Euler, 2nd order Runge-Kutta and 4th order Runge-Kutta. 4th order Runge Kutta produced the least error amongst the three. Still the error accumulates with time.

We obtained trajectories for some setups of fields. Results match our theoretical expectations.

4 Further

1. As discussed earlier, we need a method which does not accumulate error with time. Boris Push is such a method. This could not be covered within the scope of this project.
2. We can obtain trajectories for even complex setups of fields such as the ones generating curvature drift, magnetic mirrors, adiabatic compression etc.

5 Bibliography

1. R. Chasnov, J., n.d. Numerical Methods. [online] Math.hkust.edu.hk. Available at: <https://www.math.hkust.edu.hk/ machas/numerical-methods.pdf> [Accessed 11 September 2022].
2. Chen, F., n.d. Introduction to plasma physics.