Engineering Electromagnetic Theory Laboratory 4

Gan Yuhao

May 21, 2024

Contents

1 Introduction **5**

2 Implement **7**

2.1 Declarations 7

2.2 Storage Structure 8

2.3 Simulation 9

3 Result and Analysis **11**

3.1 Simulation Results 11

3.2 Effect of Different dt 11

4 Conclusion and Inspiration **19**

1

Introduction

This is the report for Engineering Electromagnetic Theory Lab 4.

In this experiment we will use MATLAB to analyze the magnetic focusing phenomenon. For a beam of charged particles with small angle of divergence, given the same velocity component at the direction of the magnetic field B, their trajectory will have the same screw pitch. After a period, they will converge at another point. The phenomenon that the diverged charged particles focus at one point is similar to the phenomenon that lens can let the light beam focus at one point. Therefore, it is called as magnetic focusing.

Conditions for magnetic focusing:

1. The charged particles have similar initial velocity v;

2. The angle between v and B is sufficiently small so that each particle will do helicalmotion.

Our situation is that:

16 charges, they have equal mass $m = 0.02kg$ and each carries $q = 0.0016C$. The initial velocities are the same: $\vec{r}(1) = 0$(at the origin of the coordinate). Electric field: $E = 0$; magnetic flux density: $\vec{B} = 8\vec{a_z}Wb/m^2$. These 16 charges' initial velocities along z-axis are equal: $v_z(1) = 10m/s$. Their initial velocities along x-axis

and y-axis can be expressed as: $v_x(0) = 0.1sin(k\pi/8)m/s$, $v_y(0) = 0.1cos(k\pi/8)m/s$, where k = 0, 1, 2, ... , 15.

2

Implement

2.1 Declarations

First set the initial parameters of the simulation scene.

```
%% Clear
clear;
clc;
%% Initialization
% Basic Parameters
m = 0.02;
q = 0.016;
dt = 0.0001;
T = 5;
E = [0, 0, 0];
B = [0, 0, 8];
% Particles
n = 16;
k = 0 : n - 1;
v = 0.1 * sin(k' * pi / 8) * [1, 0, 0] + 0.1 * cos(k' * pi / 8) * [0, 1, 0] + ones(n, 1) * [0,
↪  0, 10];
r = zeros(n, 3);
% Recording
R = zeros(1, n * 3); % [x_0, x_1, ..., x_(n-1), y_0, ..., y_(n-1), z_0, ... , z_(n-1); ...]
```

2.2 Storage Structure

In order to simplify the code and make full use of the performance of MATLAB, we pack the parameters of each particle into a matrix for batch processing. For example, the instantaneous velocity of 16 particles is represented using the matrix v, which contains 16 rows and 3 columns, each row being the instantaneous velocity of the corresponding particle:

$$\mathbf{v} = \begin{bmatrix} \overrightarrow{v_0} \\ \overrightarrow{v_1} \\ \vdots \\ \overrightarrow{v_{n-1}} \end{bmatrix} = \begin{bmatrix} v_{x(0)} & v_{y(0)} & v_{z(0)} \\ v_{x(1)} & v_{y(1)} & v_{z(1)} \\ \vdots & \vdots & \vdots \\ v_{x(n-1)} & v_{y(n-1)} & v_{z(n-1)} \end{bmatrix} \tag{2.1}$$

The same is true for the representation of instantaneous displacement r, external force F, etc.

It is worth noting that we know that:

$$\vec{F} = q(\vec{E} + \vec{v} \times \vec{B}) \tag{2.2}$$

The problem if using the above representation is that MATLAB's fork product calculation function cross does not support simultaneous operations on multiple rows of the matrix.

The solution is to use a fork multiplication matrix and just convert the fork multiplication to matrix multiplication.

Whereas our magnetic flux density $\vec{B}$ is determined, the number of velocity vectors $\vec{v}$ is equal to the number of particles. So instead of calculating $\vec{v} \times \vec{B}$, during convenience, we calculate $-\vec{B}\times\vec{v}$. This way we can turn $\vec{B}$ into a fork multiplication matrix:

$$\mathbf{B}_\times = \begin{bmatrix} 0 & -B_z & B_y \\ B_z & 0 & -B_x \\ -B_y & B_x & 0 \end{bmatrix} \tag{2.3}$$

Taking into account the storage of the velocity matrix, we can transpose the order and transpose the fork multiplication matrix to obtain the equivalent result:

```
%% Simulation
B_X = [0, -B(3), B(2); B(3), 0, -B(1); -B(2), B(1), 0];
for t = 0 : dt : T
    F = q * (E - v * B_X'); % F = q * (E + v x B)
    v = v + F / m * dt;
    r = r + v * dt;
    R = [R; reshape(r, 1, n * 3)];
end
```

In addition to that, we need to store the instantaneous displacement of each particle at each step of the simulation, considering that the trajectory line needs to be drawn at the end.

Still for convenience, we use the reshape function to reduce the instantaneous displacement matrix of the column main order to one row. The compressed matrix is then inserted as a row at the end of the record matrix R. Finally, we use the matrix slicing operation when plotting with plot3, and just take out the component information in three directions.

After the above operation, we can get a very streamlined and intuitive code. See the code for details of the implementation.

2.3 Simulation

The simulation and plotting parts of the code add up to just a little bit as follows.

```
%% Simulation
B_X = [0, -B(3), B(2); B(3), 0, -B(1); -B(2), B(1), 0];
for t = 0 : dt : T
    F = q * (E - v * B_X'); % F = q * (E + v x B)
    v = v + F / m * dt;
    r = r + v * dt;
    R = [R; reshape(r, 1, n * 3)];
end

%% Plotting
figure(1);
hold on, grid on;
plot3(R(:, 1:n), R(:, n+1:2*n), R(:, 2*n+1:3*n));
title(["Particle Motion Trajectories (dt = " + dt + "s, T = " + T + "s)", "(Gan Yuhao,
↪  12211629)"]);
xlabel("x (m)"), ylabel("y (m)"), zlabel("z (m)");
daspect([0.5,0.5,50]);
saveas(1,"dt = "+dt+"s"+", T = "+T+"s.png")
```

Without going into details, it is a very intuitive step-by-step simulation process.

3

Result and Analysis

3.1 Simulation Results

Improved accuracy and simulation duration (Figure 2).

3.2 Effect of Different dt

When we use different dt, it means that the simulation has different step sizes. the closer the dt is to 0, the higher the accuracy and the smaller the cumulative error.

For example in Figure 3, the trajectory can be seen to gradually spread outward along the z-axis

A little more obvious example is in Figure 4

We modify the step size and observe the top view of the trajectory at different step sizes (Figure 5, 6 and 7).

We can clearly observe that the smaller the dt the better the simulation is, but at the same time the time taken for the simulation is also greatly increased.

Therefore, we need to choose the appropriate dt and obtain the balance.

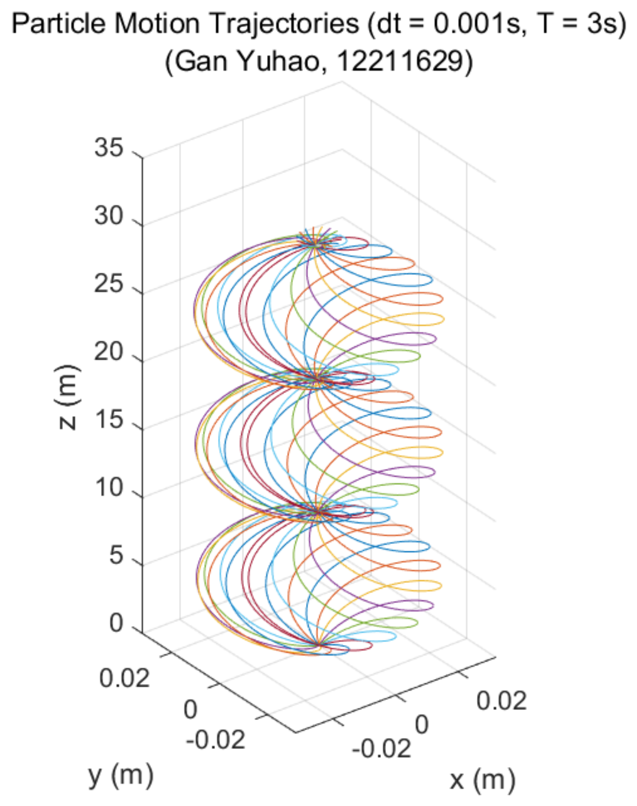

Figure 3.1: Particle Motion Trajectories (dt = 0.001s, T = 3s)

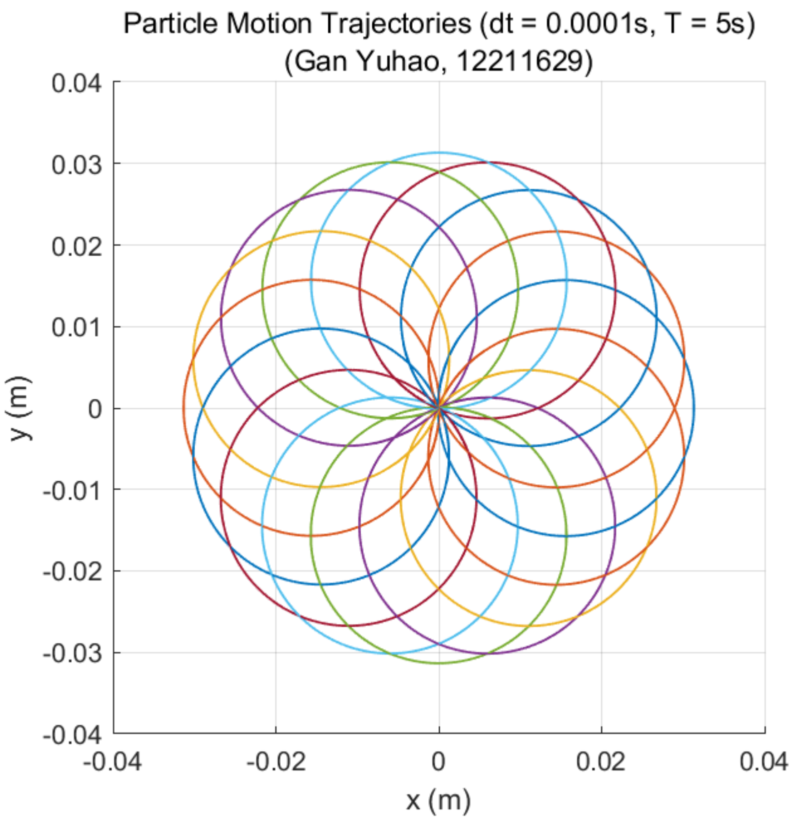

Figure 3.2: Particle Motion Trajectories (dt = 0.0001s, T = 5s)

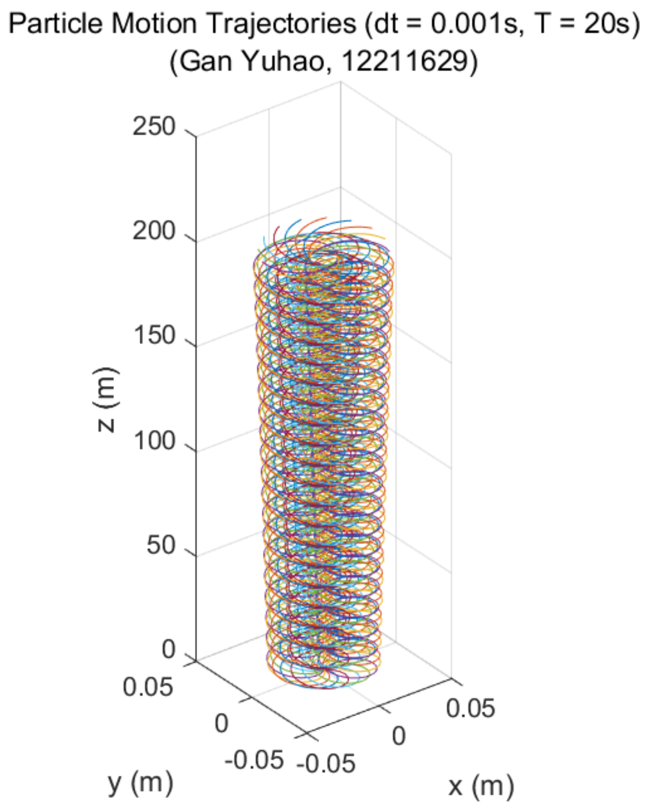

Figure 3.3: Particle Motion Trajectories (dt = 0.001s, T = 20s)

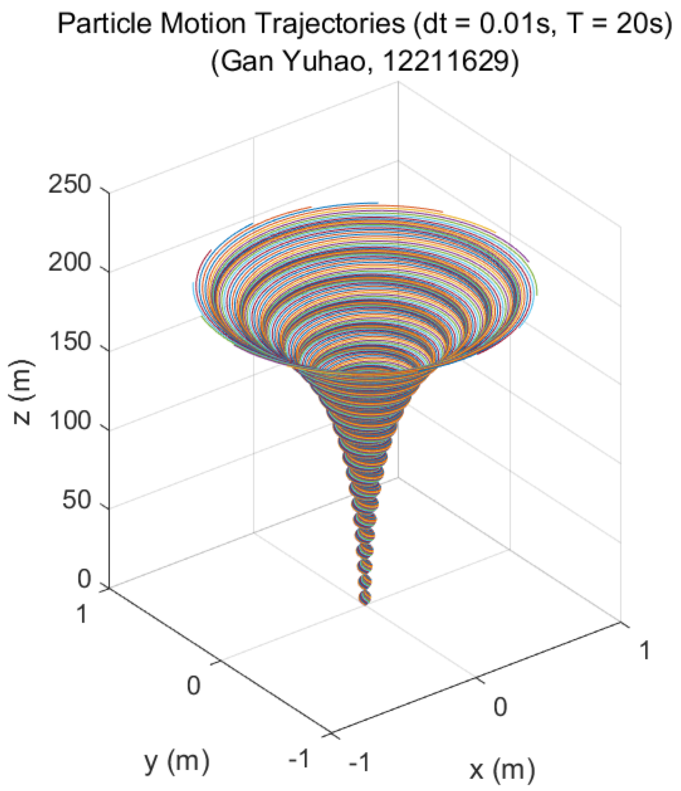

Figure 3.4: Particle Motion Trajectories (dt = 0.01s, T = 20s)

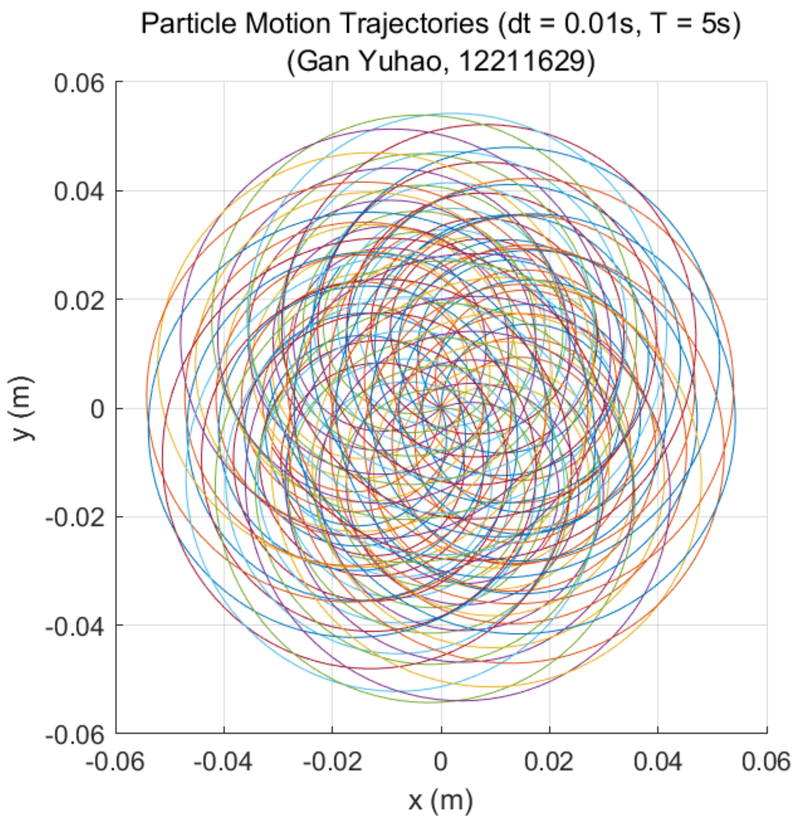

Figure 3.5: Particle Motion Trajectories (dt = 0.01s, T = 5s)

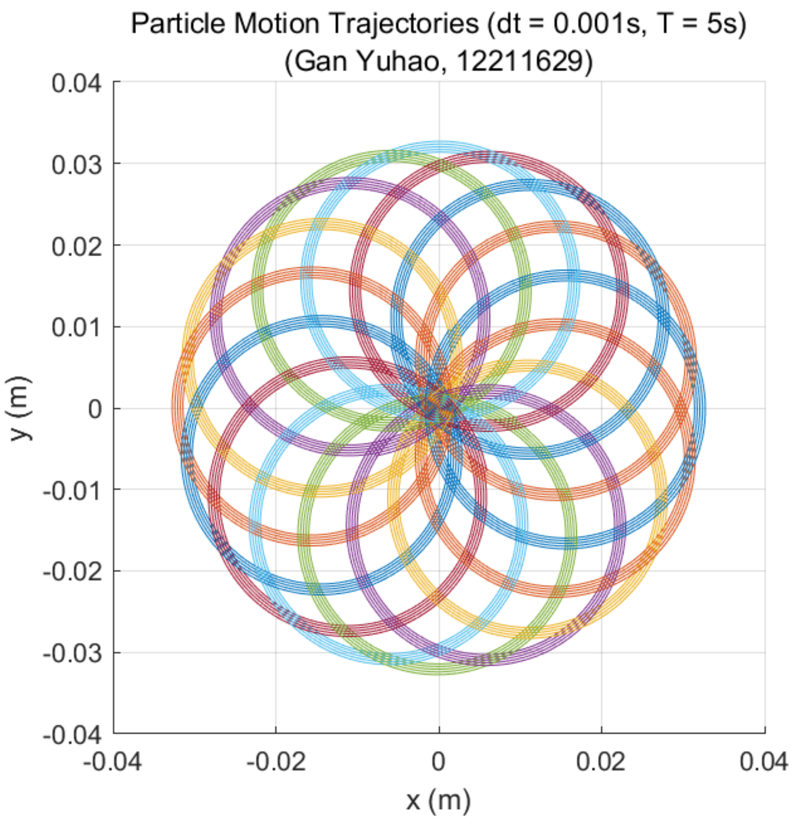

Figure 3.6: Particle Motion Trajectories (dt = 0.001s, T = 5s)

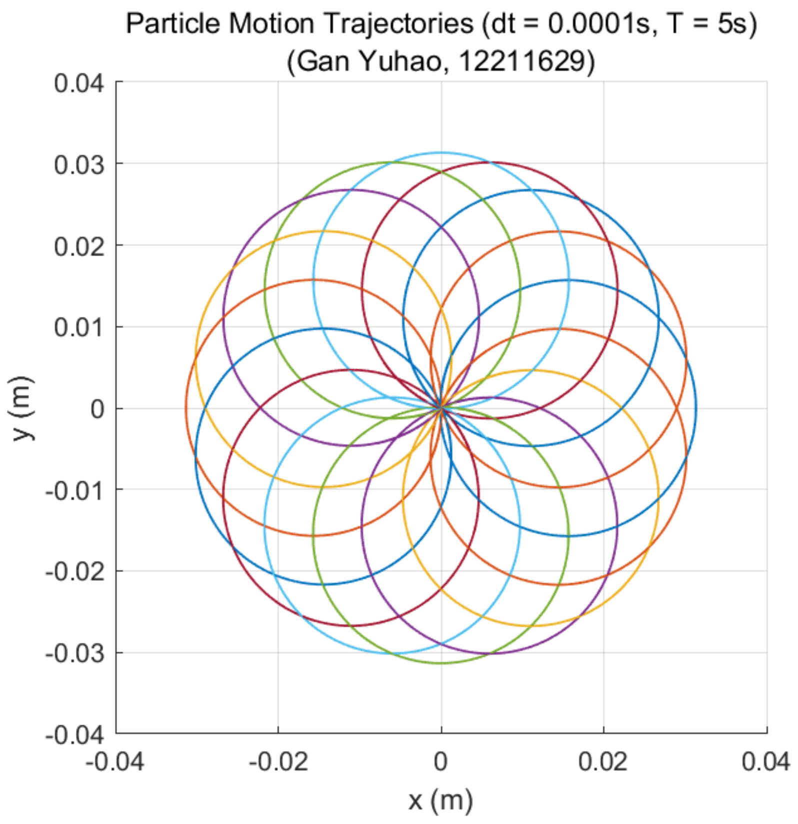

Figure 3.7: Particle Motion Trajectories (dt = 0.0001s, T = 5s)

4

Conclusion and Inspiration

In this experiment, we observed the motion of particles with different initial velocities in a magnetic field.

And, we also realized that the step size of numerical simulations can have a significant impact on the cumulative error of experimental results, and performed a comparative analysis.