Assignment 4

Rajdeep Pinge
201401103

Q1. Charge Particle trajectories under Lorentz force: Write a general MATLAB code to reproduce charge particle motions under Lorentz force in the following cases (as shown in Lecture-15 slide, choice of correct initial conditions is important to reproduce the trajectories). Report on the initial conditions and the rationale behind observed motion. Support your answer with several supporting graphs. (Try different 3D plotting schemes in MATLAB for better visualization, other than "plot3" as discussed in the class). Analyze the motion for t=0 to a reasonable value of t=t_final.

(i) Static and uniform B field. (for +ve and –ve charges)
(ii) Static and uniform E and B. (ExB drift) (what happens when v=v0x; B=B0z; and E=E0y; and v0=E0/B0).
(iii) Static and non-uniform B field (grad B drift)
(iv) Static and uniform B, and under gravitational force (for different mass)

Investigate for at-least three different initial conditions for all the cases (i-iv). Compare the results for different cases.

MATLAB Code:

```
clear;
close all;

%initial conditions
global Bx; global By; global Bz;    %magnetic field
Bx = 0; By = 0; Bz = 1;

global Ex; global Ey; global Ez;    %electric field
Ex = 0; Ey = 0; Ez = 0;

global q;                %charge
q = 1.6e-19;

global g;                %gravitational constant
global m;                %mass of particle
m = 9.1e-31;
g=9.8;

%time scale
total_time = 100;
dt = 0.1;

%initial position
x = 0;
y = 0;
z = 0;
```

```
%initial velocity
vx =0;
vy =0;
vz =1;

% set the initial and final times
tstart=0;
tfinal=total_time;

% set the initial conditions in the y0 column vector
u_init = zeros(6,1);
u_init(1) = x;
u_init(2) = y;
u_init(3) = z;
u_init(4) = vx;
u_init(5) = vy;
u_init(6) = vz;

%solving using ode-solver
options=odeset('RelTol',1e-5);
[t,u] = ode45(@q1_ode_lorentz_force, [tstart:dt:tfinal], u_init, options);

% store the solution that comes back into x and v arrays
pos_x = u(:,1);
pos_y = u(:,2);
pos_z = u(:,3);
vel_x = u(:,4);
vel_y = u(:,5);
vel_z = u(:,6);

plot3(pos_x,pos_y,pos_z)
title('motion under constant and uniform magnetic field');
xlable('x')
ylable('y')
zlable('z')
```

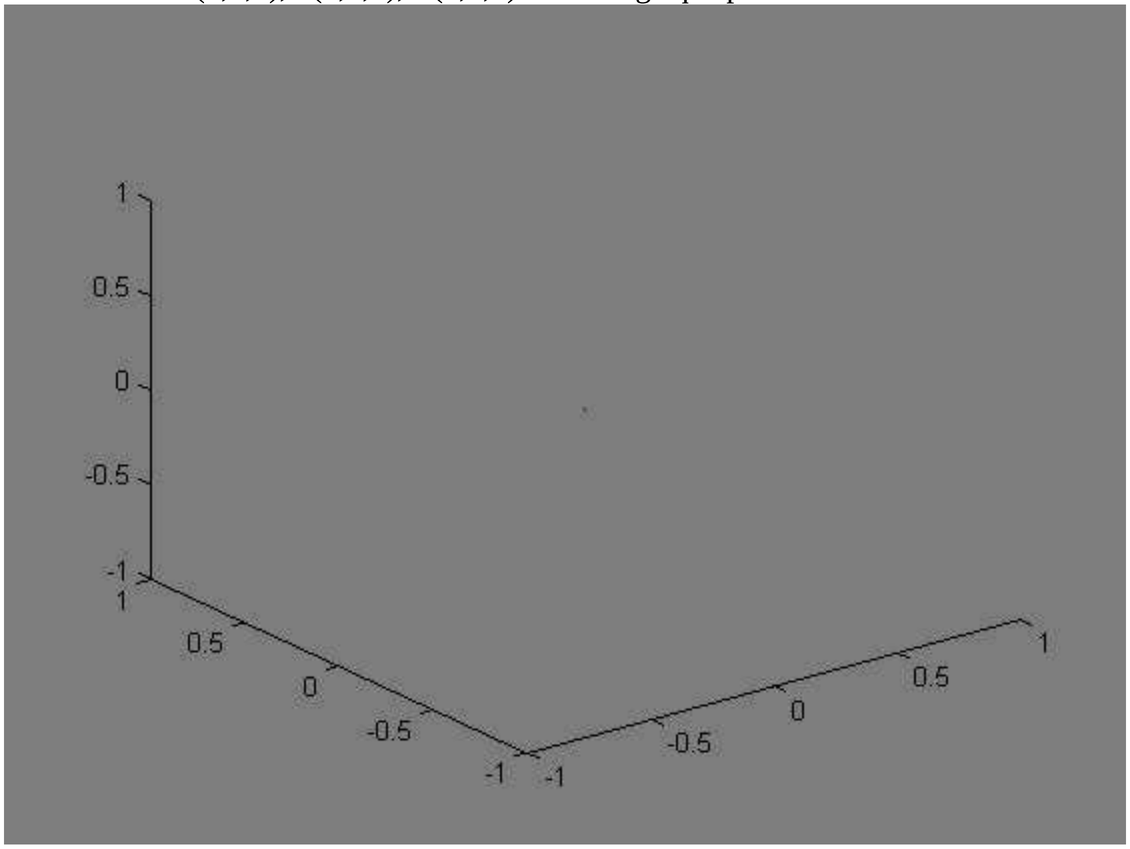
ODE solver Function:

```
function F = q1_ode_lorentz_force(t,u)
% function output =name(input)
% right-hand side function for Matlab's ODE solver,
F=zeros(length(u),1);

% In our case we will use:
% u(1) -> x
% u(2) -> y
% u(3) -> z
% u(4) -> vx
% u(5) -> vy
% u(6) -> vz

% declare the globals so its value
% set in the main script can be used here
global Bx; global By; global Bz;
global Ex; global Ey; global Ez;
global q; global m; global g;

%Bz=0.11*u(1); //uncomment this when we want a position dependent field

F(1) = u(4);
F(2) = u(5);
F(3) = u(6);
F(4) = q/m * (Ex + u(5)*Bz - By*u(6));
F(5) = q/m * (Ey + u(6)*Bx - Bz*u(4));
F(6) = q/m * (Ez + u(4)*By - Bx*u(5));% - g;       % -g is to see the effect
of gravity which is in z-direction
```

Coordinate convention used in the 3d plot- x axis – rightwards horizontally
y axis – into the page
z axis – staright up

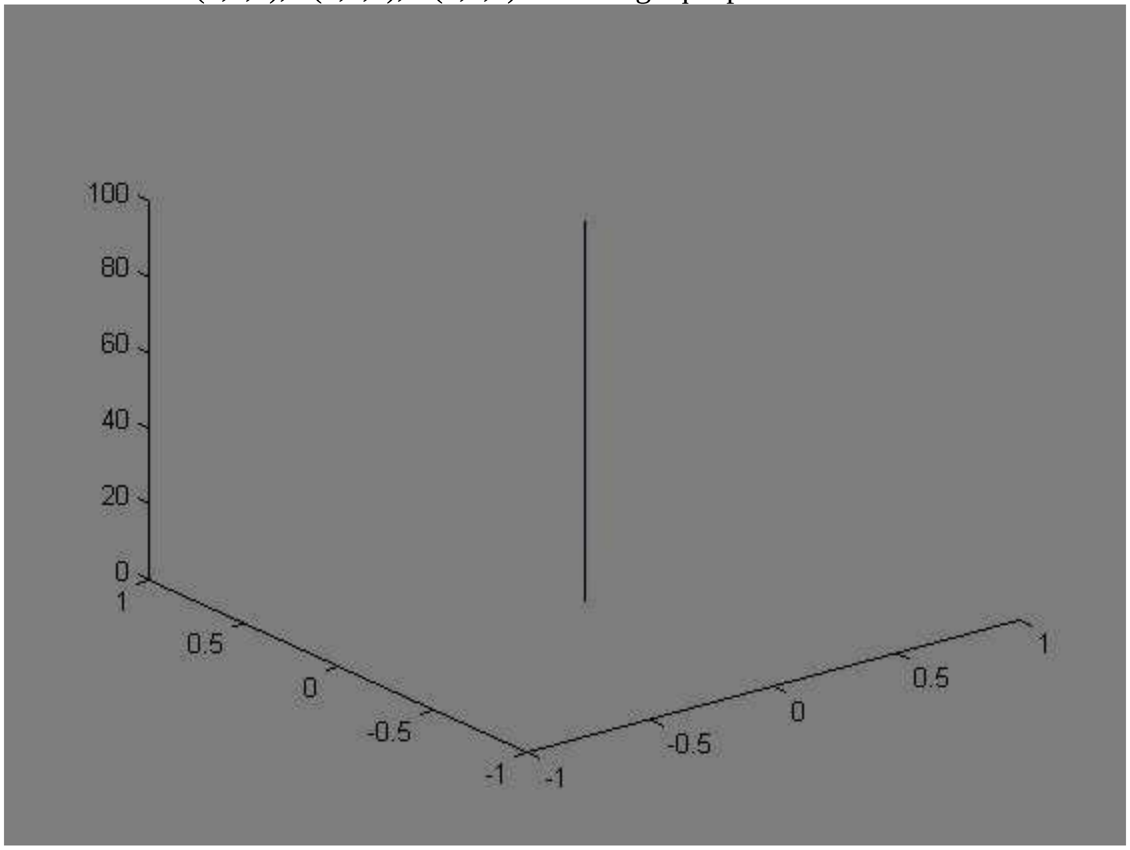
(i) Static and uniform B field. (for +ve and –ve charges)

Condition 1: B(0,0,1), v(0,0,0), E(0,0,0) and charge q is positive.



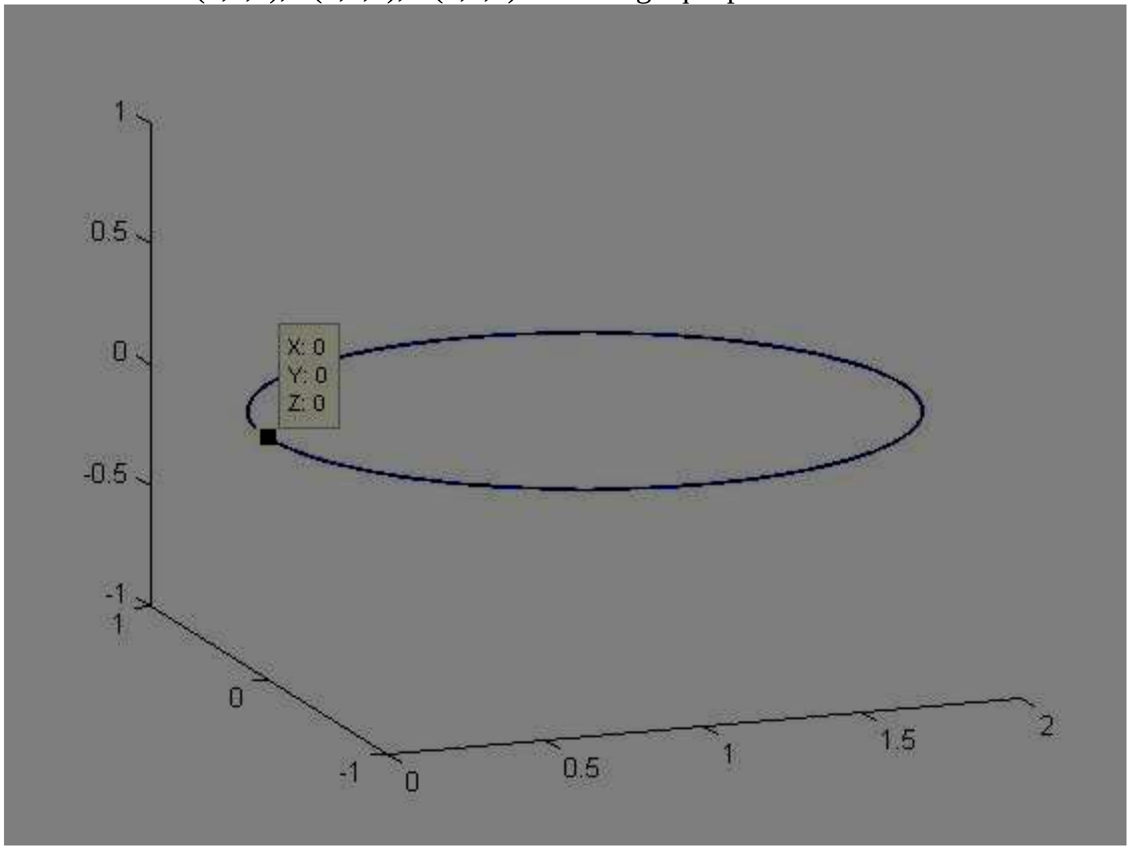
Since v = 0, Lorentz force is zero. Hence there is no motion.

Condition 2: B(0,0,1), v(0,0,1), E(0,0,0) and charge q is positive.



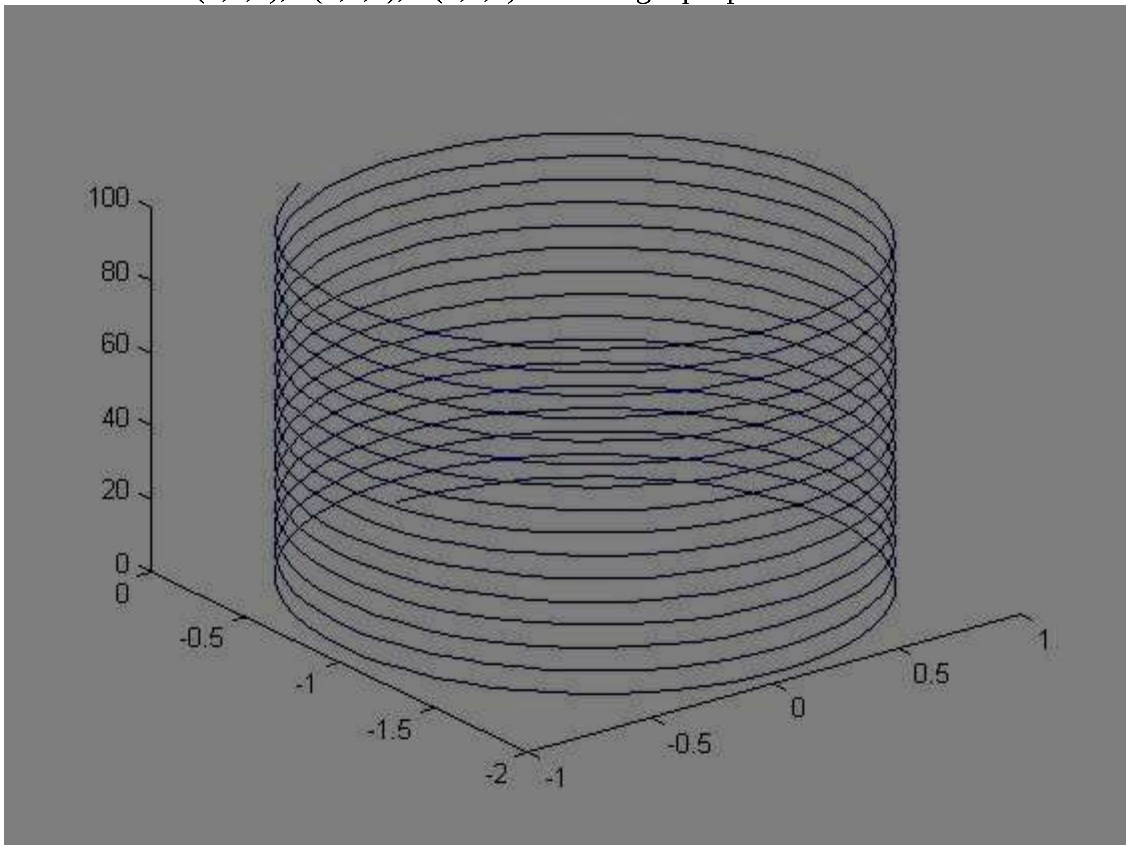
Here velocity and B are parallel to each other, therefore v x B = 0, hence the motion is a straight line.

Condition 3: B(0,0,1), v(1,0,0), E(0,0,0) and charge q is positive.



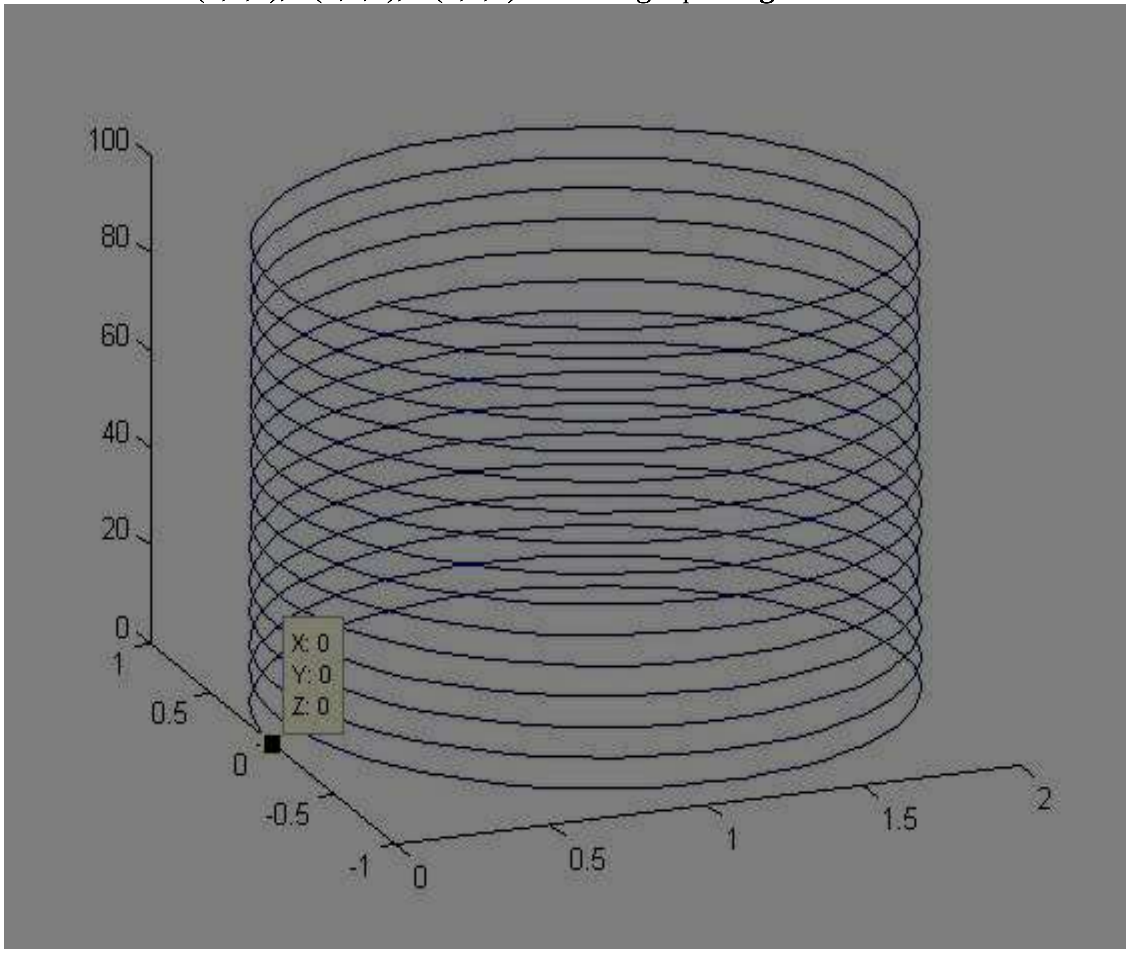
Since the velocity is perpendicular to the direction of magnetic field, the thumb rule gives the direction of motion. In this case it anti-clockwise due to positive charge.

Condition 4: B(0,0,1), v(1,0,1), E(0,0,0) and charge q is positive.



Velocity in z-direction, supports the motion in z-direction. At the same time, the x-component of velocity, perpendicular to the magnetic field, causes circular motion. The net effect of these two cases results in the above helix motion. The direction of motion is anti-clockwise given by the thumb rule.

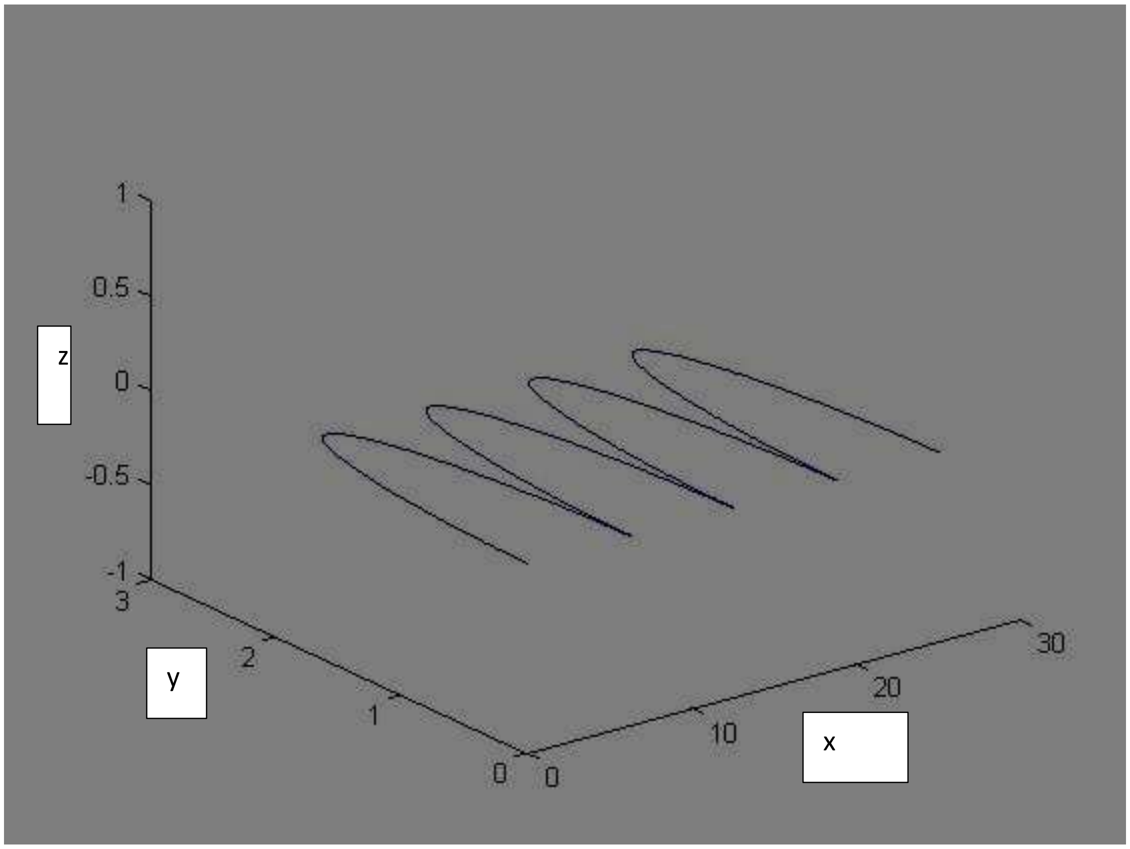
Condition 5: B(0,0,1), v(1,1,1), E(0,0,0) and charge q is **negative**.



Velocity in z-direction, supports the motion in z-direction. Here, due to negative charge, the motion is upwards. At the same time, the x-component of velocity, perpendicular to the magnetic field, causes circular motion. The direction of the circular movement changes to clockwise given by the left-hand curl of fingers when thumb pointing in the direction of magnetic field, since the charge is negative. The net effect of these two cases results in the above helix motion.

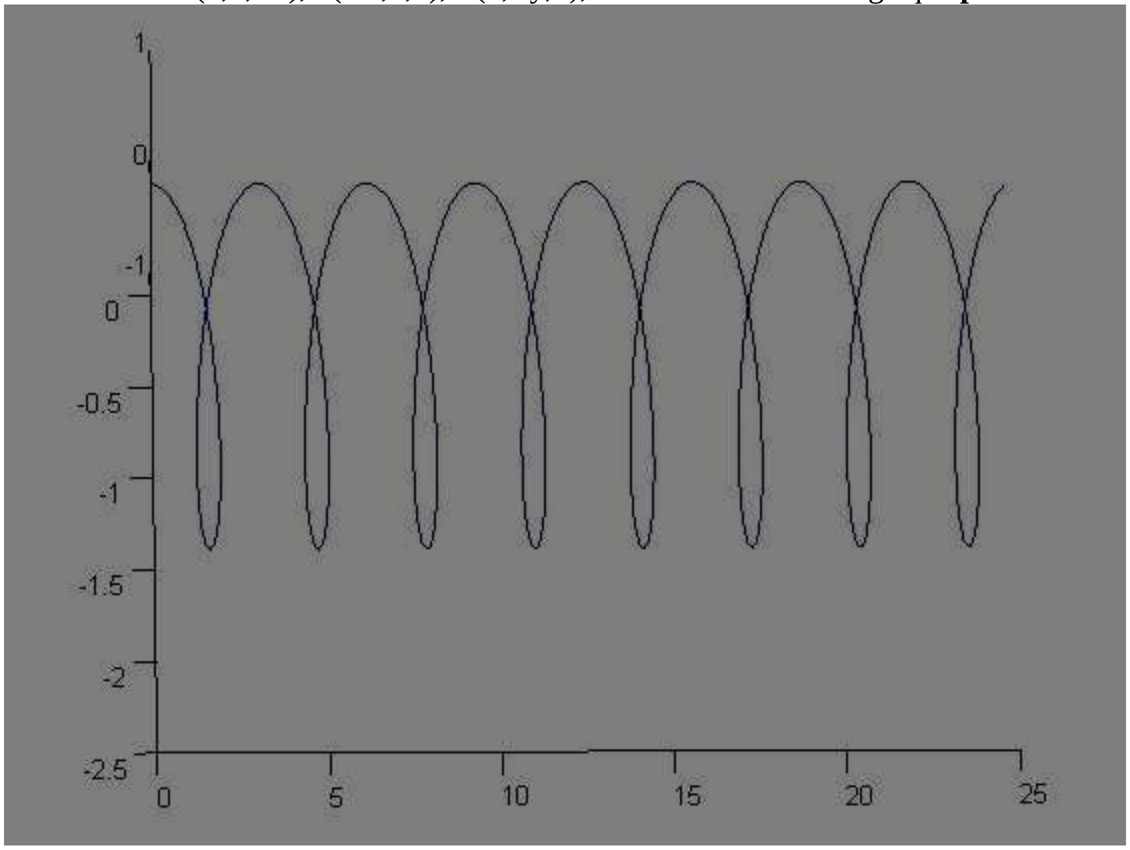
(ii) Static and uniform E and B. (ExB drift) (what happens when v=v0x; B=B0z; and E=E0y; and v0=E0/B0).

Condition 1: B(0,0,1), v(0,0,0), E(0,1,0) and charge q is **positive**.



Initial velocity is 0. Therefore no magnetic force. Particle moves under the influence of electric field in y-direction. As soon as particle gets velocity, magnetic force starts acting and pushes the particle in x-direction. Eventually, the velocity has a y-component. This y-component counter acts the velocity due to electric field and the net y-velocity of particle becomes 0. Then, particle starts moving in –y direction but the electric field still retards the particle in y-direction and it eventually changes back its direction. The drift velocity is constant and a cross product of E and B (y-dir x z-dir) and hence particle continues to move in x-direction.

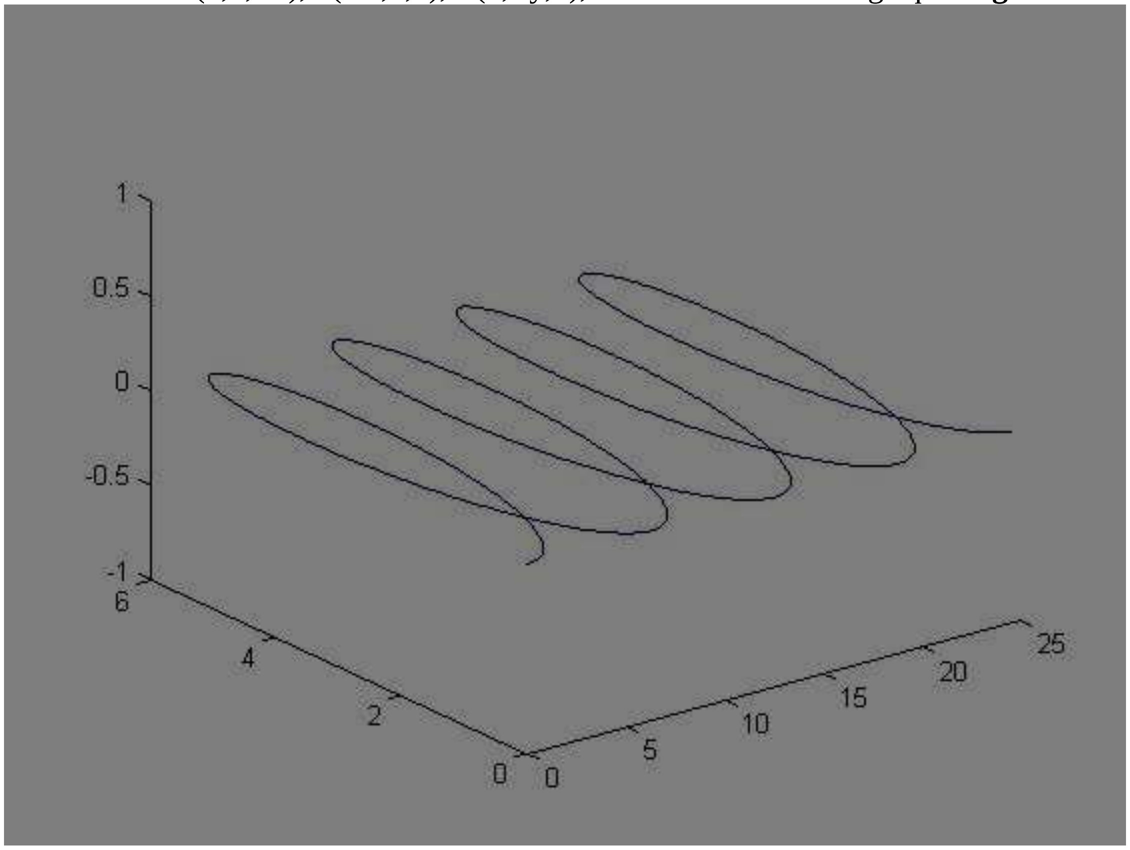
Condition 2: B(0,0,Bz), v(Vx,0,0), E(0,Ey,0), Vx > E0/B0 and charge q is **positive**.



3D graph has been rotated to get a 2D view.

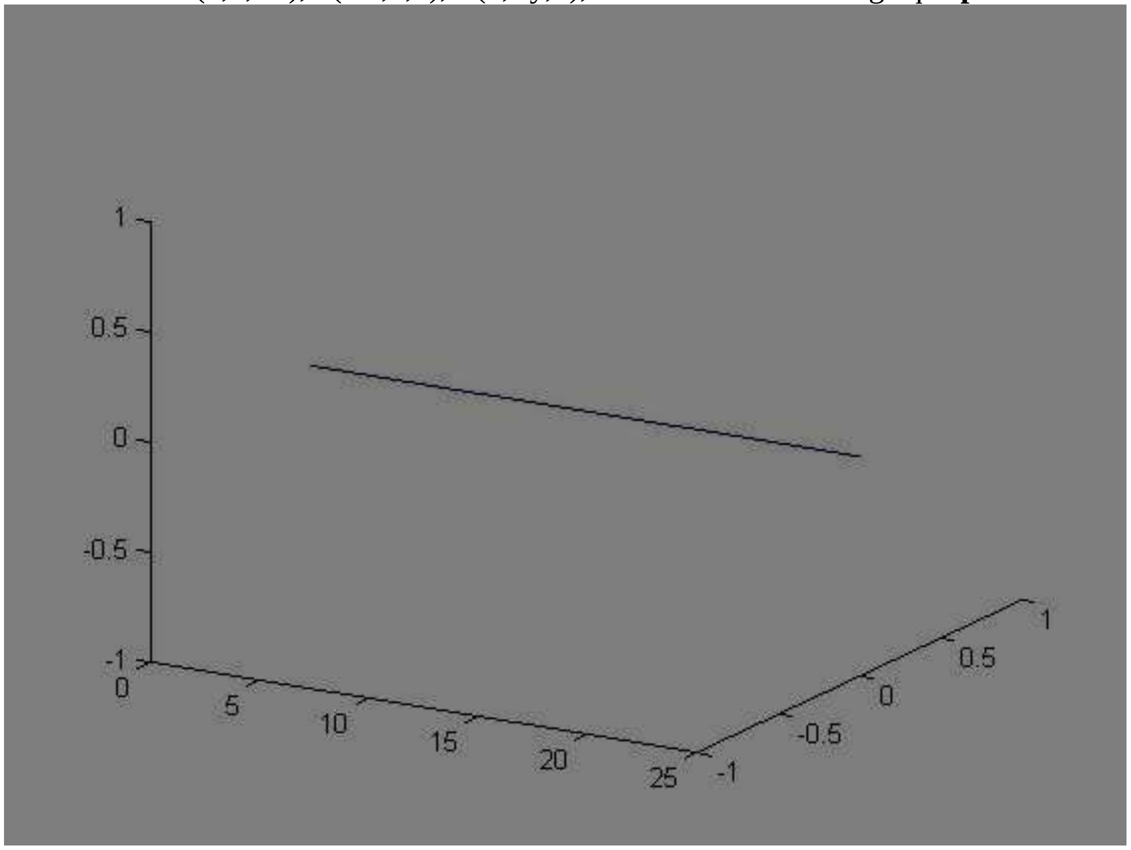
Here velocity is in x-direction perpendicular to direction of magnetic field(z-direction). From previous case, this causes circular motion in the absence of electric field. Now when electric field is present, There is a constant drift velocity in x-direction. Hence when particle moves in x-dir, it supports the motion. Due to magnetic force, particle moves in y-dir as well. It eventually turns back after half circle due to magnetic force. But due to constant drift velocity rather than completing the circle it gives the above motion where particle continues to move ahead in x-dir.

Condition 3: B(0,0,Bz), v(Vx,0,0), E(0,Ey,0), Vx > E0/B0 and charge q is **negative**.



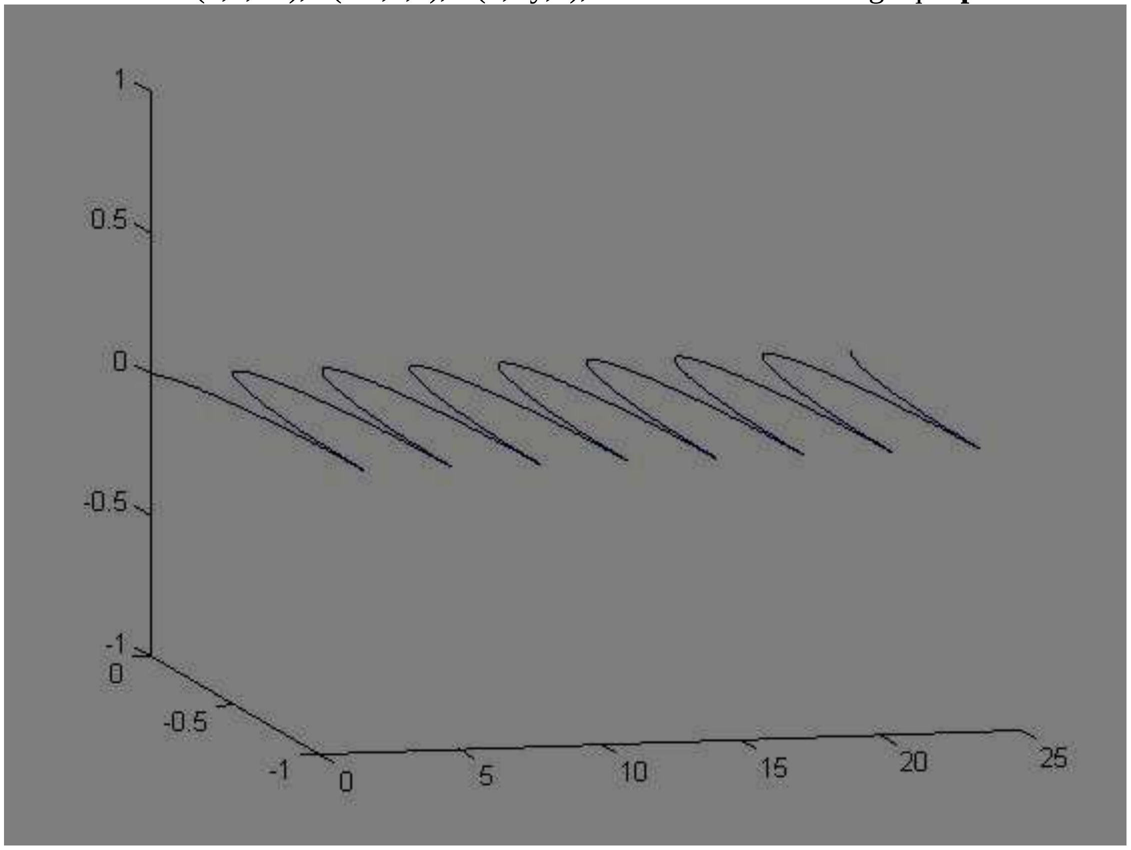
This condition is same as above except for the sign of the charge. In this case, the charge is negative which causes anti-clockwise movement. The direction of motion is also same – x-direction. This shows that the magnetic force depends on the nature of charge but the electric force and the drift velocity is independent of the nature of charge.

Condition 4: B(0,0,Bz), v(Vx,0,0), E(0,Ey,0), Vx = E0/B0 and charge q is **positive**.



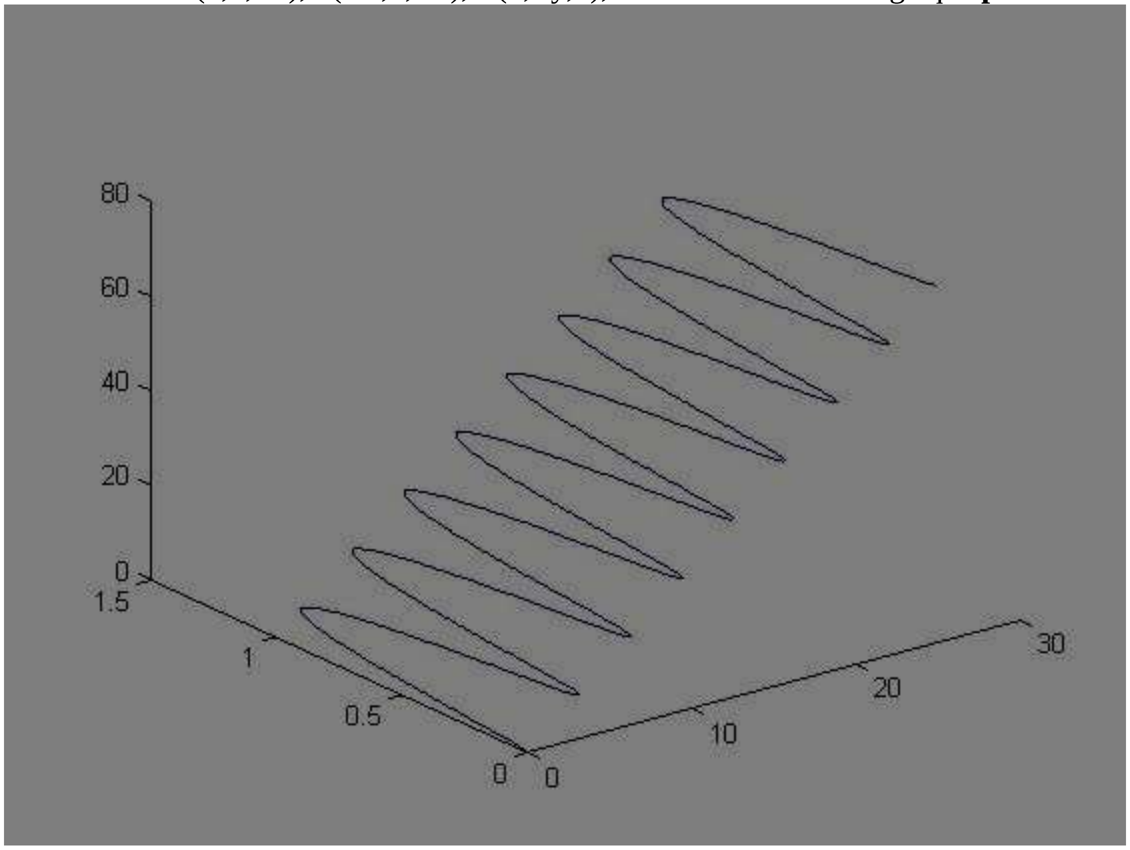
This is a special case of straight line motion. Here the magnetic force and the electric force acting opposite to each other cancel each other due to equal magnitude. Hence no force acts on particle and it continues with constant velocity.

Condition 5: B(0,0,Bz), v(Vx,0,0), E(0,Ey,0), Vx < E0/B0 and charge q is **positive**.



In this condition, we don’t get circular motion or any backward loop by the particle! This is because, magnetic forces are dominated by electric forces and hence magnetic force trying to turn the particle back is overcome by the electric force which makes the particle continuously move in the positive x-dir.

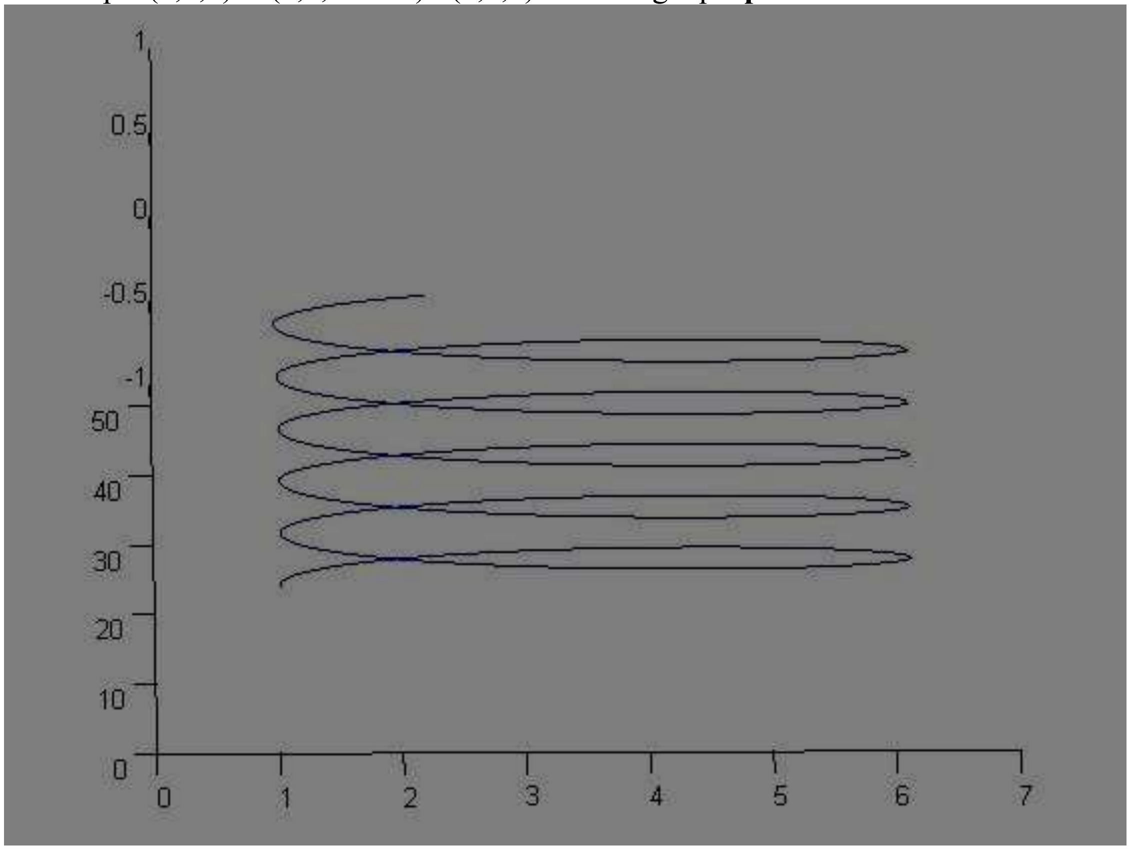
Condition 6: B(0,0,Bz), v(Vx,0,Vz), E(0,Ey,0), Vx < E0/B0 and charge q is **positive**.



Here the velocity has a z-component. Hence the motion is in 3 dimension.

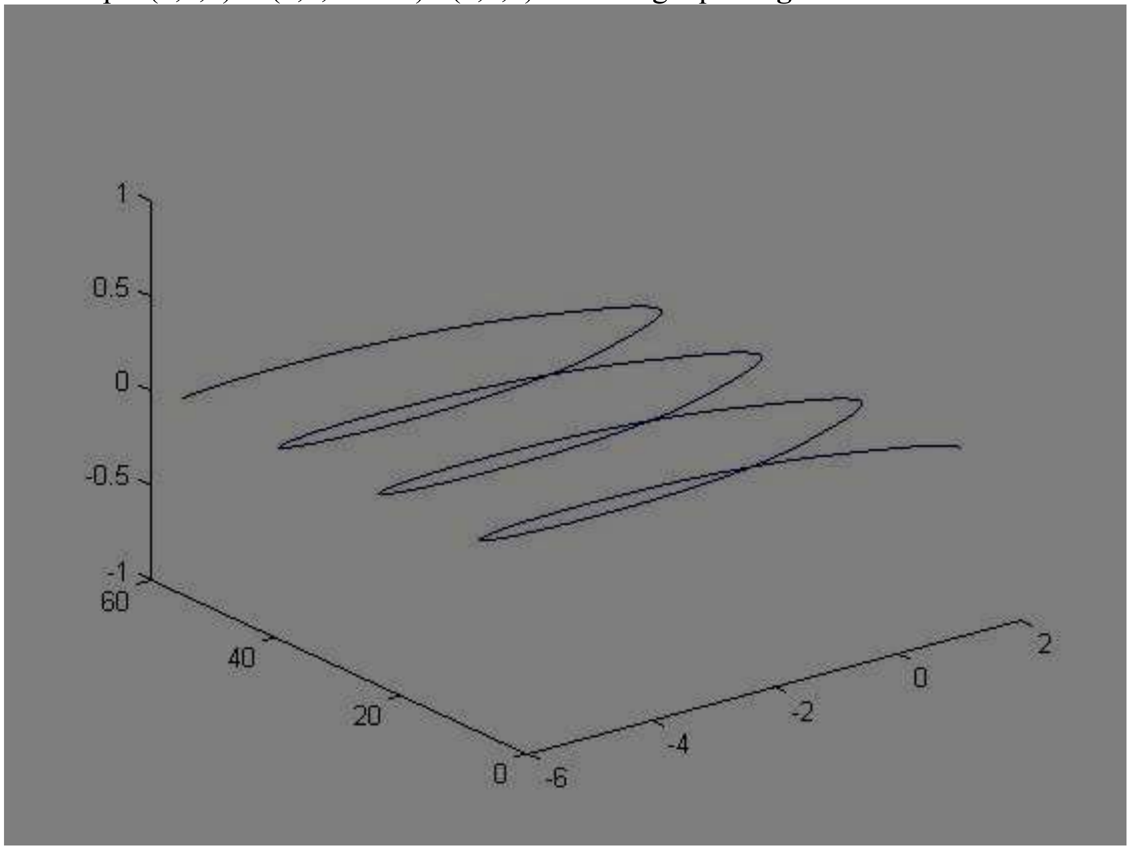
(iii) Static and non-uniform B field (grad B drift)

Condition1: pos(1,0,0) B(0,0,0.11*x) v(0,1,0) and charge q is **positive.**



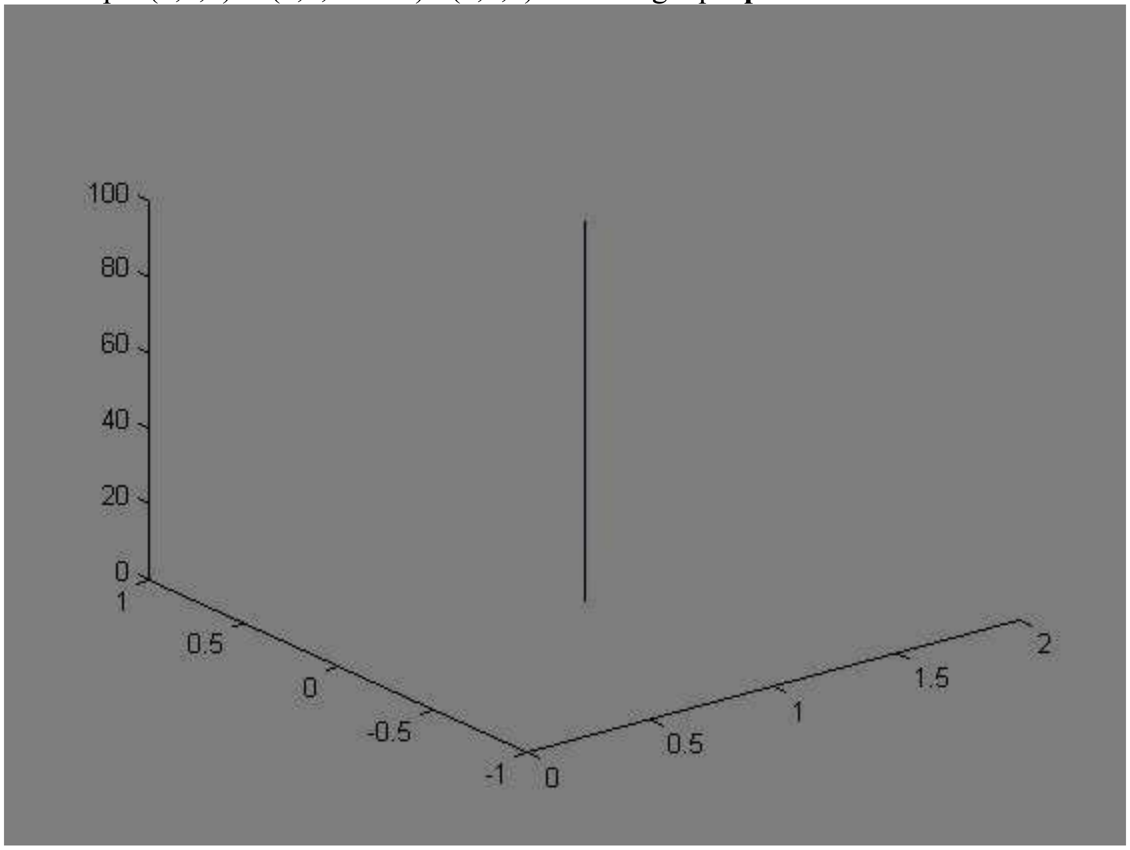
This is a rotated view. Here the drift is due to the non uniformity of the magnetic field. Direction of drift velocity in this case is given by the cross product of the gradient and the magnetic field. Hence we get a drift towards the positive y direciton.

Condition2: pos(1,0,0) B(0,0,0.11*x) v(0,1,0) and charge q is **negative.**



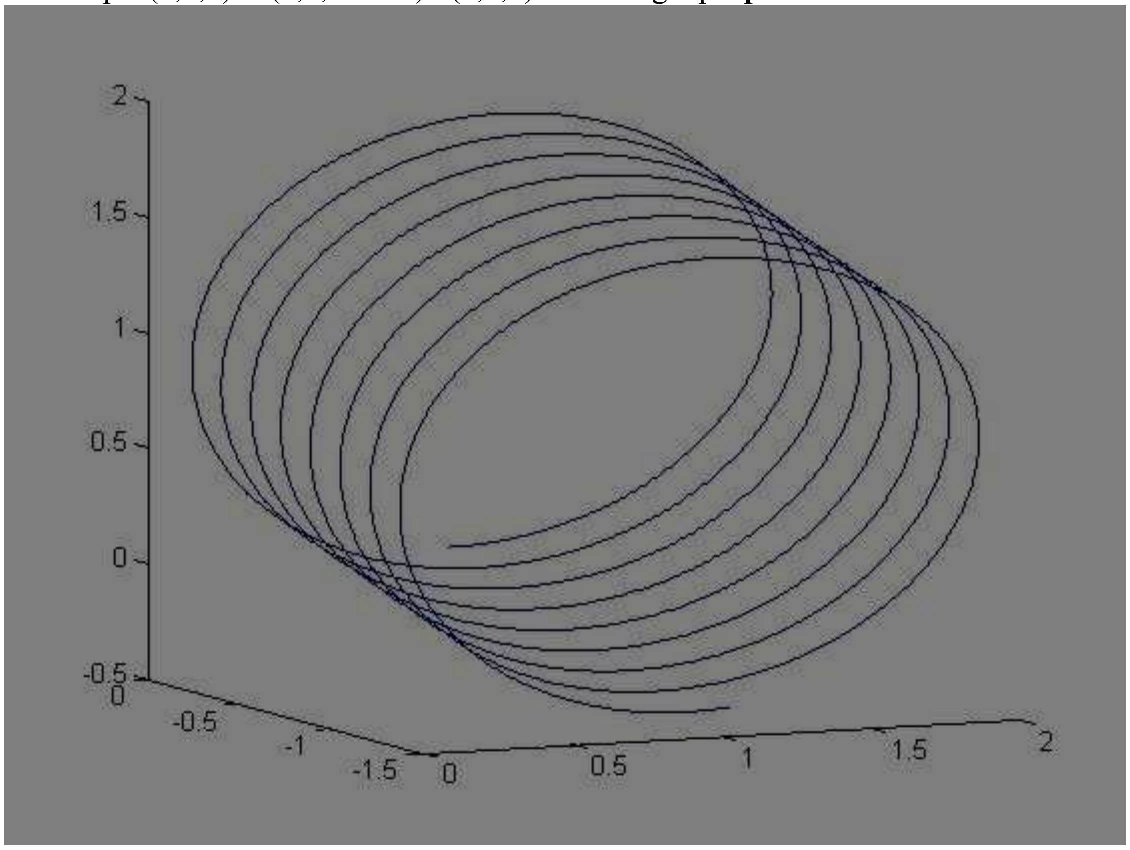
Here also the net direction of motion is same which means direction of rotation depends on the negative charge but the direction of drift is independent of the charge.

Condition3: pos(1,0,0) B(0,0,0.11*x) v(0,0,1) and charge q is **positive.**

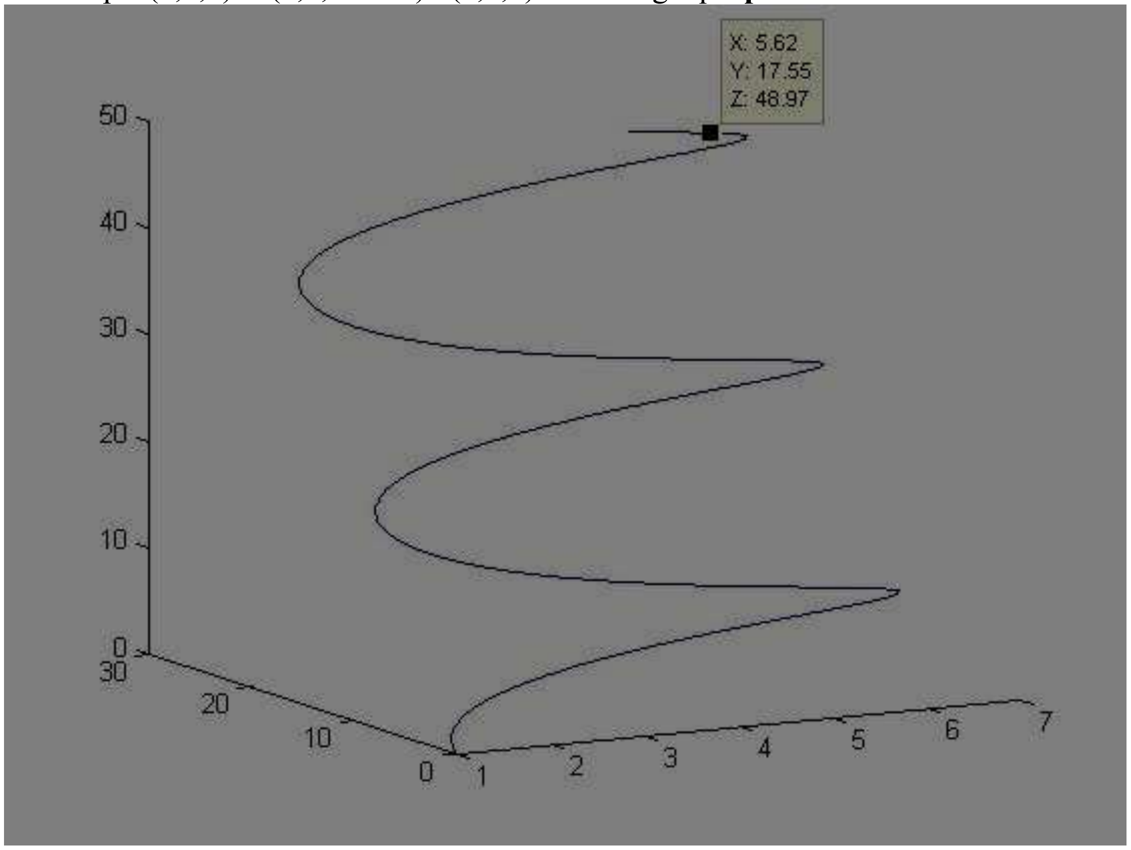


Since velocity is in the direction of the electric field, net force is zero. Hence straight line motion. The same holds for a negatively charged particle.

Condition4: pos(1,0,0) B(0,1,0.11*x) v(1,0,0) and charge q is **positive.**



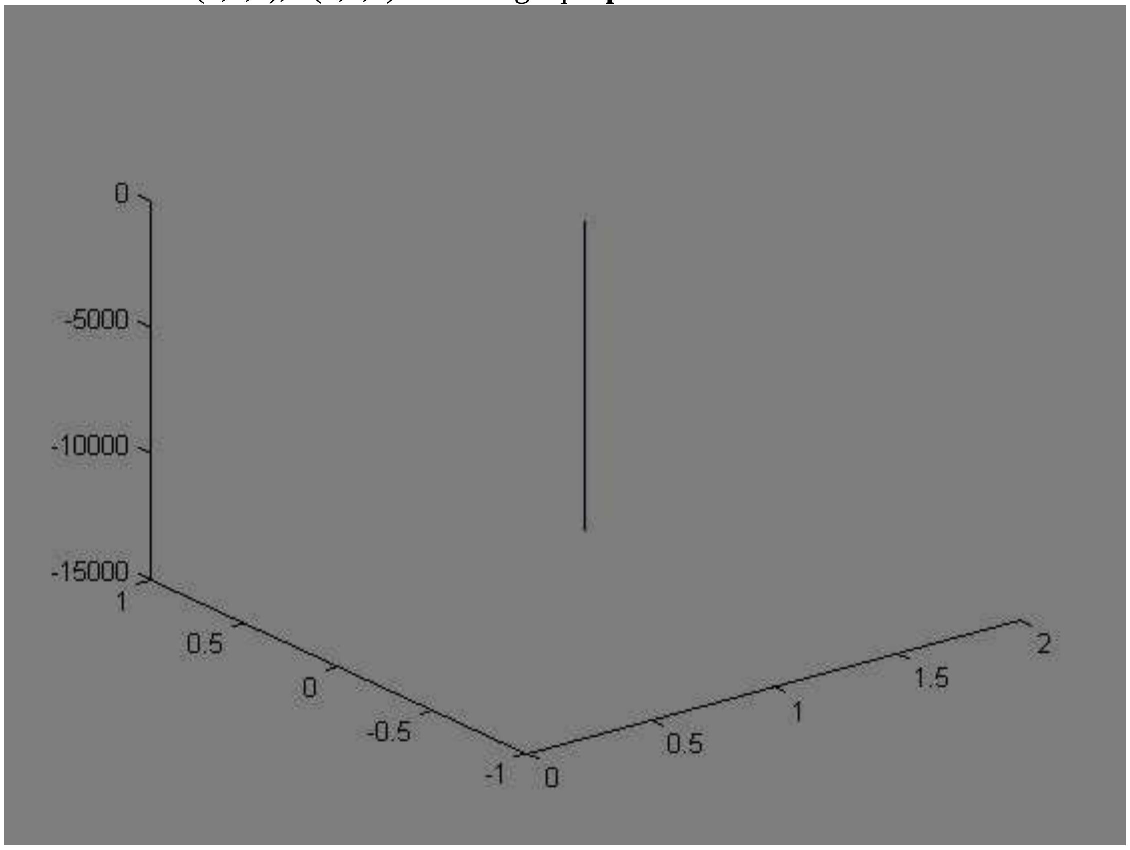
Condition5: pos(1,0,0) B(0,1,0.11*x) v(1,0,1) and charge q is **positive.**



This is essentially same as that the motion in part 4, except that as no force acts in the z direction, velocity in that direction stays constant. Hence apart from performing the motion in part 4, now it will also rise in height along z.

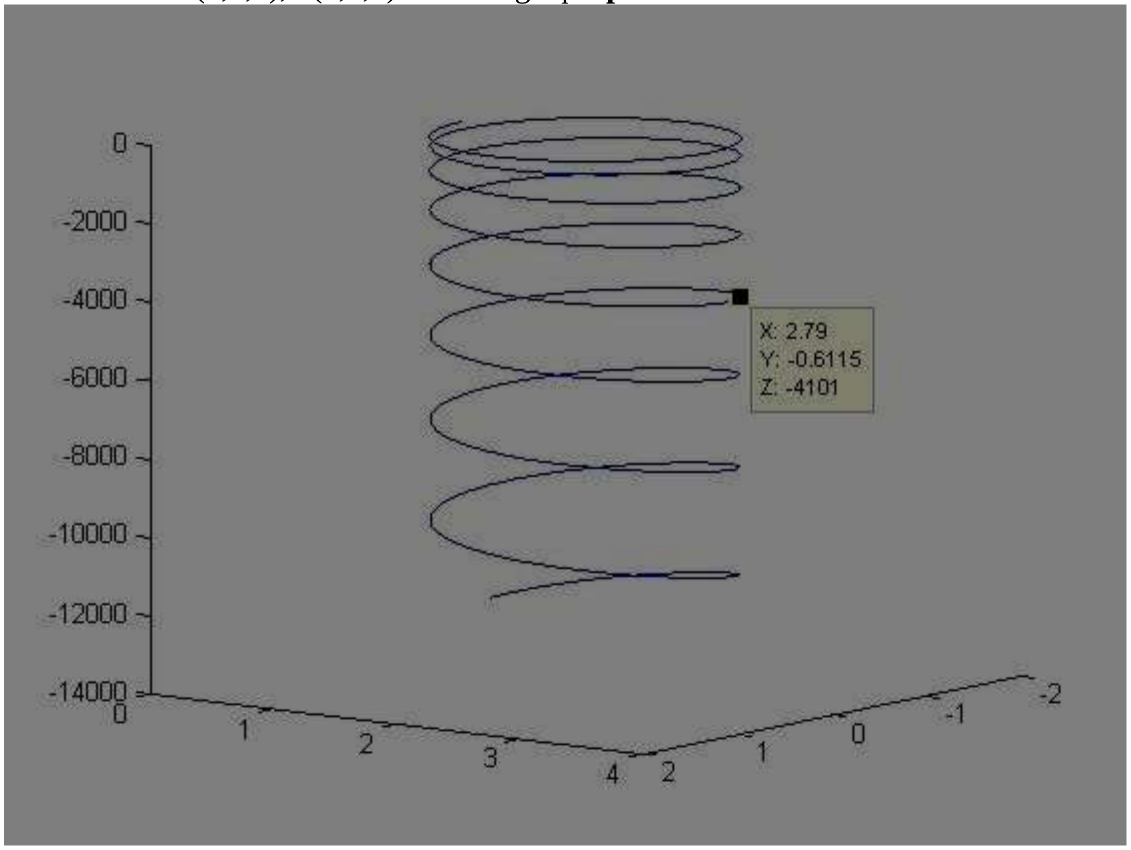
(iv) Static and uniform B, and under gravitational force (for different mass)

Condition 1: B(0,0,1), v(0,0,1) and charge q is **positive**.



Here v || B hence Lorentz force = 0. The only force acting on particle is gravity. Hence straight line motion in downward direction.

Condition 2: B(0,0,1), v(0,1,0) and charge q is **positive**.

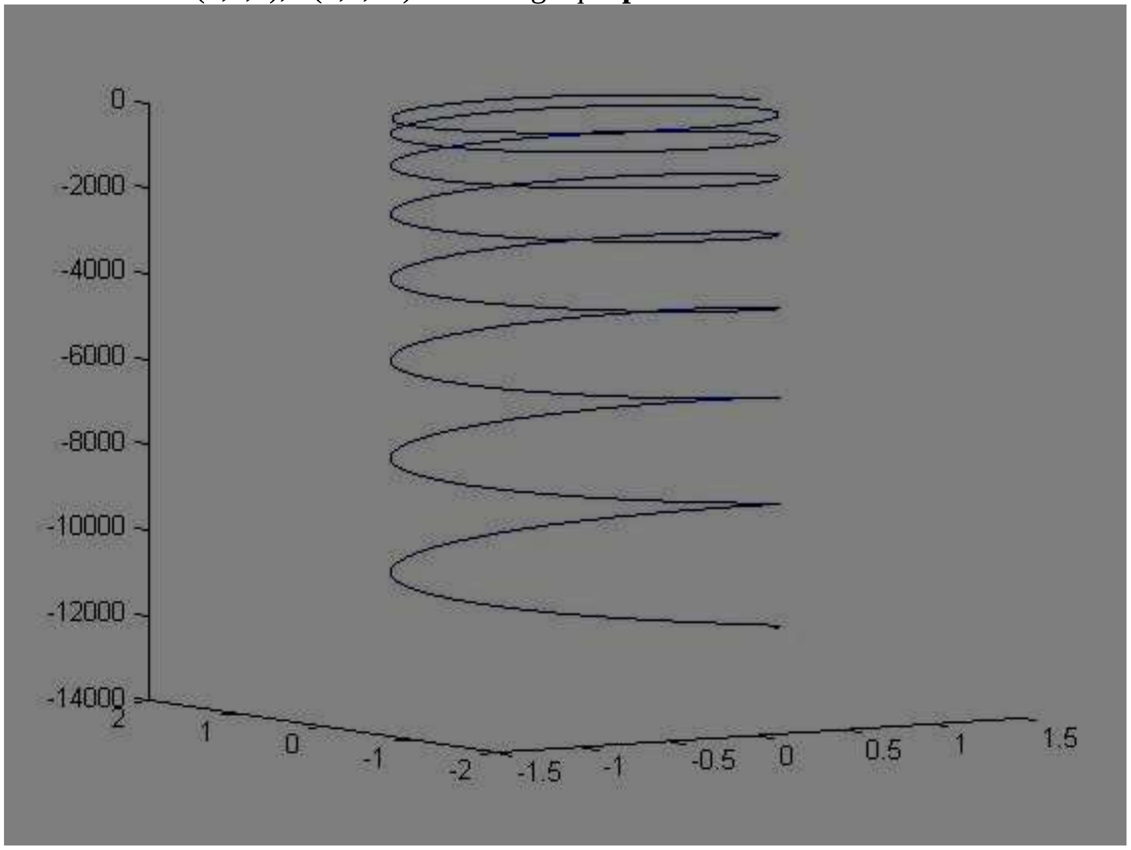


Here velocity and magnetic field are perpendicular which causes circular motion. At the same time, due to constant gravitational force, the velocity in y-direction increases continuously resulting in the above motion.

Condition 3: B(0,0,1), v(1,1,0) and charge q is **positive**.
The motion is same as above condition except for the radius of the circle increases due to increase in magnitude of velocity.

Condition 4: B(0,0,1), v(1,1,0) and charge q is **negative**.
The direction of motion changes due to thumb rule since the charge is negative but the nature of motion remains same. This is because, gravity always acts downwards.

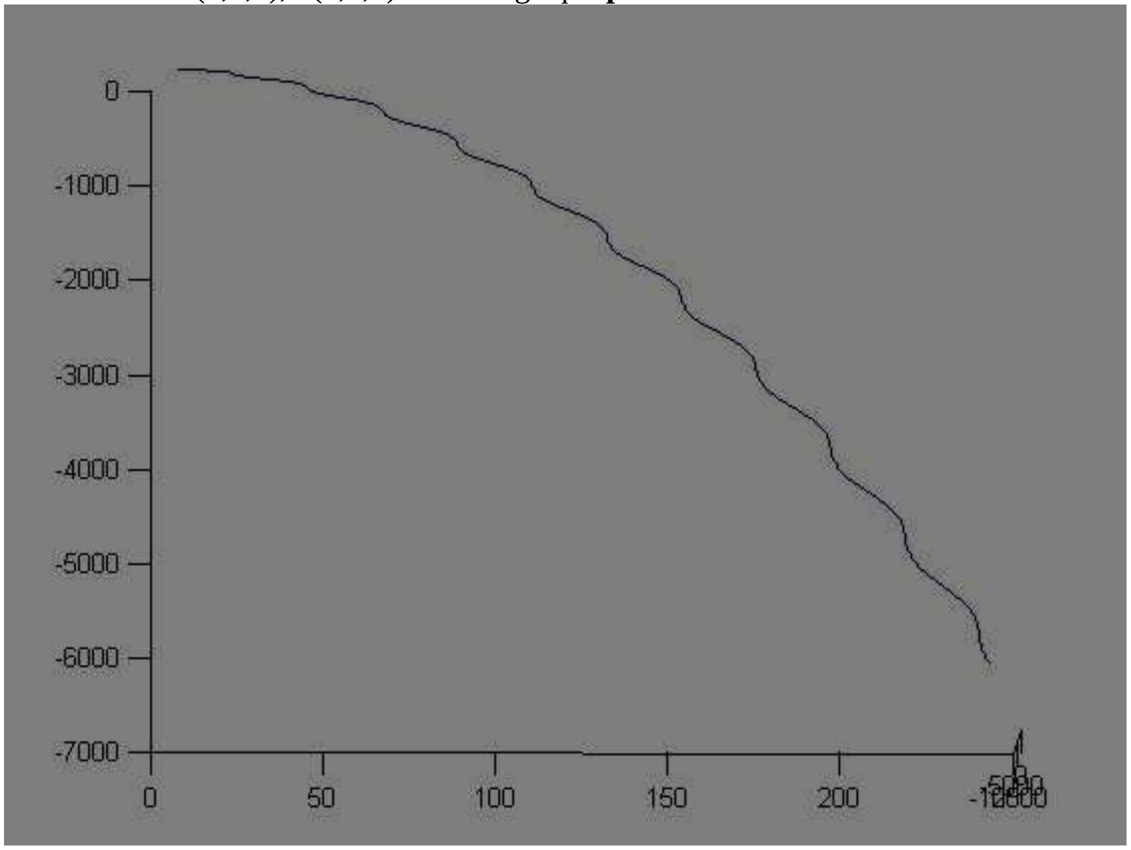
Condition 5: B(0,0,1), v(1,1,10) and charge q is **positive**.



Here, there is initial velocity in +ve z –dir hence particle first moves upwards until due to gravity, this velocity becomes 0 and upward movement stops. After this, the particle spirals downward as in the above three conditions.

Effect of changing mass in the above 5 cases: As the mass increases, the gravity dominates. Hence the downward force increases and particles falls downward extremely fast. The motion is almost a straight line.

Condition 6: B(0,1,1), v(1,1,1) and charge q is **positive**.



Gravity always pulls the particle down. The kinks are due to the magnetic force which has two components. The exact nature of force is difficult to determine theoretically due to the presence of multiple components. Hence computational solution is the best possible solution in such cases.

Q2. Compute with your matlab code, the cyclotron frequency and the cyclotron radius for – an electron in the Earth's ionosphere at 300 km altitude, where the magnetic flux density B~.00005 Tesla, considering that the electron moves at the thermal velocity (kT/m), with T=1000 K, where "k" is Boltzmann's constant. Plot a graph to show the motion/results and compare your results with analytical calculations.

Explanation:

The thermal velocity of the electron is sqrt(k*t/q).
Theoretically the cyclotron radius is given by mv/qB while the angular cyclotron frequency is given by qB/m.
Here we have made an assumption that the particle is not directly above the poles and hence, earth's magnetic lines are parallel to the surface from north to south. We take this to be x-direction. While the gravity acts in z-direction.

There are two ways to find cyclotron radius.

1. Calculate the point at maximum distance from starting point. Get the maximum distance and take half of it.
2. First find the total time period of one iteration. Particle reaches the other end of the diameter in half the time. So measure the distance between initial point and the position of particle at half the time period. This is the diameter. Take half to get the radius.

To find the frequency, we find the time period when the particle returns to its initial position and take the total time taken. This gives us time period. 1/time period = frequency.

MATLAB Code:
```
clear;
close all;

%initial conditions
global Bx; global By; global Bz;
Bx = 5e-5; By = 0; Bz = 0;          %Earth's magnetic field is constant and
is taken in x-direction

global Ex; global Ey; global Ez;
Ex = 0; Ey = 0; Ez = 0;             %Electric field is zero

global q;               %charge of electron
q = -1.6e-19;
global m;               %mass of electron
m = 9.1e-31;
global g
g = 9.8;

k = 1.38e-23;             %Boltzmann's constant
T = 1000;                 %Temperature in kelvin
v_thermal = sqrt(k*T/m);  %Thermal velocity

%time scale
total_time = 1e-6;
```

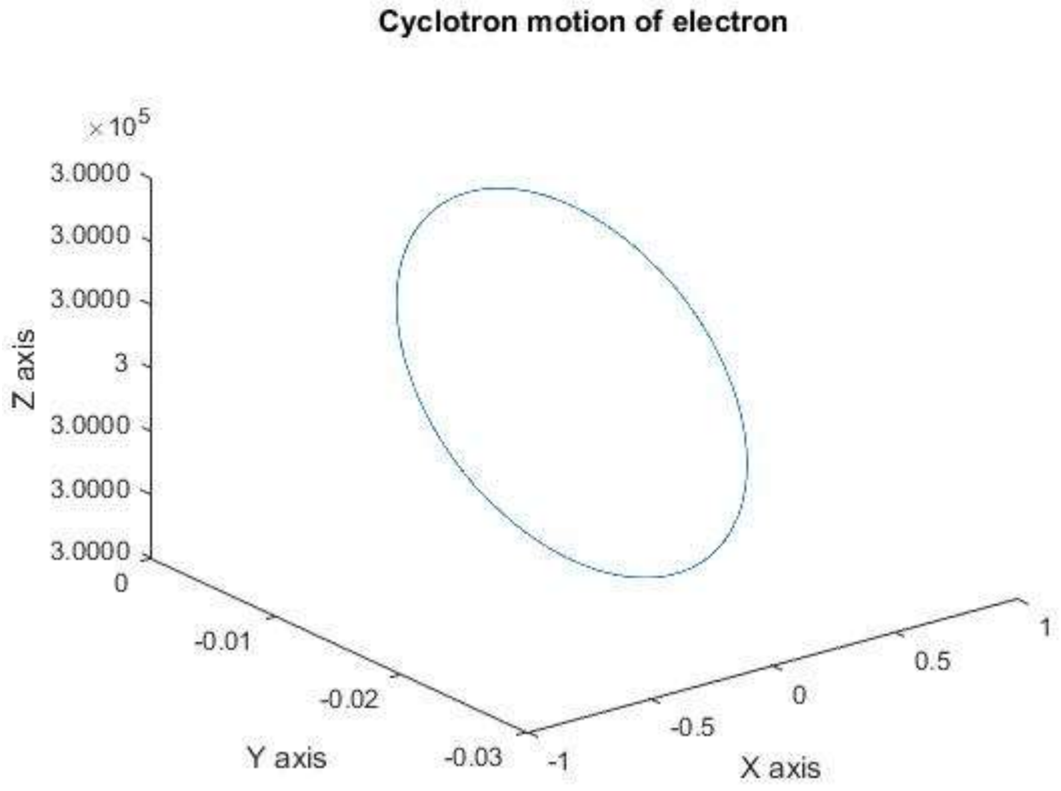
```
dt = 1e-9;

%initial position
x = 0;
y = 0;
z = 300e3;          %in z-direction 300km above surface

%initial velocity
vx = 0;
vy = 0;
vz = v_thermal;

%Analytical solution

%cyclotron frequency
cyclo_freq = abs(q)*Bx/(2*pi*m)

%time period
global cyclo_time;
cyclo_time = 1/cyclo_freq

%cyclotron radius
cyclo_rad = m*v_thermal/(abs(q)*Bx)



%Computational result

% set the initial and final times
tstart = 0;
tfinal = total_time;

% set the initial conditions in the y0 column vector
global x_init; global y_init; global z_init;
z_init = z;  x_init = x; y_init = y;           %storing the initial values

%variables to find maximum distance from which we get radius as well as the
%half time of one complete revolution
global max_dist;
max_dist = -1;
global half_time;
half_time = 0;

u_init = zeros(6,1);
u_init(1) = x;
u_init(2) = y;
u_init(3) = z;
u_init(4) = vx;
u_init(5) = vy;
u_init(6) = vz;

options=odeset('RelTol',1e-5);
[t,u] = ode45(@q2_ode_cyclotron, [tstart:dt:tfinal], u_init, options);
```

```
% store the solution that comes back into x and v arrays
pos_x = u(:,1);
pos_y = u(:,2);
pos_z = u(:,3);
vel_x = u(:,4);
vel_y = u(:,5);
vel_z = u(:,6);


%finding radius and frequency computationally
comp_radius = max_dist / 2
comp_time = half_time * 2
comp_freq = 1 / comp_time

%plotting the graph
plot3(pos_x,pos_y,pos_z)
xlabel('X axis')
ylabel('Y axis')
zlabel('Z axis')
title('Cyclotron motion of electron')
```

FUNCTION:

```
function F = q2_ode_cyclotron(t,u)
% function output =name(input)
% right-hand side function for Matlab's ODE solver,
F=zeros(length(u),1);

% In our case we will use:
% u(1) -> x
% u(2) -> y
% u(3) -> z
% u(4) -> vx
% u(5) -> vy
% u(6) -> vz

% declare the globals so its value
% set in the main script can be used here
global Bx; global By; global Bz;
global Ex; global Ey; global Ez;
global q; global m; global g;
global x_init; global y_init; global z_init;
global max_dist; global half_time;

%Bz=0.11*u(1); //uncomment this when we want a position dependent field
F(1) = u(4);
F(2) = u(5);
F(3) = u(6);
F(4) = q/m * (Ex + u(5)*Bz - By*u(6));
F(5) = q/m * (Ey + u(6)*Bx - Bz*u(4));
F(6) = q/m * (Ez + u(4)*By - Bx*u(5)) - g;

%method of finding frequency and radius using the analytical solution
%if abs(t-cyclo_time/2) <= 2e-8 && flag == 0
%  comp_radius = sqrt( (u(1)-x_init)^2 + (u(2)-y_init)^2+ (u(3)-z_init)^2 )/2
%  comp_time = 2 * t
%  flag = 1;
%end

%method of finding the maximum distance and half time period
%radius = max_dist / 2 and frequency = inverse of 2*half time period
if sqrt( (u(1)-x_init)^2 + (u(2)-y_init)^2 + (u(3)-z_init)^2 ) >= max_dist
   max_dist = sqrt( (u(1)-x_init)^2 + (u(2)-y_init)^2 + (u(3)-z_init)^2 );
   half_time = t;
end
```

OUTPUT:



Command Window:

```
>> Q2_Cyclotron_Motion

%This is analytical solution
cyclo_freq =

   1.3992e+06

cyclo_time =

   7.1471e-07
```

```
cyclo_rad =

    0.0140

%This is computational solution
comp_radius =

    0.0140

comp_time =

   7.1280e-07

comp_freq =

   1.4029e+06

>>
```

From the above output, it can be clearly seen that the analytical solution closely matches with the computational result.

Q3. What will be the gravitational drift velocity “vg” in the above case? Compare your computational result with theoretical result.

Explanation:

Theoratically the gravitational drift velocity is given by (m*g x B)/ (q*B*B). Analytically the value is 1.1e-6 m/s. This is a very small dritft since the mass of electron is very small which reduces the significance of gravitational force. Drift velocity can interpreted as the velocity of the shift of the guiding centre of motion.

Using this definition, the Idea here is that we find the trajectory in which the electron would have been moving had gravity been absent. Here we get one centre. Now, at the next instant (t+dt), due to presence of drift velocity, the particle will move to some other location. For that location by the above process, we will have another instantaneous circle and thus it follows that we will have another centre. Since the drift velocity is constant, it is given by the distance between these centres divided by the dt chosen, because they have been found at a distance dt apart.

Hence to implement this logic, we have used two functions, one to give the ideal trajectory and another to give the actual trajectory as mentioned in the problem.

MATLAB Code:

```
clear;
close all;

%initial conditions
global Bx; global By; global Bz;
Bx =5e-5; By =0 ; Bz =0;

global Ex; global Ey; global Ez;
Ex = 0; Ey =0; Ez =0;

global q;
q = 1.6e-19;




global m;
m =9.1e-31; %we have taken the actual mass of an electron, have NOT made
%the assumption that q/m is 1.

k=1.38e-23;
T=1000;
v_thermal=sqrt(k*T/m);
total_time =2e-6;
global dt; %note dt about 100 times lower than tot_time is reasonable.
global counter;
counter=0;

dt = 0.99e-9;
```

```
x = 0;
y = 0;
z = 300e3; %later can take into consideration the change in g with h.
vx =0;
vy =0;
vz =v_thermal;

global cyclo_time;

cyclo_freq=q*Bx/(2*pi*m);
cyclo_time=1/cyclo_freq;

% set the initial and final times
global tstart;
global tfinal;

tstart=0;
tfinal=total_time;
% set the initial conditions in the y0 column vector
global x_init;global y_init; global z_init;

z_init=z;  x_init=x; y_init=y;
global comp_radius;
comp_radius=0;
global comp_freq;
comp_freq=0;

u_init = zeros(6,1);
u_init(1) = x; % initial position; %theta(1)=.2;
u_init(2) = y; % initial velocity
u_init(3) = z;
u_init(4) = vx;
u_init(5) = vy;
u_init(6) = vz;

options=odeset('RelTol',1e-5); %we have used options for the ode,
                               %check what are its implications
[t,u] = ode45(@q3_ode_drift_vel,[tstart:dt:tfinal],u_init,close);

% store the solution that comes back into x and v arrays
pos_x = u(:,1);
pos_y = u(:,2);
pos_z = u(:,3);
vel_x = u(:,4);
vel_y = u(:,5);
vel_z = u(:,6);

plot3(pos_x,pos_y,pos_z)
```

Actual trajectory function: This function calculates actual trajectory considering the drift velocity.

```matlab
function F = q3_ode_drift_vel(t,u)
% function output =name(input)
% right-hand side function for Matlab's ODE solver,
F=zeros(length(u),1);

% In our case we will use:
% u(1) -> x
% u(2) -> y
% u(3) -> z
% u(4) -> vx
% u(5) -> vy
% u(6) -> vz

% declare the globals so its value
% set in the main script can be used here
global Bx; global By; global Bz;
global Ex; global Ey; global Ez;
global q; global m;
global x_init; global y_init; global z_init;
global cyclo_time;
global comp_radius;
global comp_freq;

global drift_v;
global t1; global t2;

global tstart; global tfinal; global dt;
global counter;
global c1n; global c2n;

xf=0; yf=0; zf=300e3;

%Bz=0.11*u(1); //uncomment this when we want a position dependent field
F(1) = u(4);
F(2) = u(5);
F(3) = u(6);
F(4) = q/m * (Ex + u(5)*Bz - By*u(6));
F(5) = q/m * (Ey + u(6)*Bx - Bz*u(4));
F(6) = q/m * (Ez + u(4)*By - Bx*u(5))-9.8; %weight  considered.

counter=counter+1; %increments in each iteration

if t==0
   %now find the ideal case for point 1
[t1,u1] = ode45(@q3_auxode,[tstart:dt:tfinal],u,close);

 pos_u1_y = u1(:,2);

    for k=1:length(u1)
            %search for time t/2
            %extract the y coodi array
            if t1(k)>=cyclo_time
                    c1n=(pos_u1_y(k)+u(2))/2
                    break;
            end
    end

    %now do the ideal case for init point t+dt
    u_init_new=u;

end

if counter==2
    u_init_new(1)=u(1);
    u_init_new(2)=u(2);
    u_init_new(3)=u(3);
    u_init_new(4)=u(4);
    u_init_new(5)=u(5);
    u_init_new(6)=u(6); %all incer by dt

    [t2,u2] = ode45(@q3_auxode,[tstart:dt:tfinal],u_init_new,close);

    pos_u2_y = u2(:,2);

    for j=1:length(u2)
            %search for time t/2
            if t2(j)>=cyclo_time
                    c2n=(pos_u2_y(j)+ u_init_new(2))/2
                    break;
            end
    end

    drift_v=(c2n-c1n)/dt

end
```

Ideal trajectory function which I call the auxillary ode: This function finds the ideal trajectory when the particle is at a particular point without considering the drift.

```
function F = q3_auxode(t,u)
% function output =name(input)
% right-hand side function for Matlab's ODE solver,
F=zeros(length(u),1);

% In our case we will use:
% u(1) -> x
% u(2) -> y
% u(3) -> z
% u(4) -> vx
% u(5) -> vy
% u(6) -> vz

% declare the globals so its value
% set in the main script can be used here
global Bx; global By; global Bz;
global Ex; global Ey; global Ez;
global q; global m;

%Bz=0.11*u(1); //uncomment this when we want a position dependent field
F(1) = u(4);
F(2) = u(5);
F(3) = u(6);
F(4) = q/m * (Ex + u(5)*Bz - By*u(6));
F(5) = q/m * (Ey + u(6)*Bx - Bz*u(4));
F(6) = q/m * (Ez + u(4)*By - Bx*u(5))-9.8; %weight considered.
```

Analysis:

From this approximate computational method I obtained the gravitational drift velocity as 1.677e-6 which is very close to the theoratical value of 1.1e-6. The error in the answer depends on the precision, the time scale and the number of points that we are using to calculate. If we take some other dt, smaller of greater, the amount of error increases drastically. This is due to both mathematical and round off errors.

Another possible method to find drift velocity:

By thumb rule we expect the particle to move in y-direction depending on our assumption. Hence, we simulate the motion for a sufficiently large time, say 100 iterations. We find the y coordinate of particle at the end of first iteration as well as at the end of the 100^{th} iteration. We subtract then so as to get the total distance travelled by the particle in the y-direction. We subtract it by total simulation time to get velocity in y-direction. Assuming that the initial velocity in y-direction is 0. The velocity that we get is the drift velocity. This is the gravitational drift velocity since is it caused by gravity.

Limitation of this method:

It takes a lot of time to calculate since doing 100 iterations each with at least 100 points is very time consuming. Also in order to get precise answer, we need to reduce dt which increases the time more.

Another limitation is that even after doing so much precision, I was able to get the drift velocity in the range of 10s of m/s. This is way too large than the answer calculated by analysis. Hence there is some precision problem with the code which gives out the distance travelled in y-direction to be much larger than what is expected.