Madhav Krishna

Simulating the Trajectory of Particles in a Cyclotron

Abstract:
This project involves the simulation of a Cyclotron – a type of particle accelerator that uses constant magnetic fields and varying electric fields to accelerate particles (usually relatively heavier particles such as protons and hydrogen nuclei). I have used position and time dependent electric and magnetic fields to simulate the motion of a particle in a cyclotron. I compared the motion of an electron and hydrogen nucleus in a cyclotron with the same electric and magnetic fields to highlight the fact that a different strength and frequency of electric field is required for particles of different q/m (charge to mass) ratios to ensure proper acceleration. I also introduced relativity to simulate the considerable relativistic change in mass of particles that occurs at high velocities. I simulated this relativistic change in mass for an electron in a cyclotron and came to the conclusion that a cyclotron fails as speeds of particles get closer to the speed of light. This is because a cyclotron has an alternating electric field at a constant frequency, but change in mass of particles results in time period of circular motion varying, causing the electric field to go out of phase with the particle.

Introduction and Motivation:

I have been very interested in particle physics for a long time now. I find it extremely unusual yet amazing how particles can interact with seemingly independent forces, electric and magnetic, in a very dependent manner. It's almost like these particles give meaning to the relationship between electric and magnetic forces.

Accelerating particles is essential in the study of particle collisions, since particles interact (collide) when they have extremely high energies and thus at extremely high speeds. Studying particle collisions has helped us better understand how different particles behave in different environments and how they interact with one another. One of the biggest physics discoveries in recent history, the Higgs Boson, was made possible due to particle accelerators and colliders. Over the years, methods of accelerating particles have slowly evolved from using linear accelerators, to more space efficient cyclotrons and even more advanced synchrocyclotrons and synchrotrons.

Background:

The 3 main ideas required to understand the working of a Cyclotron are:

- The Lorentz Force
- Circular Motion
- Electric fields and the acceleration of charged particles in these fields

The Lorentz Force:

The Lorentz force is the force experiences by moving charged particles in a magnetic field. The direction and magnitude of the force experienced by the charged particles is dependent on the orientation the particle's velocity and magnetic field.

The force of the charged particles is given by:

$$F_{Lorentz} = q \cdot \vec{v} \times \vec{B}$$

Where q is the charge of the particle moving in the magnetic field, $\vec{v}$ is the velocity vector of the particle and $\vec{B}$ is the magnetic field vector.

For the purpose of a cyclotron, we do not need to worry about the cross product, since our magnetic field and particle velocity will be mutually perpendicular at all instants.

Circular Motion:
A body undergoes circular motion if it experiences a centripetal force (net force towards the centre). The relationship between the centripetal force, radius of the circular path of the particle, speed of the particle and mass of the particle is given as (from Physics 160):

$$F_{net} = \frac{mv^2}{r}$$

Where m is mass of the particle, v is speed of the particle and r is the radius of the circular path of the particle and F_{net} is the magnitude of centripetal force.

From Newton's 2nd law of motion, we know that:

$$F_{net} = ma$$

and we have equations for angular velocity as follows:

$$\omega = \frac{2\pi}{T}$$

$$\Rightarrow \omega r = \frac{2\pi r}{T}$$

But since the particle moves at a constant speed in uniform circular motion, it completes each oscillation in a fixed amount of time (T), thus we can substitute and obtain:

$$\omega r = v$$

$$\Rightarrow \omega = \frac{v}{r}$$

Where v is the speed of the particle.
Thus we can substitute into the centripetal force equation and obtain:

$$F_{net} = m\omega^2 r$$
$$\Rightarrow a = \omega^2 r$$

Forces on Charged Particles in Electric Fields:

In an Electric field, charged particles experience a force due to the Electric fields given by (from class):

$$\vec{F_{Ef}} = q \cdot \vec{E}$$

Where, q is the charge of the particle in the electric field and $\vec{E}$ is the electric field vector.
In the case of a cyclotron, we will be dealing with electric fields pointing in the same direction as the motion of the particle, thus force due to the electric field will also be in the same direction as the motion of the particle.

Also, in a region of constant electric field, we have:

$$V = Ed$$

Where V is the potential difference between 2 points separated by distance d.

We can thus bring all the concepts together to analyze the working of a cyclotron.

Consider the motion of a charged particle (we will use a positively charged particle for the sake of argument) in a magnetic field.

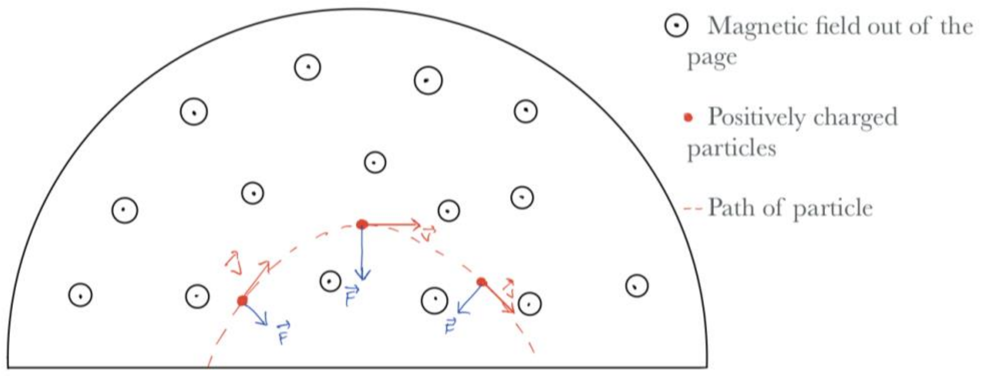

⊙ Magnetic field out of the page
● Positively charged particles
-- Path of particle
$\vec{v}$
$\vec{F}$

Figure 1 - Motion of a positively charged particle in a perpendicular magnetic field

In the above diagram, since the positively charged particle's motion is the plane perpendicular to the magnetic field, we see (by the right hand rule), that the force on the particle is perpendicular to its direction of velocity, and thus always directed towards the center of the circular path it follows (from class).
Also, since the direction of force has no component along the direction of velocity, speed of the particle does not change and thus magnitude of force on the particle does not change.
Thus, we can find a relationship for the radius of the circular path of the particle by substituting the Lorentz force equation for the centripetal force:

$$r = \frac{mv^2}{qvB}$$

Thus, after cancelling we get:

$$r = \frac{mv}{qB}$$

Thus, it is clear the radius is directly proportional to the speed of the particle. This will become relevant when we get to the working of a cyclotron.

We also see, by moving variables around in the above equation:

$$\frac{v}{r} = \frac{qB}{m}$$
$$\Rightarrow \omega = \frac{qB}{m}$$

Thus, it is clear that the angular velocity of the particle doesn't in fact depend on the speed of the particle. It only depends on the q/m ratio and magnitude of the magnetic field, B.

Working of a Cyclotron:

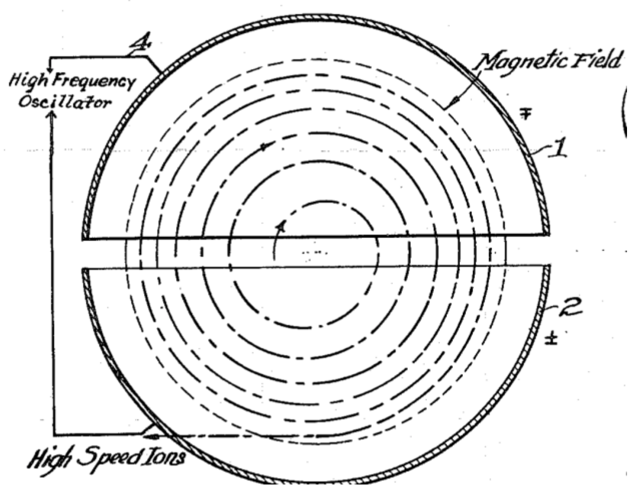
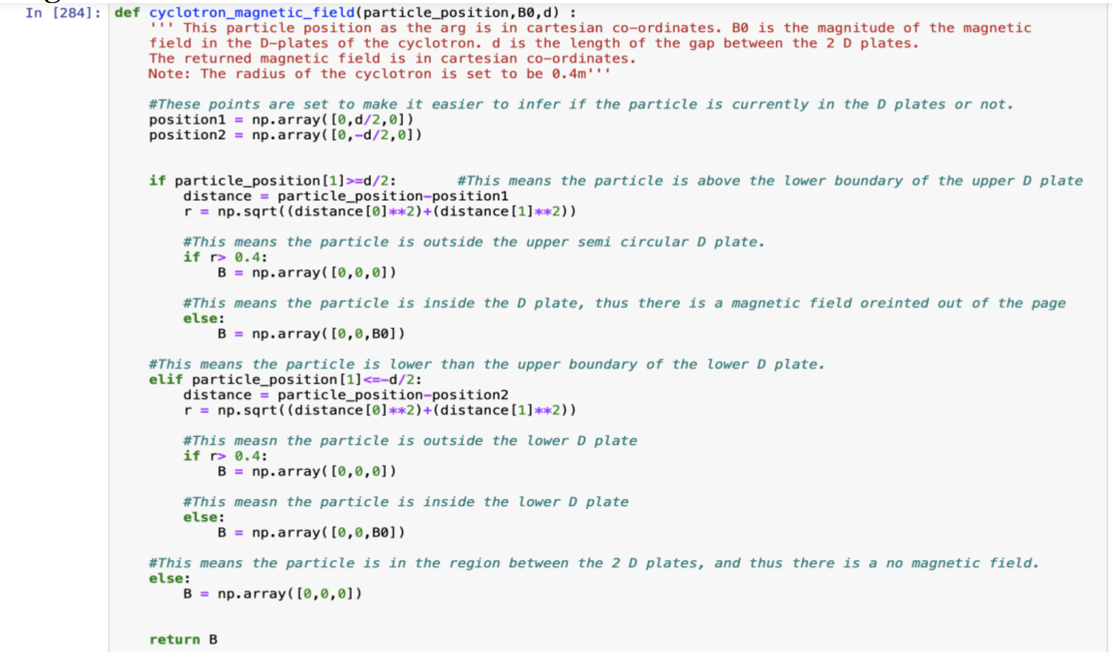


Figure 2 – Cyclotron (Source: U.S. Patent)

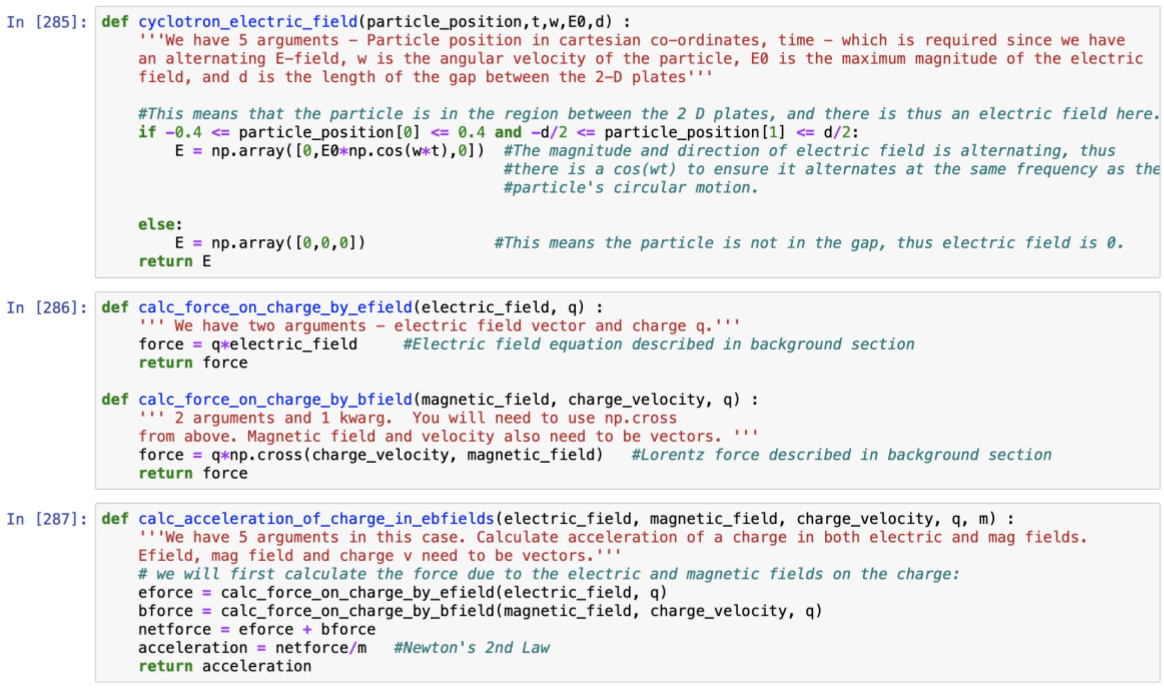
As clear from the above diagram, a cyclotron is made up of 2 D shaped plates, which have a constant magnetic field directed out of the page (like in figure 1). The thin region in between the D-plates is a region where there is an alternating electric field set at a frequency depending on the particle being accelerated. The frequency of the electric field is set such that the direction of the electric field points in the same direction as the velocity of the particle as soon as it emerges out of a D-plate. This is done so that the particle can accelerate and increase its speed due to the electric force (as given in electric field equation described above). Note that, as we showed, the angular velocity of the particle does not depend on its speed, thus throughout the accelerating process, the angular velocity of the particle's circular motion remains constant (assuming no relativistic effects). Thus, once the motion of the particle is in phase with the alternating electric field, it remains so for as long as the particle is within the cyclotron.
This working will become clearer when the code used for the simulation is described.

Setup:

Magnetic Field and Electric Field Functions:



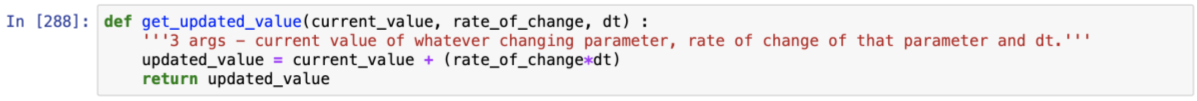
```
In [284]: def cyclotron_magnetic_field(particle_position,B0,d) :
              ''' This particle position as the arg is in cartesian co-ordinates. B0 is the magnitude of the magnetic
              field in the D-plates of the cyclotron. d is the length of the gap between the 2 D plates.
              The returned magnetic field is in cartesian co-ordinates.
              Note: The radius of the cyclotron is set to be 0.4m'''

              #These points are set to make it easier to infer if the particle is currently in the D plates or not.
              position1 = np.array([0,d/2,0])
              position2 = np.array([0,-d/2,0])


              if particle_position[1]>=d/2:        #This means the particle is above the lower boundary of the upper D plate
                  distance = particle_position-position1
                  r = np.sqrt((distance[0]**2)+(distance[1]**2))

                  #This means the particle is outside the upper semi circular D plate.
                  if r> 0.4:
                      B = np.array([0,0,0])

                  #This means the particle is inside the D plate, thus there is a magnetic field oreinted out of the page
                  else:
                      B = np.array([0,0,B0])

              #This means the particle is lower than the upper boundary of the lower D plate.
              elif particle_position[1]<=-d/2:
                  distance = particle_position-position2
                  r = np.sqrt((distance[0]**2)+(distance[1]**2))

                  #This measn the particle is outside the lower D plate
                  if r> 0.4:
                      B = np.array([0,0,0])

                  #This measn the particle is inside the lower D plate
                  else:
                      B = np.array([0,0,B0])

              #This means the particle is in the region between the 2 D plates, and thus there is a no magnetic field.
              else:
                  B = np.array([0,0,0])


              return B
```



```
In [285]: def cyclotron_electric_field(particle_position,t,w,E0,d) :
              '''We have 5 arguments - Particle position in cartesian co-ordinates, time - which is required since we have
              an alternating E-field, w is the angular velocity of the particle, E0 is the maximum magnitude of the electric
              field, and d is the length of the gap between the 2-D plates'''

              #This means that the particle is in the region between the 2 D plates, and there is thus an electric field here.
              if -0.4 <= particle_position[0] <= 0.4 and -d/2 <= particle_position[1] <= d/2:
                  E = np.array([0,E0*np.cos(w*t),0])  #The magnitude and direction of electric field is alternating, thus
                                                      #there is a cos(wt) to ensure it alternates at the same frequency as the
                                                      #particle's circular motion.

              else:
                  E = np.array([0,0,0])              #This means the particle is not in the gap, thus electric field is 0.
              return E

In [286]: def calc_force_on_charge_by_efield(electric_field, q) :
              ''' We have two arguments - electric field vector and charge q.'''
              force = q*electric_field     #Electric field equation described in background section
              return force

          def calc_force_on_charge_by_bfield(magnetic_field, charge_velocity, q) :
              ''' 2 arguments and 1 kwarg.  You will need to use np.cross
              from above. Magnetic field and velocity also need to be vectors. '''
              force = q*np.cross(charge_velocity, magnetic_field)   #Lorentz force described in background section
              return force

In [287]: def calc_acceleration_of_charge_in_ebfields(electric_field, magnetic_field, charge_velocity, q, m) :
              '''We have 5 arguments in this case. Calculate acceleration of a charge in both electric and mag fields.
              Efield, mag field and charge v need to be vectors.'''
              # we will first calculate the force due to the electric and magnetic fields on the charge:
              eforce = calc_force_on_charge_by_efield(electric_field, q)
              bforce = calc_force_on_charge_by_bfield(magnetic_field, charge_velocity, q)
              netforce = eforce + bforce
              acceleration = netforce/m   #Newton's 2nd Law
              return acceleration
```

These functions are used to define the electric and magnetic fields at the position of the inputted particle. There are some other arguments whose reasons for inclusion have been described in the docstrings.
After defining the electric and magnetic fields, we calculate force on the particle at its particular position using the Lorentz force equation and Electric field equation described in the background section.
We then create a function to calculate acceleration on particles depending on the forces on the particle at that position and instant in time.

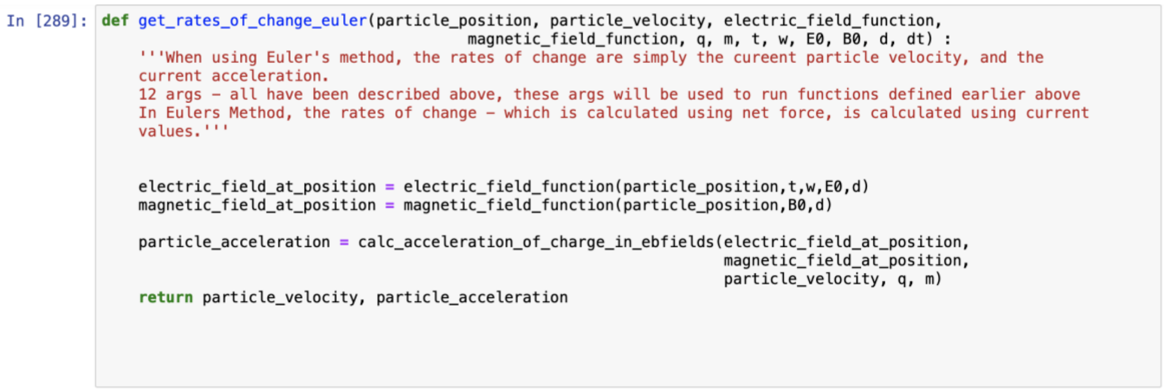
Updated Values Function:



```
In [288]: def get_updated_value(current_value, rate_of_change, dt) :
              '''3 args - current value of whatever changing parameter, rate of change of that parameter and dt.'''
              updated_value = current_value + (rate_of_change*dt)
              return updated_value
```

This updated values function is to calculate new values, of position and velocity in our case, using the rate of change for these variables – velocity and acceleration respectively.

Rates of Change Functions:



```
In [289]: def get_rates_of_change_euler(particle_position, particle_velocity, electric_field_function,
                                        magnetic_field_function, q, m, t, w, E0, B0, d, dt) :
              '''When using Euler's method, the rates of change are simply the cureent particle velocity, and the
              current acceleration.
              12 args - all have been described above, these args will be used to run functions defined earlier above
              In Eulers Method, the rates of change - which is calculated using net force, is calculated using current
              values.'''


              electric_field_at_position = electric_field_function(particle_position,t,w,E0,d)
              magnetic_field_at_position = magnetic_field_function(particle_position,B0,d)

              particle_acceleration = calc_acceleration_of_charge_in_ebfields(electric_field_at_position,
                                                                              magnetic_field_at_position,
                                                                              particle_velocity, q, m)

              return particle_velocity, particle_acceleration
```

```
In [290]: def get_rates_of_change_runge_kutta(particle_position, particle_velocity, electric_field_function,
                                              magnetic_field_function, q, m, t, w, E0, B0, d, dt) :
              ''' This uses the Runge-Kutta method to calculate the updated value.  This method relies on mid-points.
              Note, this method can replace get_rates_of_change_euler in the argument, get_rates_of_change_function, that
              is input to calc_trajectory_in_fields function defined below. Args are the same as in the Euler Method'''

              current_velocity, current_acceleration = get_rates_of_change_euler(particle_position, particle_velocity, electri
                                              magnetic_field_function, q, m, t, w, E0, B0, d, dt)

              #Mid particle velocity and position is used to calculate rates of change, to be a little more precise
              mid_particle_velocity = current_velocity + current_acceleration * dt/2
              mid_particle_position = particle_position + mid_particle_velocity * dt/2

              # Calculate the acceleration due to the electric and magnetic fields at the mid_particle_position
              runge_kutta_velocity, runge_kutta_acceleration = get_rates_of_change_euler(mid_particle_position,
                                                                                          mid_particle_velocity,
                                                                                          electric_field_function,
                                                                                          magnetic_field_function,
                                                                                          q, m, t, w, E0, B0, d, dt)


              return runge_kutta_velocity, runge_kutta_acceleration
```

These are 2 methods to calculate rate of change using current values and calculating net acceleration (rate of change of velocity) using the functions we have defined earlier.
The runge-kutta method is more precise the Euler method, since it uses midpoints and thus results in more precise rates of change calculated for a given dt. However, even though in our simulation we will use the more precise runge-kutta method, we require the euler method to calculate the midpoint values that are used for the runge-kutta method.

Placeholder Array Function:

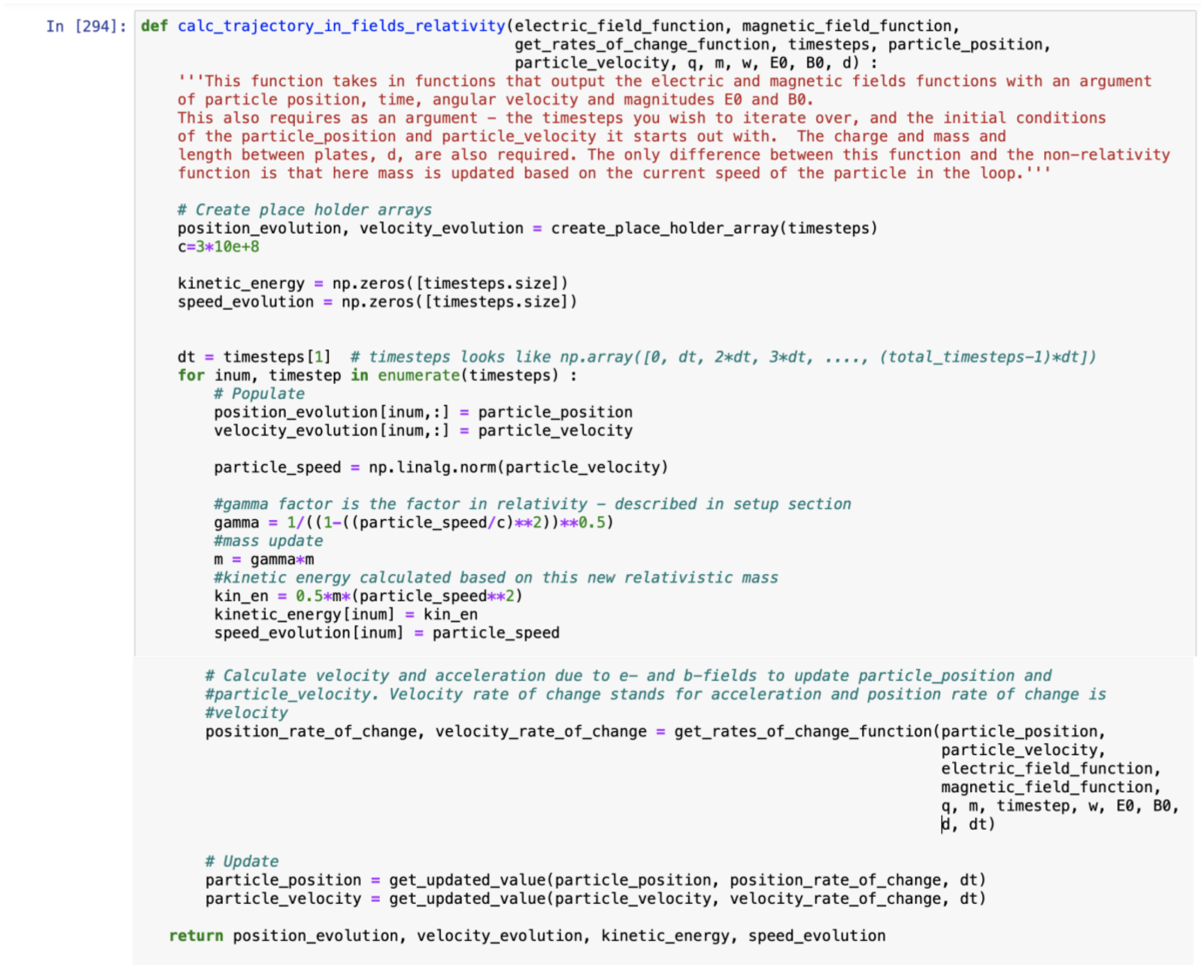
```
In [291]: def create_place_holder_array(timesteps) :
              ''' Returns a placeholder array. 1 Arg - timesteps because the length of the placeholder array must
              be the same as the number of timesteps.'''
              position_evolution = np.zeros([timesteps.size, 3])
              velocity_evolution = np.zeros([timesteps.size, 3])
              return position_evolution, velocity_evolution
```

This function is used to create arrays to store position and velocity values at different instants in time. This is useful when plotting trajectory, kinetic energy and speed.

Trajectory Functions (Relativistic and Non-Relativistic)

```
In [293]: def calc_trajectory_in_fields(electric_field_function, magnetic_field_function, get_rates_of_change_function,
                                        timesteps, particle_position, particle_velocity, q, m, w, E0, B0, d) :
              '''This function takes in functions that output the electric and magnetic fields functions with an argument
              of particle position, time, angular velocity and magnitudes E0 and B0.
              This also requires as an argument - the timesteps you wish to iterate over, and the initial conditions
              of the particle_position and particle_velocity it starts out with.  The charge and mass and
              length between plates, d, are also required'''

              # Create place holder arrays for position, velocity, kinetic energy and speed.
              position_evolution, velocity_evolution = create_place_holder_array(timesteps)
              kinetic_energy = np.zeros([timesteps.size])
              speed_evolution = np.zeros([timesteps.size])


              dt = timesteps[1]  # timesteps looks like np.array([0, dt, 2*dt, 3*dt, ...., (total_timesteps-1)*dt])
              for inum, timestep in enumerate(timesteps) :
                  # Populate the position, velocity, speed and kinetic energy
                  position_evolution[inum,:] = particle_position
                  velocity_evolution[inum,:] = particle_velocity
                  particle_speed = np.linalg.norm(particle_velocity)
                  kin_en = 0.5*m*(particle_speed**2)
                  kinetic_energy[inum] = kin_en
                  speed_evolution[inum] = particle_speed

                  # Calculate velocity and acceleration due to e- and b-fields to update particle_position and
                  #particle_velocity. Velocity rate of change stands for acceleration and position rate of change is
                  #velocity
                  position_rate_of_change, velocity_rate_of_change = get_rates_of_change_function(particle_position,
                                                                                                  particle_velocity,
                                                                                                  electric_field_function,
                                                                                                  magnetic_field_function,
                                                                                                  q, m, timestep, w, E0, B0, d

                  # Update
                  particle_position = get_updated_value(particle_position, position_rate_of_change, dt)
                  particle_velocity = get_updated_value(particle_velocity, velocity_rate_of_change, dt)

              return position_evolution, velocity_evolution, kinetic_energy, speed_evolution
```

This is the first main function that brings together all the previously defined functions. We have timesteps array that is initialized and used to create a loop to fill in values for the position, velocity, speed and kinetic energy arrays defined. Inside the loop, after filling in one entry of our position, velocity, speed and kinetic energy arrays (corresponding to the current timestep value), we use the current values of position and velocity to calculate rates of change values for each respectively, using the runge-kutta rates of change function. We then use these rates of change values for position and velocity to calculate updated position and velocity of the particle using the get_updated_values function defined earlier.
In this manner, we completely fill in our arrays that now hold the details of the particle in its trajectory.



```
In [294]: def calc_trajectory_in_fields_relativity(electric_field_function, magnetic_field_function,
                                                    get_rates_of_change_function, timesteps, particle_position,
                                                    particle_velocity, q, m, w, E0, B0, d) :
              '''This function takes in functions that output the electric and magnetic fields functions with an argument
              of particle position, time, angular velocity and magnitudes E0 and B0.
              This also requires as an argument - the timesteps you wish to iterate over, and the initial conditions
              of the particle_position and particle_velocity it starts out with.  The charge and mass and
              length between plates, d, are also required. The only difference between this function and the non-relativity
              function is that here mass is updated based on the current speed of the particle in the loop.'''

              # Create place holder arrays
              position_evolution, velocity_evolution = create_place_holder_array(timesteps)
              c=3*10e+8

              kinetic_energy = np.zeros([timesteps.size])
              speed_evolution = np.zeros([timesteps.size])


              dt = timesteps[1]  # timesteps looks like np.array([0, dt, 2*dt, 3*dt, ...., (total_timesteps-1)*dt])
              for inum, timestep in enumerate(timesteps) :
                  # Populate
                  position_evolution[inum,:] = particle_position
                  velocity_evolution[inum,:] = particle_velocity

                  particle_speed = np.linalg.norm(particle_velocity)

                  #gamma factor is the factor in relativity - described in setup section
                  gamma = 1/((1-((particle_speed/c)**2))**0.5)
                  #mass update
                  m = gamma*m
                  #kinetic energy calculated based on this new relativistic mass
                  kin_en = 0.5*m*(particle_speed**2)
                  kinetic_energy[inum] = kin_en
                  speed_evolution[inum] = particle_speed

                  # Calculate velocity and acceleration due to e- and b-fields to update particle_position and
                  #particle_velocity. Velocity rate of change stands for acceleration and position rate of change is
                  #velocity
                  position_rate_of_change, velocity_rate_of_change = get_rates_of_change_function(particle_position,
                                                                      particle_velocity,
                                                                      electric_field_function,
                                                                      magnetic_field_function,
                                                                      q, m, timestep, w, E0, B0,
                                                                      d, dt)

                  # Update
                  particle_position = get_updated_value(particle_position, position_rate_of_change, dt)
                  particle_velocity = get_updated_value(particle_velocity, velocity_rate_of_change, dt)

              return position_evolution, velocity_evolution, kinetic_energy, speed_evolution
```

This is the trajectory function, but with the added component of relativistic effects. The function is the same as the non-relativistic function except the fact that mass is also updated in the loop depending on the current value of speed. This mass update comes from the relativistic mass formula:

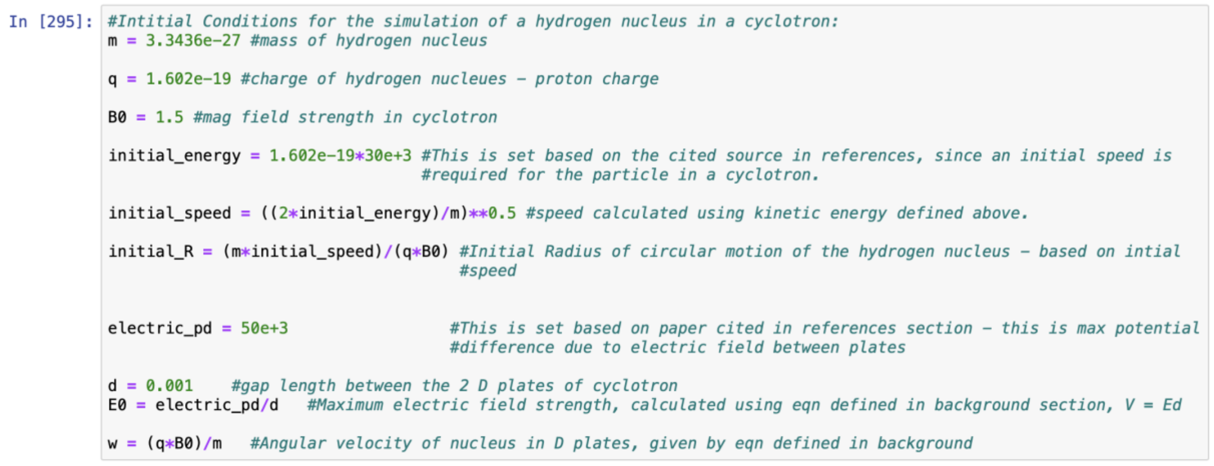
$$m = \gamma m_0$$

Where m_0 is the initial rest mass of the body. And gamma is the Lorentz factor that depends on the current speed of the particle (from Physics 160)

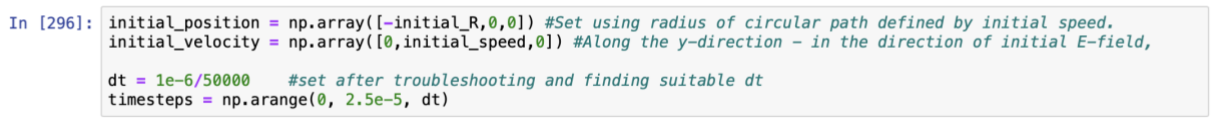
$$\gamma = \frac{m_0}{\sqrt{(1-\frac{v^2}{c^2})}}$$

Now that we have all our functions defined, we can move to the 2 cases that were analyzed in this project.

Case 1: Hydrogen Nucleus in a Cyclotron:



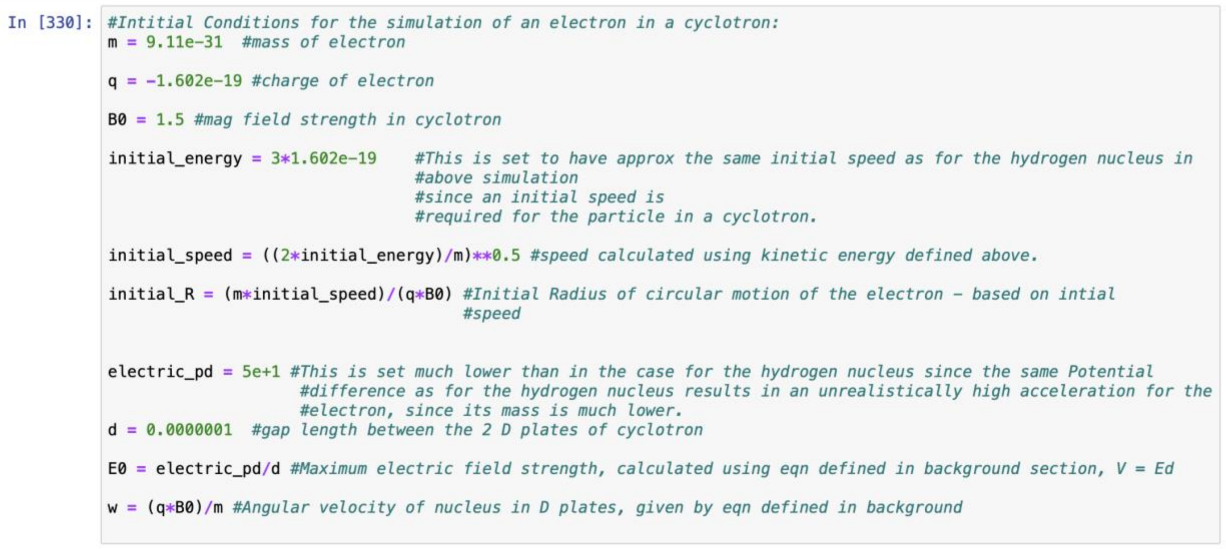
```
In [295]: #Intitial Conditions for the simulation of a hydrogen nucleus in a cyclotron:
          m = 3.3436e-27 #mass of hydrogen nucleus

          q = 1.602e-19 #charge of hydrogen nucleues - proton charge

          B0 = 1.5 #mag field strength in cyclotron

          initial_energy = 1.602e-19*30e+3 #This is set based on the cited source in references, since an initial speed is
                                            #required for the particle in a cyclotron.

          initial_speed = ((2*initial_energy)/m)**0.5 #speed calculated using kinetic energy defined above.

          initial_R = (m*initial_speed)/(q*B0) #Initial Radius of circular motion of the hydrogen nucleus - based on intial
                                                #speed


          electric_pd = 50e+3                  #This is set based on paper cited in references section - this is max potential
                                               #difference due to electric field between plates

          d = 0.001    #gap length between the 2 D plates of cyclotron
          E0 = electric_pd/d    #Maximum electric field strength, calculated using eqn defined in background section, V = Ed

          w = (q*B0)/m   #Angular velocity of nucleus in D plates, given by eqn defined in background
```



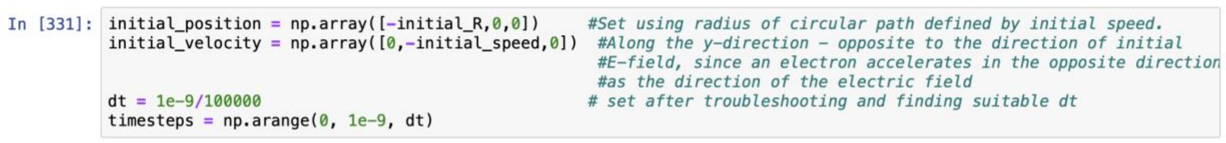
```
In [296]: initial_position = np.array([-initial_R,0,0]) #Set using radius of circular path defined by initial speed.
          initial_velocity = np.array([0,initial_speed,0]) #Along the y-direction - in the direction of initial E-field,

          dt = 1e-6/50000    #set after troubleshooting and finding suitable dt
          timesteps = np.arange(0, 2.5e-5, dt)
```

These were the values that were initialized for the hydrogen nucleus. Mass is of course the mass of a proton + neutron, and charge is of course the charge of a proton, since this is what constitutes a hydrogen nucleus.
Initial Energy and maximum potential difference between the 2 plates was set using 'Cyclotrons and their Applications'2
Other parameters were initialized using equations defined the background section, and dt was chosen after troubleshooting and determining approximately how long it would take for this particle to leave the cyclotron (at a speed that corresponded to a radius of circular motion that was larger the radius of the D-plates of the cyclotron.

Case 2: Electron in a Cyclotron:



```
In [330]: #Intitial Conditions for the simulation of an electron in a cyclotron:
          m = 9.11e-31  #mass of electron

          q = -1.602e-19 #charge of electron

          B0 = 1.5 #mag field strength in cyclotron

          initial_energy = 3*1.602e-19    #This is set to have approx the same initial speed as for the hydrogen nucleus in
                                          #above simulation
                                          #since an initial speed is
                                          #required for the particle in a cyclotron.

          initial_speed = ((2*initial_energy)/m)**0.5 #speed calculated using kinetic energy defined above.

          initial_R = (m*initial_speed)/(q*B0) #Initial Radius of circular motion of the electron - based on intial
                                                #speed


          electric_pd = 5e+1 #This is set much lower than in the case for the hydrogen nucleus since the same Potential
                              #difference as for the hydrogen nucleus results in an unrealistically high acceleration for the
                              #electron, since its mass is much lower.
          d = 0.0000001  #gap length between the 2 D plates of cyclotron

          E0 = electric_pd/d #Maximum electric field strength, calculated using eqn defined in background section, V = Ed

          w = (q*B0)/m #Angular velocity of nucleus in D plates, given by eqn defined in background
```

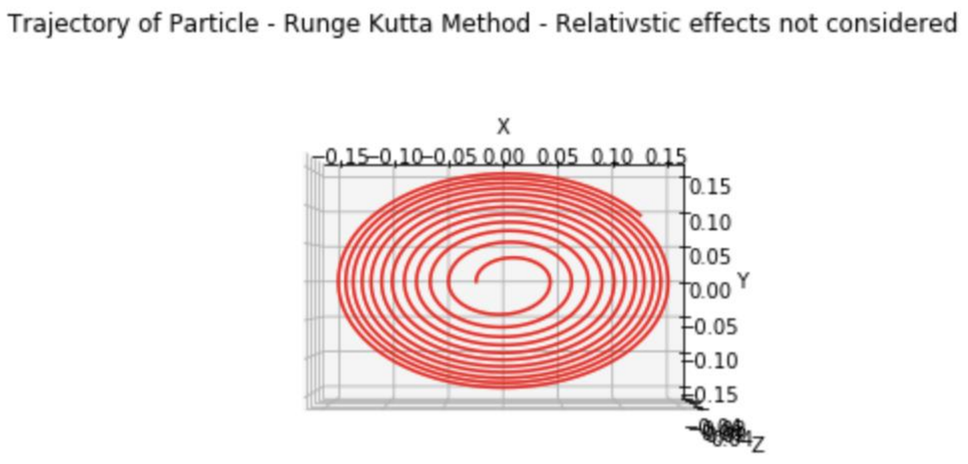


```
In [331]: initial_position = np.array([-initial_R,0,0])     #Set using radius of circular path defined by initial speed.
          initial_velocity = np.array([0,-initial_speed,0])  #Along the y-direction - opposite to the direction of initial
                                                              #E-field, since an electron accelerates in the opposite direction
                                                              #as the direction of the electric field
          dt = 1e-9/100000                                  # set after troubleshooting and finding suitable dt
          timesteps = np.arange(0, 1e-9, dt)
```

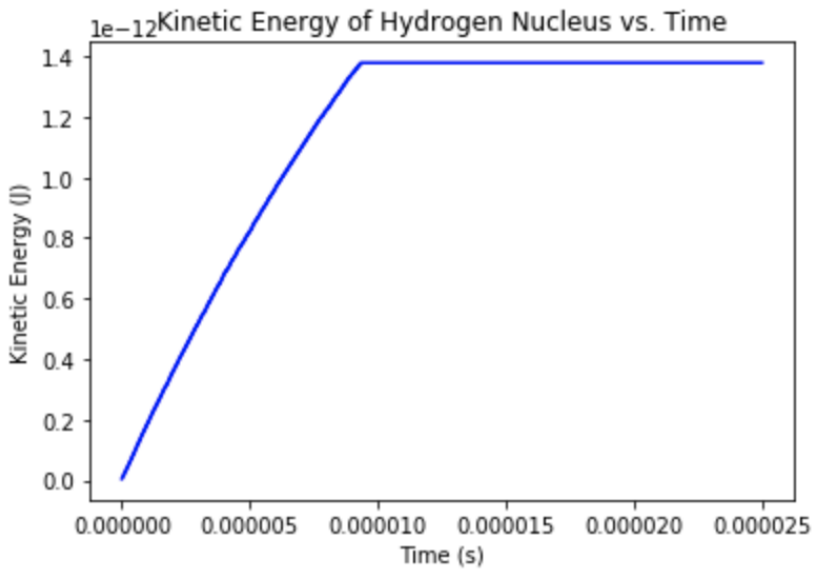
The origin of all these initialized values have been suitably described in the comments. Initial energy value was chosen in a way that the initial speed of the electron corresponded to the initial speed of the hydrogen nucleus in case 1. Also, maximum potential difference value was adjusted based on what a suitable amount of acceleration would be for the electron. This is because the same value of potential difference as in the case of the hydrogen nucleus would result in a much larger increase in electron velocity, since the mass of an electron is negligible compared to that of a hydrogen nucleus. (The reason such large speed jumps could not be used is described in the results and evaluation section).

Results and Evaluation:

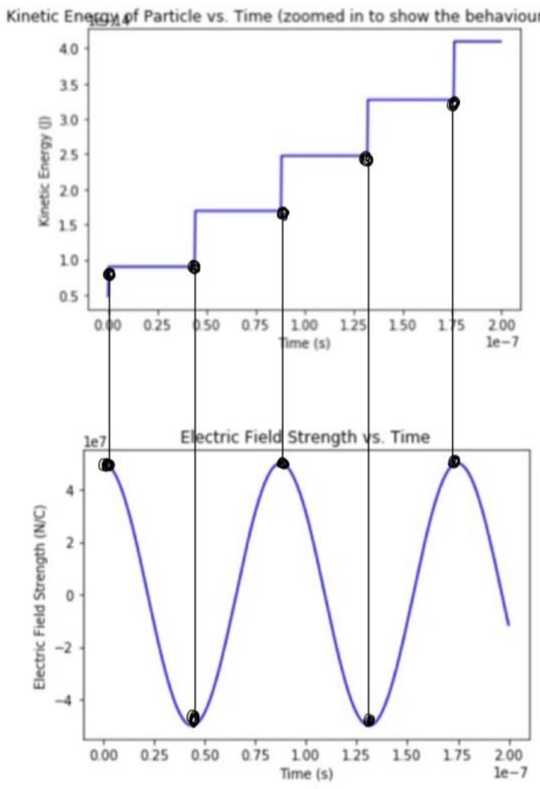
Case 1: Hydrogen Nucleus in Cyclotron



For the case of the hydrogen nucleus, we did not consider relativistic effects and got a graph that matched what was expected. An outward spiraling trajectory in a single plane. The particle moves in circular motion in the D plates, increases speed as it accelerates in the electric field region between the plates, and then continues circular motion (with a larger radius) as it enters the next D plate.



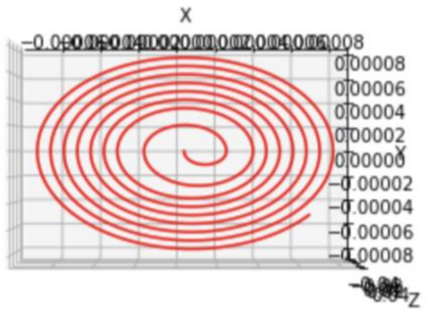
This is the graph showing the evolution of kinetic energy. It is approximately linear in the beginning region, since each time the particle passes through the region between the D – plates, its energy increases by the same amount (Field Strength * d). The flat region symbolizes the time after the particle has left the cyclotron due to its radius of circular motion now exceeding the radius of the D plates.



This a zoomed in version of the linear portion of the previous Kinettic Energy vs. Time graph. This graph compared directly the changing electric field graph. As we can see, the frequency of the alternating electric field has been set to be the frequency of the circular motion of the particle in the D plates, which results in the kinetic energy increases to happen in exactly the points where magnitude of electric field is maximum. Since the particle crosses the region between the D plates at these instants of maximum magnitude of electric field in the direction of motion of the hydrogen nucleus particle.

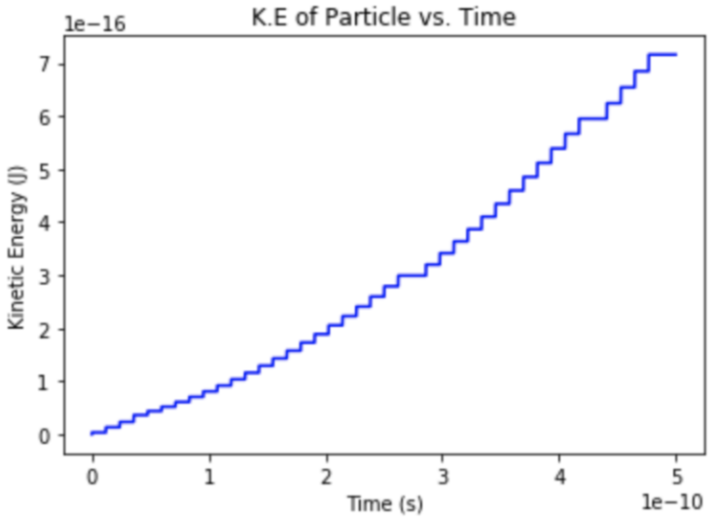
Case 2: Electron in Cyclotron:
For this case, we evaluated how results change if we add the component of relativity.

Non-Relativistic Results:



Trajectory of electron in Cyclotron

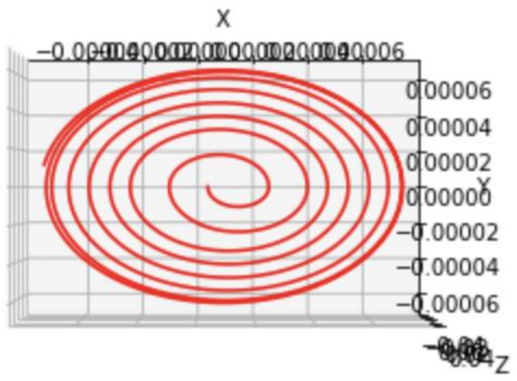
This is very similar behavior to the case of the hydrogen nucleus in the cyclotron. It is what we expect under ideal condition, an outward spiraling trajectory in a single plane.



The electron’s kinetic energy is also almost linear and thus similar to the case of the hydrogen nucleus.

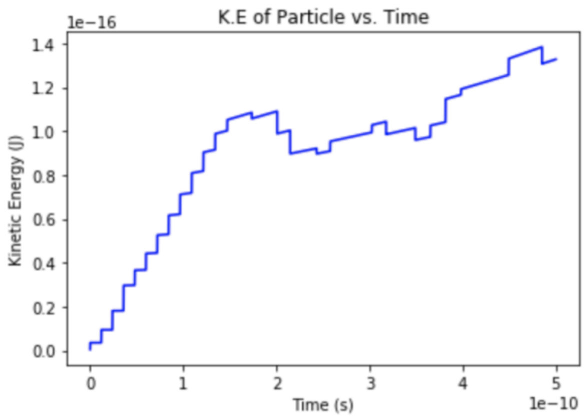
Now let us see what happens when we add the effects of relativity:

Relativistic Results:

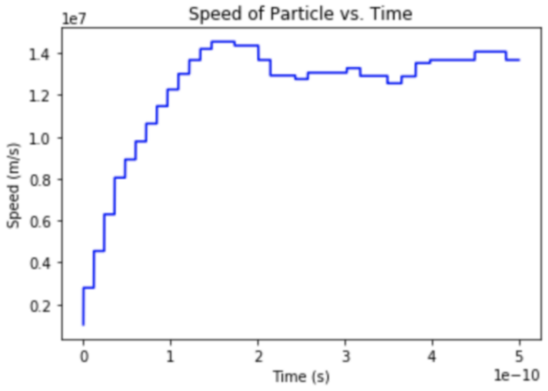


Trajectory of electron – Relativistic effects consdiered

As evident in this case, the particle seems to not behave the wave we intend it to towards the edge, as the trajectory seems to be overlapping. This will make more sense when we look at the graph of kinetic energy.



Here we see the Kinetic energy doesn't keep increasing like we want it to. There are periods where it even starts decreasing. Even when we look the speed graph:



There are regions where the speed decreases. This is because, due to relativistic effects, mass changes and this results in a change the angular velocity and thus frequency of circular motion of the electron (since we showed the angular frequency depends on the q/m ratio and here the q remains constant but m changes). Thus, this causes the electron to go out of phase with the alternating electric fields that have been set at a constant frequency. When the particle and electric field go completely out of phase, the electric field ends up being in the same direction to the motion of the electron as it passes through the gap between the D plates. This results in the force on the electron to end up being in the opposite direction of its velocity, and thus the speed and kinetic energy of the electron end up reducing.

We can also point out that these relativistic effects come into play at higher velocities. Initially, as evident from the graphs of speed and kinetic energy vs time, we see that the behavior is similar to that of the non-relativistic simulation. The Kinetic energy seems to be increasing approximately linearly. It is only at higher speeds that mass changes considerably, causing a considerable change in angular frequency of the electrons circular motion in the D plates, which in turn causes the electron's motion to go out of phase with the alternating electric field. This highlights one of the big disadvantages of a cyclotron. A synchrocyclotron on the other hand, takes care of this issue since it involves an electric field that alternates with a changing frequency based on the changing frequency of the particles circular motion[3].

Also, we note that even though we considerably reduced the maximum electric potential energy between the plates for the case of the electron, the electron accelerated suitably and reached similar speeds to that of the hydrogen nucleus. This is because the mass of an electron is much lower than that of a hydrogen nucleus and thus the same amount of kinetic energy added to an electron ends up amounting a much larger increase in speed. This is not realistic, since the speed of light is the limit for speeds and thus at extremely high speeds close the speed of light, relativistic effects become extremely large.

References:

1. "Collection of Solved Problems in Physics." *Cyclotron - Collection of Solved Problems*, physicstasks.eu/551/cyclotron.

2. "Cyclotron Description and Parameters." *Cyclotrons and Their Applications: Proceedings of the 14th International Conference: Cape Town, South Africa, 8-13 October 1995*, by John C. Cornell, World Scientific, 1996, pp. 129–130.

3. "Cyclotron." *Wikipedia*, Wikimedia Foundation, 17 Apr. 2020, en.wikipedia.org/wiki/Cyclotron.

4. More, Hemant. "Cyclotron: Particle Accelerator. Principle, Construction and Working." *The Fact Factor*, 4 Jan. 2020, thefactfactor.com/facts/pure_science/physics/particle-accelerator-cyclotron/5916/.

5. Mraz, Stephen. "What Are the Differences Between Linear Accelerators, Cyclotrons, and Synchrotrons?" *StackPath*, 3 Feb. 2017, www.machinedesign.com/learning-resources/whats-the-difference-between/article/21832184/what-are-the-differences-between-linear-accelerators-cyclotrons-and-synchrotrons.