### I Introduction

Our idea was to simulate a particle accelerator colliding electrons and positrons (e- and e+).We mostly focused on the physics of the collision and particles subsequently produced rather than the actual acceleration of the particles to high energies or detecting the products of their annihilation. Basically, we modeled each particle's path using differential equations, and use random probabilities/Monte Carlo simulation to determine their mass, lifetime, and decays, while conserving energy and momenta.

First, we modeled e- and e+ colliding in a detector with some energy (in our simulation, 2.5 GeV energy each). When the particles collided, they annihilated and the particles produced were random but based on some probability. In this collision, energy and momentum from the original particles was conserved and imparted to their "daughter" particles. These particles then traveled through the detector with some mass, energy, momentum, etc. A magnetic field was present in the detector causing charged particles to travel in a helix, while uncharged particles simply travel in a straight line. Unstable particles have a finite lifetime with average lifetimes ranging from extremely small values, like 4.5\*10-24 seconds for rho mesons, to very large values, such as 881 seconds for free neutrons. Each timestep, these daughter particles had some probability of decaying. When decaying, a particle spontaneously changes from themselves into two or more daughter particles. An example is that a D0 may decay to a kaon (K-) and a pion (pi+). Both aspects are based on probabilities dependent on the individual particles. In the case of D0, there is a 4% chance that it will decay to K- and pi+. Again, charge, energy, etc. were conserved in this decay. Eventually, either stable particles were created or a particle reached the detector. Information on the particles' masses, widths, lifetimes, and decay probabilities are available from the particle data group at<http://pdg.lbl.gov/2015/listings/contents_listings.html> .

### II Code Description

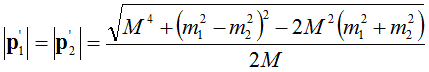
Our code attempts to simulate the physics of particles travelling, colliding, and decaying in a detector. First, we needed a way to represent particles. We did this by creating a class called Particle (in particle\_class.py) that accepts four parameters: the name of the particle, its mass, its charge, and its lifetime. The Particle class has several methods for “linking” particles to their parents and daughters: append and append\_parent set the links, while daughter and parent return the linked particles. We used the set\_pos and set\_v methods to set the positions and velocities of particles. set\_v also set the momentum, energy, and the lorentz factor gamma. Last is the boost method, which accepts a particle as a parameter and boosts the original particle to the lab frame from the rest frame of the parent particles.

Once we were able to represent a particle, we moved on to the particle\_functions.py file to show the functions we could use on these particles. First, we needed to be able to make a particle. We know what charge a particle has, however the mass and the lifetime are not definite, but are instead based on probability distributions (Breit Wigner for mass and exponential for lifetime). The width of the distributions are based on the lifetime of the particle. Therefore, to make a particle, we randomly generated a number based on these probability distribution to determine the mass and actual lifetime of the particles (in the mass\_charge\_lifetime function), then initialized the particle ( in the make\_particle function). Second, we had to be able to make a particle move. In order to do this, we used the path function which takes the initial position, velocity, mass, charge, lifetime, and magnetic field and solves the set of differential equations

dx/dt = vx dy/dt = vy dz/dt = vz

dvx/dt = 0 dvy/dt = vz\*q\*B/m dvz/dt = -vy\*q\*B/m

It then set the position and velocity of the particles to the output. Once we were able to make out particles move around in our detector, we needed to simulate their decays. To simplify this, we made all of our calculations in the center of mass frame (or the parent’s rest frame). We solved for the energy and the magnitudes of the momenta using the following equations (primes indicate center of mass frame, M is the mass of the parent, m1 and m2 are the masses of the daughter particles):

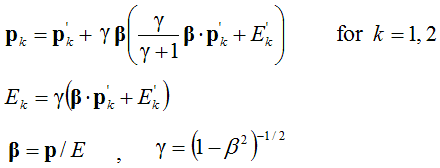


After this, we got random angles (phi and theta) to determine the direction the particles would travel in which gave us the final momenta:

We then converted the momenta and energy to velocity using the equation:



We also set the position of both daughter particles to be the same as the position where the parent particle decayed. The last step was to perform a lorentz boost. This was done using the boost method of our particle class. When performing a lorentz boost, the energy and momenta for each daughter particle can be solved using the equations (k=1,2 for the first and second daughter):



We then converted the momenta and energy to velocity using the same equation from earlier, and set all of the values of the particle.

We were able to make particles move and decay, leaving us to have to create a function that chooses what particles we wanted to have traveling through our detector. We use the decay chain function to do this. In this function, we created the initial conditions for our collision and added each particle we created to a list. Then, if the particle had a short lifetime we called the make\_decay function which randomly chose a decay for the particle based on the probabilities in our decay\_dict. It created both particles and linked the parent and daughters together. If the particle had a long lifetime, we moved on to the next particle. We continued to do this until all the particles had too long of a lifetime to decay in the timeframe of our simulation. Last, the decay\_chain function returned the array of all the particles.

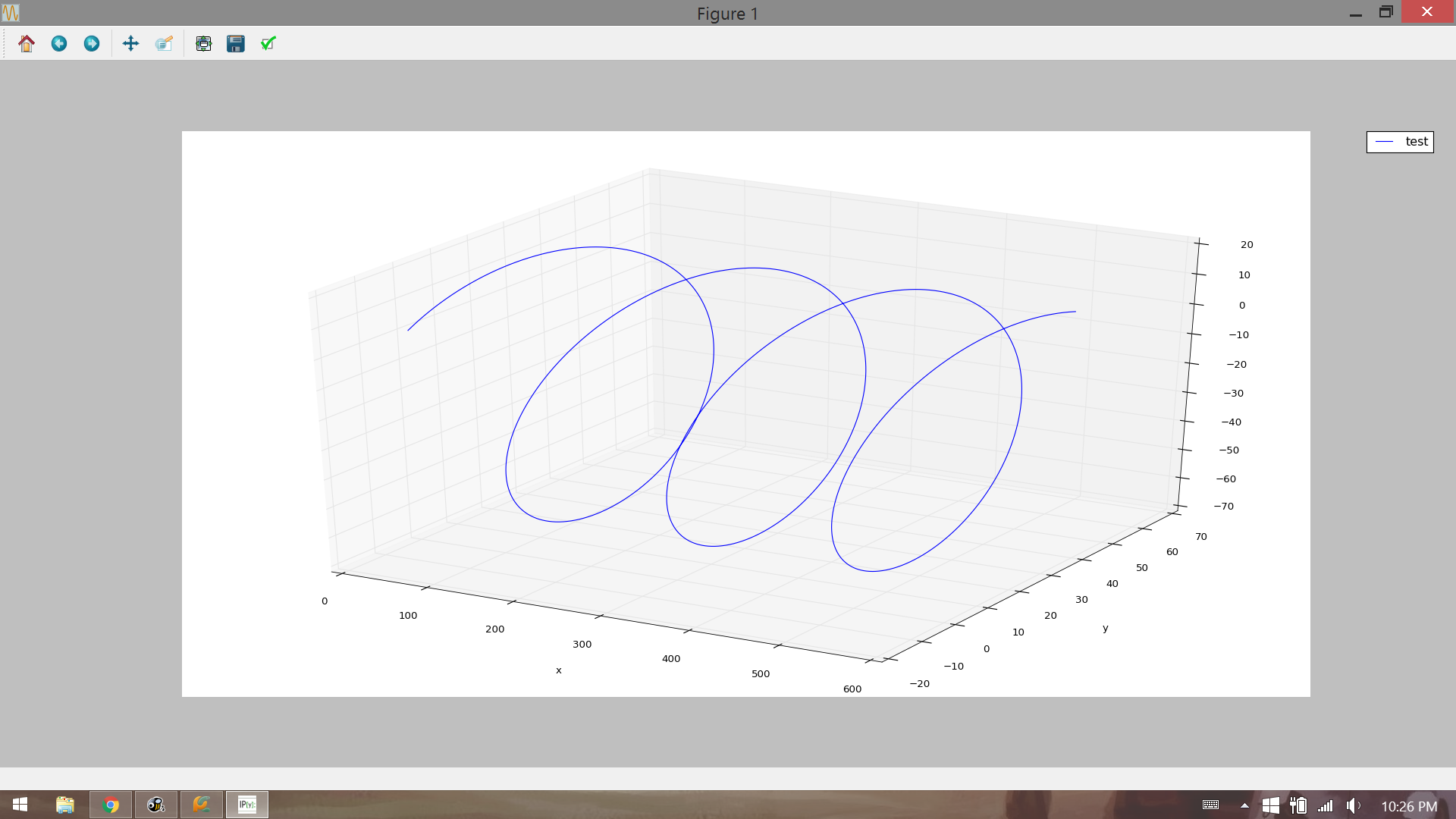
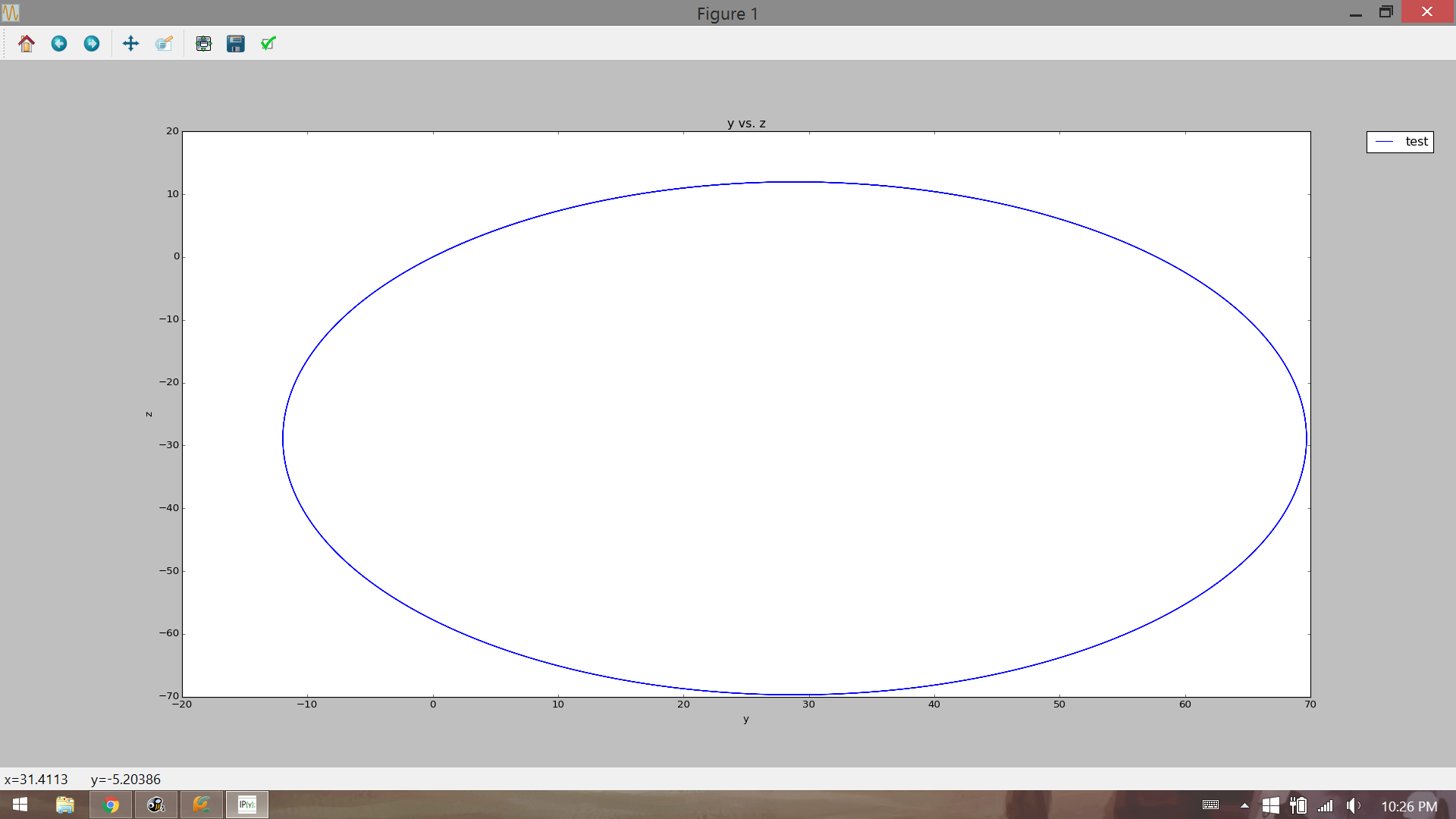
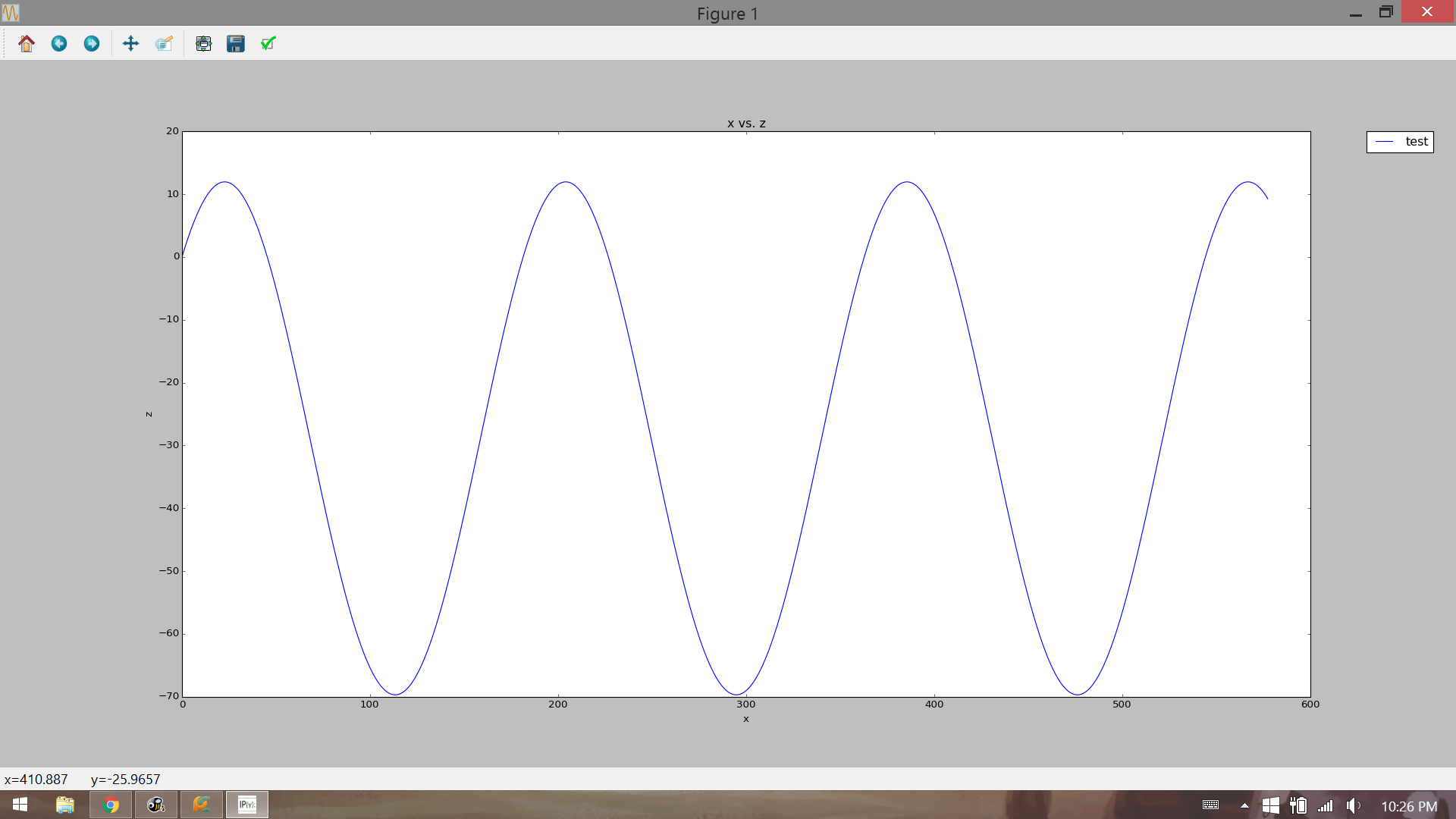
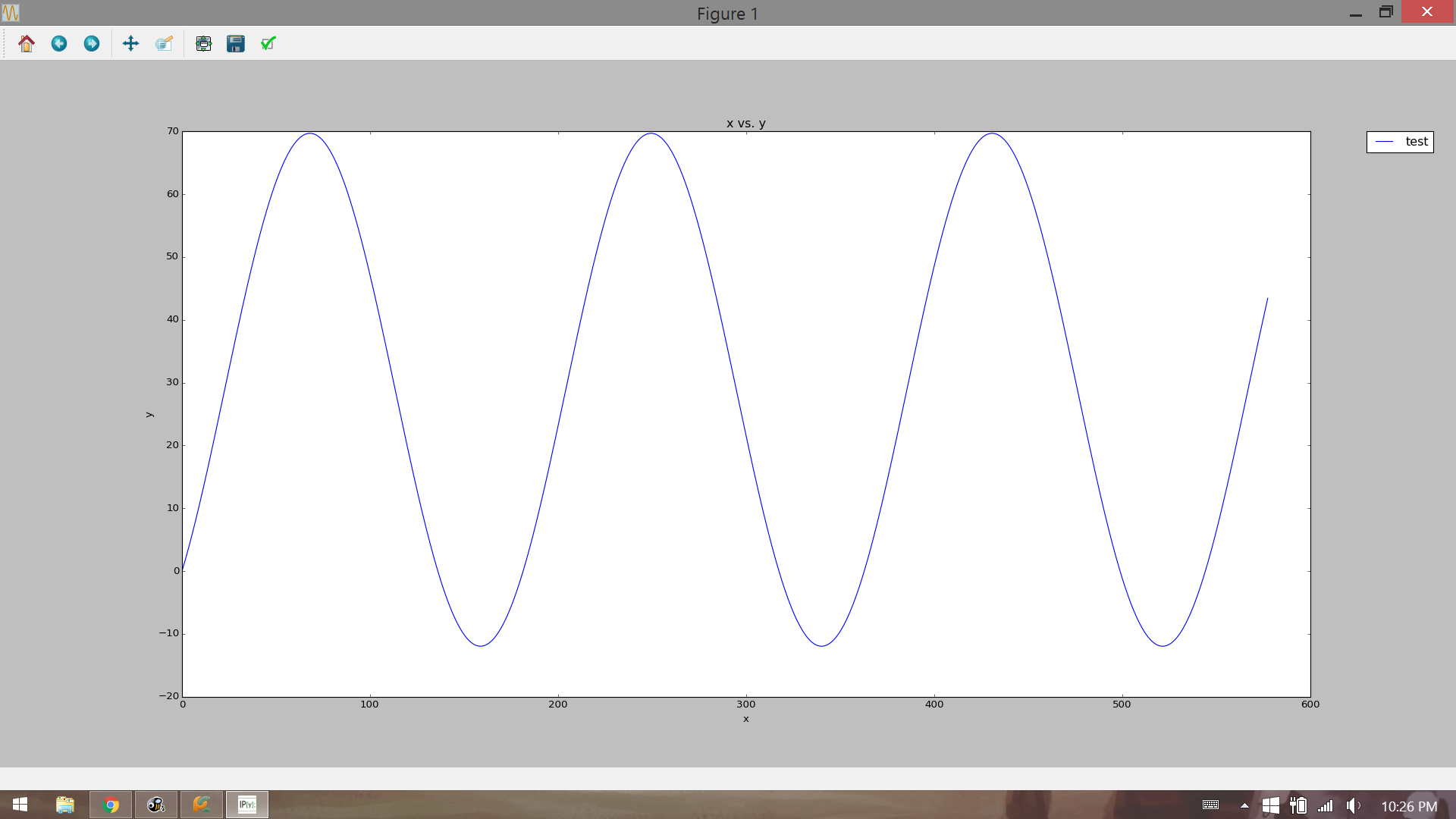
Last was the main function ( in particle\_main.py), which called the decay\_chain function to get a list of particles. It then iterated over each particle, calling the path function for every particle and the two\_body\_decay function for particles which have daughter particles. A 3D graph of the path of every particle was printed along with the mass, charge, lifetime, energy, and momentum of every particle.

Also, we built a operable interface for this project. It contains a class to define all frames and a special page which will contain all the functions that have been constructed. Thus most of the function and the dictionary of the particles are copied and put at the very beginning of the whole code. Then use button in Tkinter to operate each function in order, text to output the result and canvas to separate each steps. The first frame is built for making a decay chain with previous defined functions. Also use text to show what kind of decay-chain was made. Meanwhile, because the decay chain was based on Monte Carlo random sampling method, thus it is valuable to get the detailed information about each particles in the whole process. So in the second frame we defined the plot function in x-y, x-z and y-z terms to illustrate the movement of each particles and also print characters of each particles to collect information. The PyCharm we used to build the interface, also made it’s possible to observe the 3-D graph from any position by drag it turn around. In the third frame, there is a single button for plotting constructed decay chain and a text for printing all the information of particles involved.

### III & IV Analytical Solutions and Test Cases

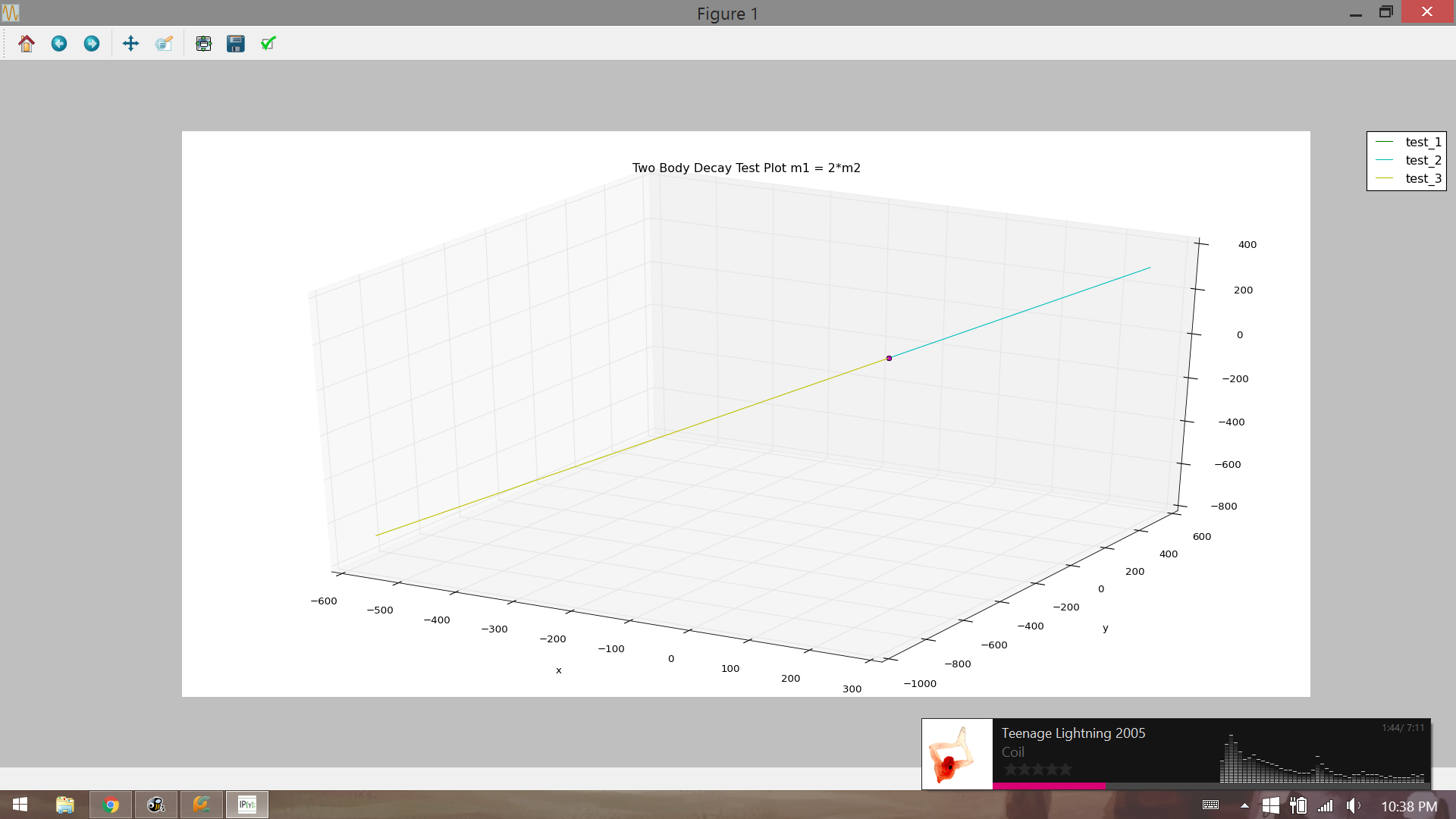
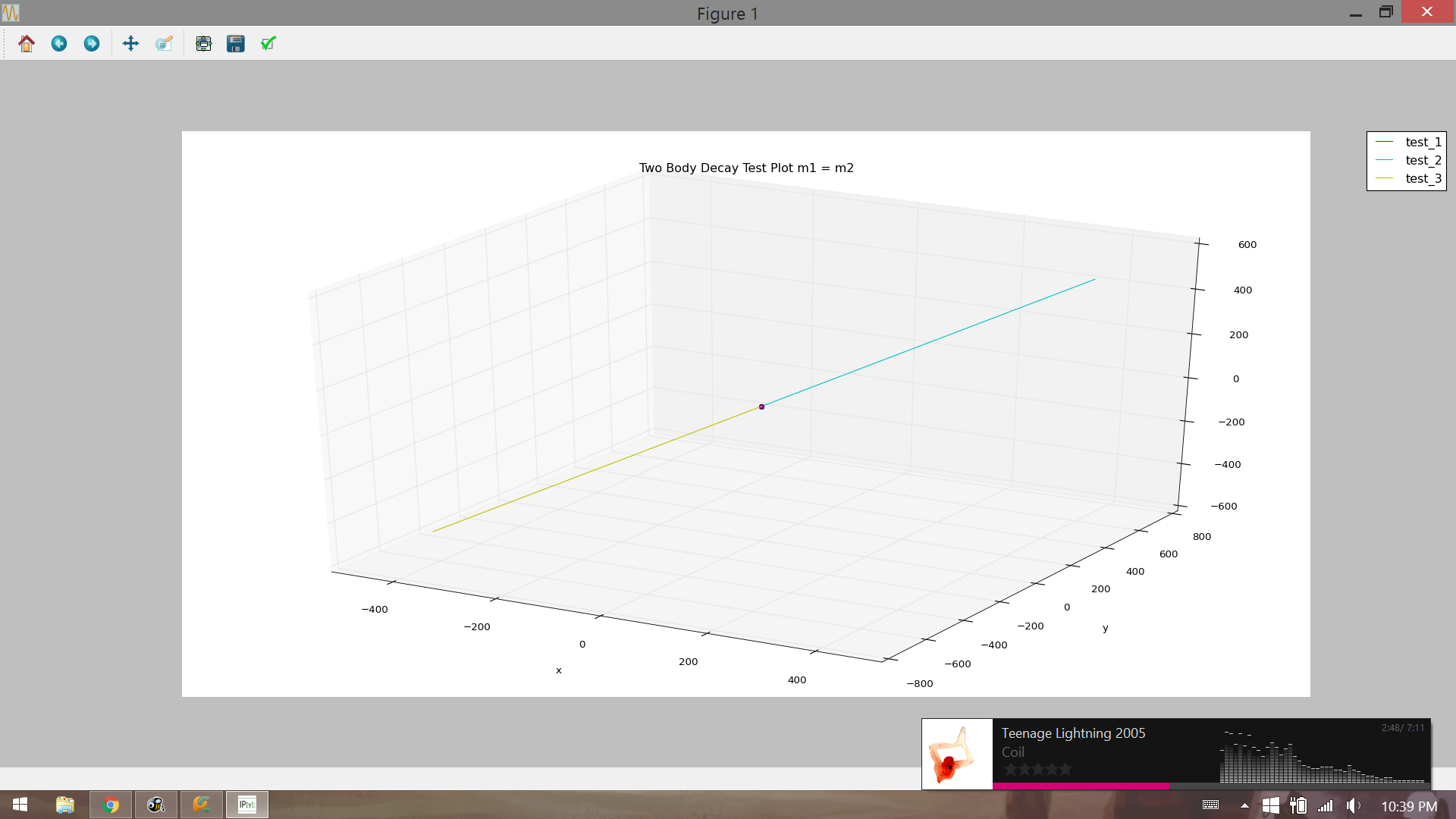
Path function:

To test our path function we simply created a charged particle with a velocity of .5\*c at the origin and called travel on it. We saw the expected sinusoidal curves in the x-y and x-z plane, and the expected circular path in the y-z plane.



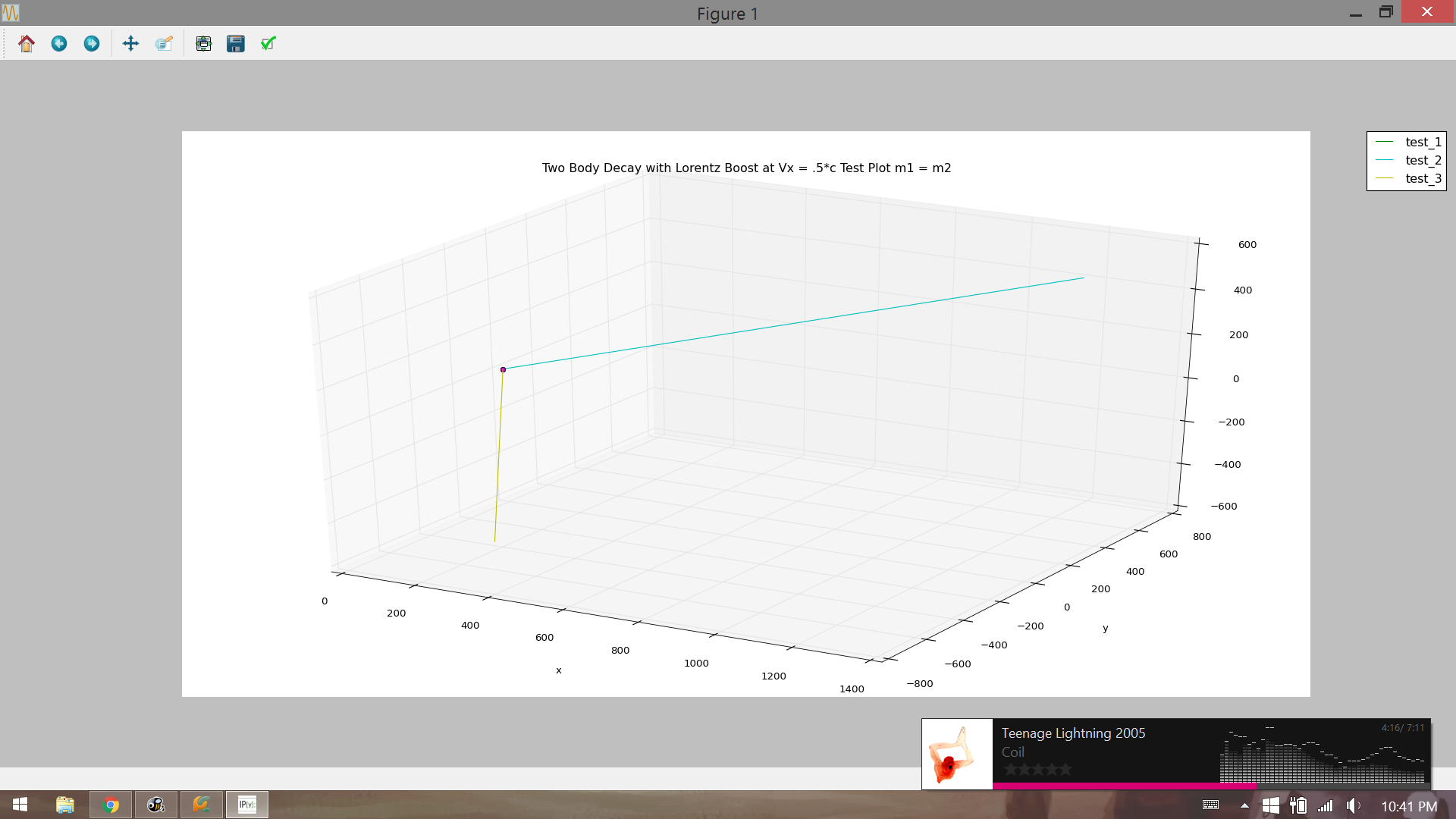
Two Body Decay Function:

To test the two body decay function we started with a particle at rest at the origin and had it decay to two particles. In one case we set the particles masses to be equal. In the second we set m1 = 2\*m2. We confirmed that energy and momentum were conserved.



Lorentz Boost:

To test the lorentz boost we made a particle at the origin travelling at .5\*c in the x direction. Using the same angles as before we let the particle decay and obtained the following decay. We also checked that energy and momentum were still conserved.



### V Full Physical System

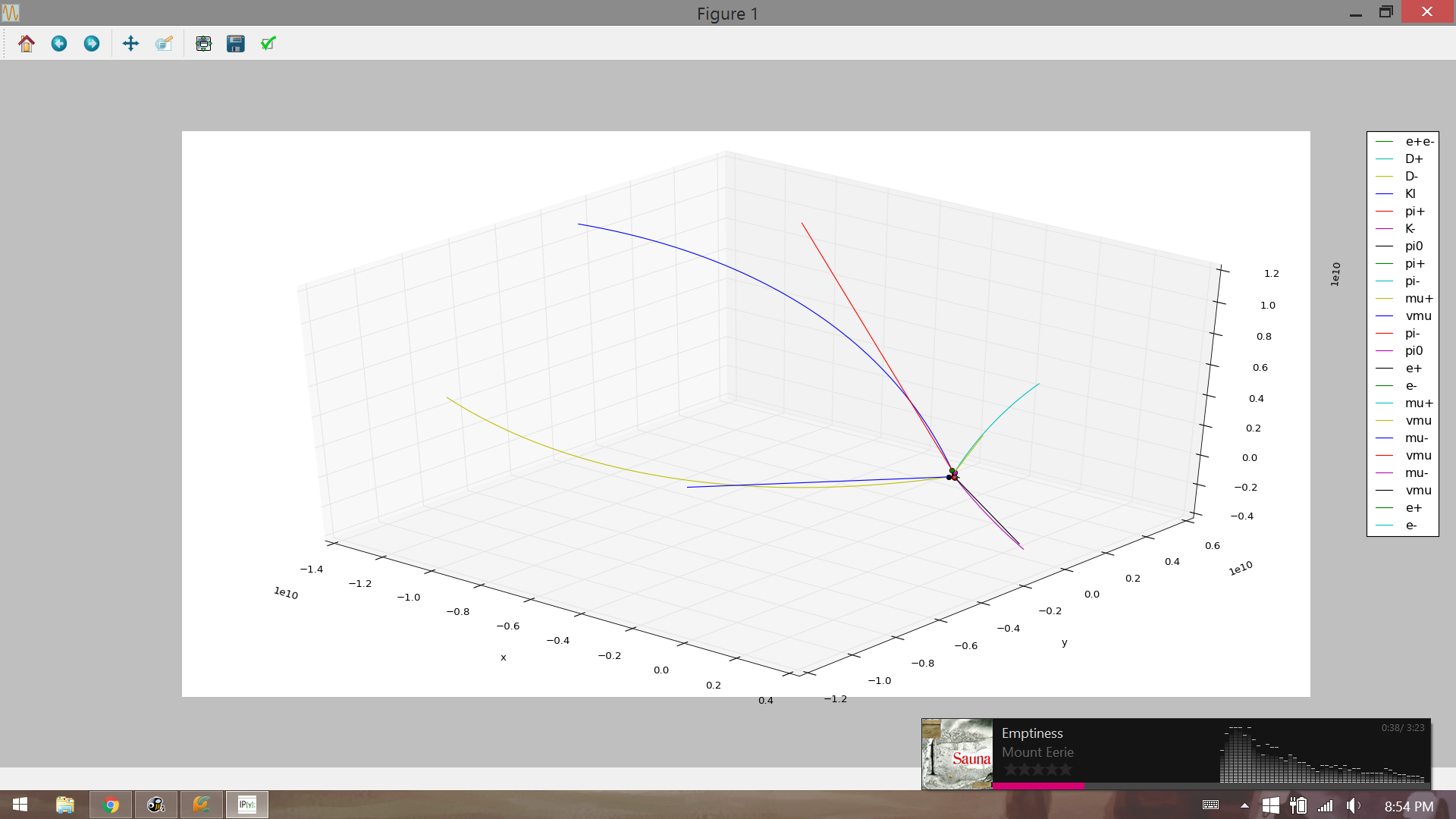
The full physical system we were trying to simulate is the collision of e- and e+ at 2.5 GeV and the particles subsequently produced in this collision. We wanted to look at these particle’s paths, masses, energies, and momenta. We specifically looked at two different scales of length, first on a macroscopic level, ~10-100 m, and second on a microscopic level, ~10-5 - 10-6 m. At the macroscopic level, our goal was to see charged particles traveling in helical paths, and neutral particles in straight paths. Almost all of the decays should happen near the origin, with the more stable particles traveling ~50 m to where a detector should realistically be. At the microscopic level, our goal was to see the flight path of the the short lived particles and to see them decay at very small distances from the origin.

### 

### 

### VI Results

Macroscopic Scale Example:



Particle Mass (MeV/c\*\*2) Charge (e) Lifetime (fs) Energy (MeV) Momentum (MeV/c)

D+ 1869.61 1 1040 2500.02 1659.72

D- 1869.61 -1 1040 2500.02 1659.72

Kl 497.611 0 5.116e+07 1583.15 1502.9

pi+ 139.57 1 2.6033e+07 916.869 906.184

K- 497.677 -1 1.238e+07 1308.85 1210.54

pi0 134.977 0 0.0852 1191.17 1183.5

pi+ 139.57 1 2.6033e+07 387.103 361.067

pi- 139.57 -1 2.6033e+07 1196.05 1187.88

mu+ 105.658 1 2.197e+09 812.926 806.031

vmu 1e-05 0 2e+08 0.00039865 0.000398525

pi- 139.57 -1 2.6033e+07 348.577 319.415

pi0 134.977 0 0.0852 960.272 950.738

e+ 0.510999 1 1e+100 935.949 935.949

e- 0.510999 -1 1e+100 254.876 254.876

mu+ 105.658 1 2.197e+09 314.286 295.994

vmu 1e-05 0 2e+08 0.000138312 0.00013795

mu- 105.658 -1 2.197e+09 899.946 893.722

vmu 1e-05 0 2e+08 0.000748999 0.000748932

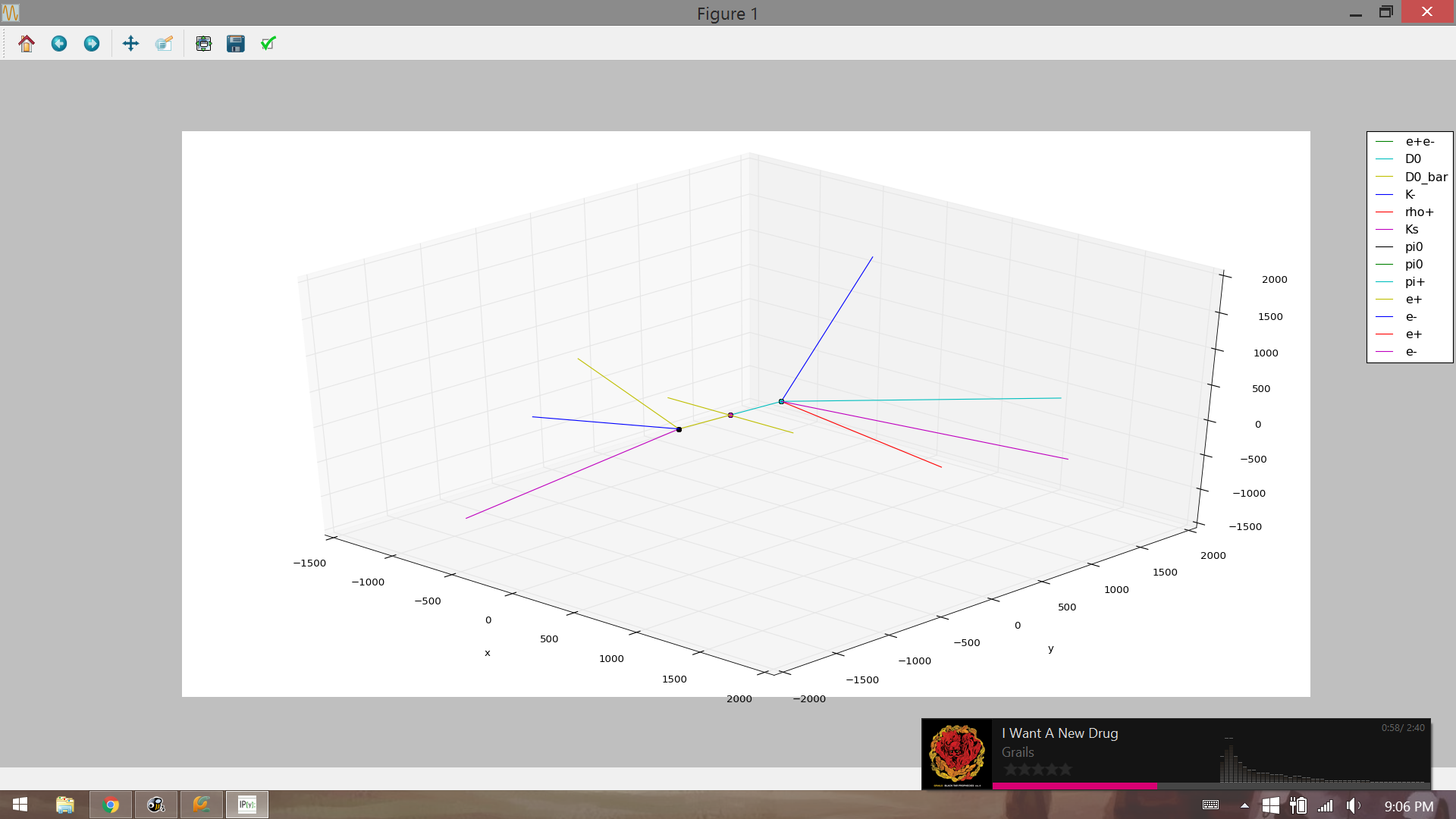
mu- 105.658 -1 2.197e+09 244.2 220.159

vmu 1e-05 0 2e+08 0.000207023 0.000206782

e+ 0.510999 1 1e+100 678.496 678.495

e- 0.510999 -1 1e+100 282.528 282.527

Microscopic Scale Example:



Particle Mass (MeV/c\*\*2) Charge (e) Lifetime (fs) Energy (MeV) Momentum (MeV/c)

D0 1864.84 0 410.1 2499.93 1664.95

D0\_bar 1864.84 0 410.1 2499.93 1664.95

K- 497.677 -1 1.238e+07 1219.09 1112.88

rho+ 621.895 1 6.22204e-08 1280.84 1119.73

Ks 497.611 0 89540 1741.53 1668.93

pi0 134.977 0 0.0852 758.401 746.293

pi0 134.977 0 0.0852 855.102 844.382

pi+ 139.57 1 2.6033e+07 425.74 402.212

e+ 0.510999 1 1e+100 460.421 460.421

e- 0.510999 -1 1e+100 297.973 297.973

e+ 0.510999 1 1e+100 70.2305 70.2286

e- 0.510999 -1 1e+100 784.748 784.748

### VII Discussion

There are a lot of interesting things going on in each example. For example, in the macroscopic example we see that all of the decays take place in a very small area in the middle. The only particles we can actually see end up being leptons because they are more stable and have much longer lifetimes than the mesons do. In the microscopic example we see that the magnetic field has almost no effect on the trajectory of the particles at such small distance scales. Also, we see that even on the microscopic scale there are still particles (rho and pi0) that decay so quickly you cannot see their paths at all which causes some decays to look more like three or four body decays even though we only simulated two body decays.

### VIII Future Directions

Because our simulation is such a basic approximation of a particle accelerator, there are seemingly infinite directions we could take. Some physics based directions include taking into account many different subtleties not included in our model, such as general relativity, electric fields, beam dynamics, and asymmetric e+ e- energies. Some less physics based directions would include animating the simulation, or showing the detector response. The two directions that we would most like to take from this point would involve including three (or more) body decays in the simulation and adding more particles and decays to the simulation. The kinematics of three body decays are more complicated than two body decays, however it would allow us to include a much greater number of decays in our simulation. Meanwhile, adding new particles and decays is fairly simple and would help create a more realistic simulation.

We can also design some entries in the interface to make it possible to change previously defined value in the dictionary. Thus it will be able to satisfy all the conditions that could happen in the realistical experiments. What’s more is that all the conditions in this models could be changed either, such as the amplitude of magnetic field. Now the magnetic field is extremely high for a convenient observation, but it will not be such high in real experiments. Thus if it is changeable with a dynamic graph, then it will offer more details in the result.

### IX References

<http://pdg.lbl.gov/>

<http://www1.gantep.edu.tr/~bingul/simulation/twoBody/>

<http://www.helsinki.fi/~www_sefo/phenomenology/Schlippe_relativistic_kinematics.pdf>

<http://ph381.edu.physics.uoc.gr/Particle_Accelerator_Physics.pdf>

<http://hyperphysics.phy-astr.gsu.edu/hbase/relativ/relcon.html>

### 

### X Code

* particle\_class.py
  + Contains the particle class and all methods for the particle class (lorentz boost function is located here)
* particle\_functions.py
  + Contains the path, two\_body\_decay, mass\_charge\_lifetime, and decay\_chain functions
  + Lines 35/36, 209/210, 212/213 determine if the simulation is macroscopic (with 35, 209, and 212 uncommented) or microscopic (with 36, 210, and 213 uncommented)
* particle\_dicts.py
  + Contains dictionaries with the masses, lifetimes, and decays of particles
* particle\_info.txt
  + Contains info on particles from the Particle Data Group
* particle\_main.py
  + Contains main function, running this runs the simulation
* particle\_class.ipynb
  + Contains lots of messy code and testing stuff used before we had to switch over to .py files
* PycharmProjects
  + Contain all the test code for learning Tkinter and gui in python
* 1st.py
  + Assemble all the function in the project
  + Build a interface to get the functions in order and illustrate the whole collision process

### XI Contributions

Anthony

* particle\_class.py
* particle\_functions.py
* particle\_dicts.py
* particle\_info.txt
* particle\_main.py
* particle\_class.ipynb
* Particle Accelerator Simulation.pptx

Bomin

* PycharmProjects
* 1st.py
* Particle Accelerator Simulation.pptx
* \_\_pycache\_\_