ORISS Code Walkthrough

There are four programs that are necessary to running the full code:

1. Edit\_Nick-Valverde\_V#.py

This is the main code that sets up the geometry for oriss, loads the particle on the beam, runs the solver, etc. The lines of interest (those that affect create the particle distribution and such) start at line 195 where the beam is created. At line 218 is where the *particle\_energy* is defined and below follow the different loading types.

1. Particle\_Class.py

This program creates the particle class that is used for the loading options. The structure of the program is a little confusing if you try to just look at it since it is practically backwards. I’ll explain the logic of it now.

Logic of Particle\_Class.py

Firstly, in order to create the instance for this class you need the energy you want your particle to be in and the beam you want to load it on. Once the instance is created the main function of the class called ‘*loader’* (line 150). This is what creates the particle load and returns it as ‘*load.’* This function invokes two other functions ‘*position’* and ‘velocity*’* which both return arrays for position (x,y,z) and velocity (vx,vy,vz). This two arrays are then stacked to make one array of (x,y,z,vx,vy,vz) which is then used with warp’s *beam.addparticles(x, y, z, vx, vy, vz)* command. For reference, here is the loader inputs with p being the instance:

p = MyParticle(particle\_energy, beam)

load = p.loader(‘gaussian’, num\_of\_particles = 1, sigma = (0,0,0), temperature = (0, 0),

avg\_velocities = (0, 0), avg\_coordinates = (0, 0, 0))

If a ‘*gaussian’* distribution is selected there is nothing further. If ‘*uniform\_ellipsoid’* is set as the distribution the ‘*position’* and ‘*velocity’* functions (lines 67 & 107 respectively) will be called bringing me to the next necessary function.

1. fill\_ellipse.py

This function is pretty straight forward. It uses the distribution notes on Dropbox to create particles in position and velocity space that satisfy the condition. It then returns ‘*position­\_array’* and ‘*velocity\_array’*.

1. plotting.py

This function is responsible for reading the trajectoryfile.txt generated by the main code and then creating the plots. This function puts all the information into a pandas dataframe which makes things a lot easier for data analysis.

From the main Oriss code the writing of trajectoryfile.txt starts at line 364 and finishes at line 399. In line 373 and 394 I formatted this txt file to be a comma separated file. Then, in line 32 of plotting.py, the file is read into a dataframe with the column labels being given by line 31.

Running the entire setup requires running 1) by itself from which running 4) will produce the plots. The example script will show how to set up different loads using the MyParticle class. This can be found in dropbox Research/ORISS/Walkthrough.

The main ORISS code (Edit\_Nick-Valverde\_V9.py) can be run right now with all the files in the right places. This will probably be a hassle, but the paths need to be altered by hand to match your paths (I don’t know how to automate this yet). For convenience I will list all the parts that need to be edited by you below. As the file is now, the program launches particles 70 particles in total where groups of 10 particles are given a different energy and each group has sigma\_x = .002. This can be seen starting at line 231 and the particles are loaded on the beam at line 255.

-Edit\_Nick\_Valverde\_V9.py

Line 335 has a savefig command that will need to be changed. This is the initial potential plot.

-plotting.py

Line 32 is the import trajectoryfile.txt line. This will need to be changed so that it can find the trajectoryile. The trajectory file is written in to whatever directory Edit\_Nick… is found in.

Line 63, 92, 117, &140 are all savefig commands and can be changed to whatever directory you want them in. I created a folder Runs\_Plots that keeps all them.

\*\*There is an oddity when importing custom files that I found. When running these, import the custom files like Particle\_Class and even warp before importing numpy and matplotlib. For some reason, if you import the custom files afterward, it is as if you never imported matplotlib and you will get “has no attribute” errors.