ComponentGroups.py

Adam Theriault-Shay | adamt@mit.edu

Summer UROP 2016

1 Overview

The ComponentGroups.py library is designed to make running polarimetry simulations quick and simple. Long and high resolution simulations can be run with just four commands at the command line. Right now the library supports simulations that run from a point source to a multi-layer mirror, then to another multi-layer mirror, and finally to a CCD detector. The dimensions and spacing of all components can be customized via functions included in the library.
When running simulations, the library will also generate a file structure in which trial data is stored. The library also includes objects that generate relevant plots and histograms.
It has the following dependencies: matplotlib, numpy, astropy, transforms3d, marxs.

2 Simulations

2.1 staticSimulation()

This class combines two subclasses (called first and second stages) and allows them to be easily moved and manipulated. The first stage is composed of a point source and ML mirror, the second stage is a ML mirror and CCD. Within the class there are functions for easy maneuvering and wiggling of each component. Details in documentation.

2.2 rotation()

This class takes a staticSimulation object, rotates the first stage about any desired axis, and stores the data in a file system. It also generates a trialDetails.txt document that records the geometry of the simulation. One can decide the resolution of rotation (via number of angles tested) and resolution of each static simulation (via exposure time and flux). Details in documentation.

2.3 Examples (iPython)

2.3.1 1

In [1]: import componentGroups
In [2]: sim = componentGroups.staticSimulation()
In [3]: rot = componentGroups.rotation()
In [4]: stats = rot.run(sim, 7, 2000, intermediates = True, rotationAxis = [10,1,0])

This will automatically create the files for 7 angles.

2.3.2 2 Altering Setup

Altering Simulation Geometry:
In [1]: import componentGroups
In [2]: sim = componentGroups.staticSimulation()
In [3]: import numpy as np
In [4]: from transforms3d.euler import euler2mat
In [5]: from transforms3d.affines import compose
In [6]: T = [0,0,6] #Translation vector
In [7]: R = euler2mat(0,0,0,'syxz') # 3x3 rotation matrix
In [8]: MATRIX = compose(T,R,np.ones(3),np.zeros(3)) # make the 4x4 transformation matrix
In [9]: sim.first.move_mirror(MATRIX)
In [10]: print sim.first #This is just to display the geometry of the first stage. Not crucial for simulation.
[OUTPUT IS APPARATUS GEOMETRY]
In [11]: rot = componentGroups.rotation()
In [12]: stats = rot.run(sim, 19, 10000, intermediates = True)

The data files will then be generated and deposited into the file system, along with a trialDetails.txt file.

3 Data and Graphing

The graphs() class makes it easy to render visualizations of photon distributions over different angles and for each specific angle. It will automatically deposit these renderings into the file system. Details in documentation.
Example:
In [14]: graph = componentGroups.graphs(27)
In [15]: graph.CCD() # Generates final images of photons where opacity is directly related to probability.

In [16]: graph.hist() # Generates CCD images of probability distributions on the final CCD.

In [17]: graph.probabilities() # Generates graph of total probabilities as a function of rotation angle.

These graphing functions all also exist for the intermediate photons (see documentation)