Fragment Sputtering Comparison

Vectorial vs. Haser

Shawn Oset

Spring 2022

Problem Statement

TODO: write this Discrepancies between python vectorial code and fortran – why it is not a problem. Put a few images here that show the differences

Vectorial Model for Haser Enthusiasts

The vectorial model extends the Haser model primarily through its incorporation of fragment momentum changes via photodissociation. Instead of a fragment carrying along at the parent's velocity as in the Haser model, the fragments are ejected isotropically in the frame of the parent with a speed determined by the physics of the photodissociation.

Outflow Axis

Due to the underlying spherical symmetry of the Haser model and the vectorial model's isotropic ejection, we need only to compute how one stream of parents distributes its fragments around the coma.

(Picture of one outflow axis with multiple isotropic fragment sources along it)

This stream is referred to as an outflow axis, and the resulting distribution of fragments is called the fragment sputter. In the vectorial model, we use the z-axis for outflow, with an accompanying spherical coordinate system superimposed with the nucleus at its center.

Fragment Sputter

To compute this fragment sputter, we place a 2d grid around the coma. At each grid point (r_i, θ_j), we calculate how heavily each point along the outflow axis contributes fragments to (r_i, θ_j). This amounts to a numerical integration along the outflow axis for each grid point.

(Picture with one grid point singled out and multiple axis points contributing to it) (Picture should maybe show density of points being higher near the closest points)

The fragments that travel the shortest distance will dominate the contributions to (r_i, θ_j) due to the exponential decay from photodissociation of fragments. There are two factors that influence how well the model can capture these dominating contributions:

- We must sample the outflow axis around the minimum fragment travel distance more heavily
- (Picture of finer sampling near minimum travel distance)
- The angular grid must be fine enough to encroach on the outflow axis and pick up fragments before they have been destroyed by photodissociation
- (Picture of two different angular grids)
- The radial grid must be fine enough to capture the heavy contributions that occur from parents dissociating near the nucleus

Unfortunately the angular grid size is fixed at 26 subdivisions, which keeps the grid farther away from the TODO: If we use same angular grid as fortran, does our new sampling make the results diverge from fortran?

For short fragment lifetimes, the fortran clamps the contributions at 8 lifetimes, which means the python will be consistently higher at larger distances

Separate into two regimes: near the nucleus, and near the edge of the grid (or past eight lifetimes)

Fortran Fragment Sputter

In function SDENT, the variable TLIMI is set to 8 times the fragment lifetime, which can artificially limit the fragment sputter, especially when dealing with large Q. The following shows the fragment sputter table for a short and long fragment lifetimes at the edge of the grid:

Fortran Column Density

In function SLINS, the following are checked before doing anything:

- If ρ is larger than the coma it returns zero, by comparing to X(JMAX), the outermost point in the grid
- If ρ is closer to the nucleus than the first radial grid point, returns zero by comparing to X(1), the first radial grid point

For the Gaussian integration, the line of sight integration is computed from z = 0 to z = SSUP, where SSUP is limited to a maximum of the edge of the grid, X(JMAX).

Python Fragment Sputter

Look at the fragment sputter table for equivalent runs