Fragment Sputtering Comparison

Vectorial vs. Haser

Shawn Oset

Spring 2022

Fortran and Python Model Testing

The python vectorial model included in sbpy is based on Festou 1981, but uses python tools and libraries, so some variation in results might be expected purely on the grounds of implementation detail. Major disagreements between the models can happen for certain sets of parameters, particularly when the parent or fragment lifetimes are small. These disagreements generally occur near the nucleus and near the edge of the radial grid, and this document will explain the origin of these discrepancies as shortcomings in the fortran version.

Vectorial Model Geometry

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 after photodissociation 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

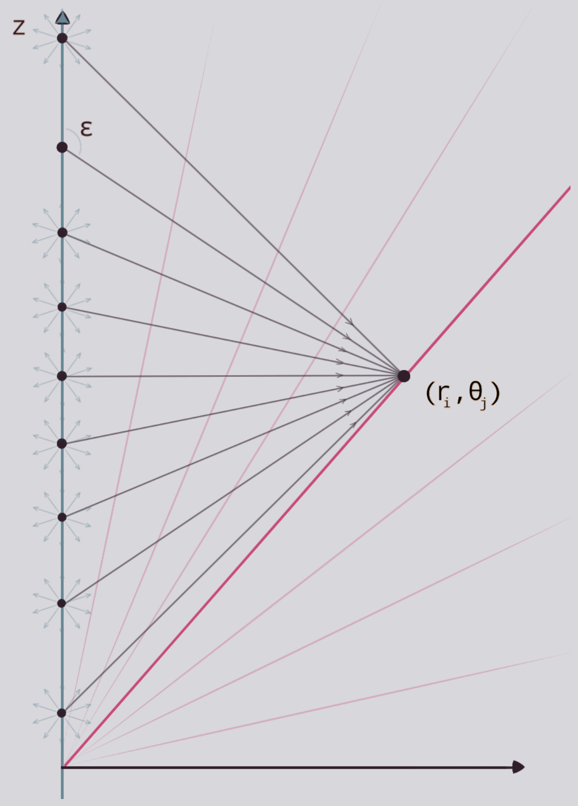


Figure 1: Fragments traveling from the outflow axis to another point (r_i, θ_j) in the coma.

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 outflowing parents distributes its fragments around the coma.

This stream of parent molecules is referred to as an **outflow axis**, and the resulting distribution of fragments is called the **fragment sputter**.

Figure 1 shows the outflow in blue on the z-axis, with the isotropic production of fragments due to photodissociation of the parents along their outflow. The nucleus of the comet is taken to be the origin, with the spatial grid around it in red. The fragment density at (r_i, θ_j) is then the sum of all points of the outflow axis that contribute by ejecting a fragment at angle ϵ.

Fragment Sputter

To compute this fragment sputter, we place a 2d grid around the coma, with a **radial grid** $\{r_i\}$ and an **angular grid** $\{\theta_j\}$. 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 an integration along the outflow axis for each grid point (r_i, θ_j).

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

- For each (r_i, θ_j), we must sample the outflow axis around the minimum fragment travel distance more heavily for the best numerical integration results.
- The angular grid must be fine enough to encroach on the outflow axis and pick up fragments before they have been destroyed by photodissociation. The fortran model will only allow the fragments to travel for **8** lifetimes, so if the grid does not encroach close enough, it will miss fragments, especially at larger radii where the distance between any grid point and the outflow axis (z-axis) is large.
- The radial grid must be constructed to capture the heavy contributions that occur from parents dissociating near the nucleus.

The fragment sputter from an outflow axis is the fundamental calculation that the model uses to build the radial fragment density $n(r_i)$ and calculate the resulting column density. Any over- or under-estimations in the fragment sputter will show in the results, so we investigate the fragment sputter of the two models as a source of discrepancy.

Spatial Gridding: Fortran

The fortran version has a hard-coded grid of $N_r =$**150** radial grid points, distributed quasi-logarithmically with the point separation pre-determined by another array. The spatial scale of the grid is based on the lifetimes of parent and fragment, and this pre-determination can affect the model negatively. Namely, small lifetime values can cause the innermost radial grid points to lie well within the **collision sphere (R_{coll})**, the region around the nucleus where the vectorial model is invalid.

The angular grid is also fixed with $N_\theta =$**26**, distributed evenly between $\theta = 0$ and $\theta = \pi$, limiting the ability of the model to encroach on the outflow axis at small θ and catch fragments before photodissociation.

Spatial Gridding: Python

The construction of a true logarithmic radial space, with N_r configurable, is trivial with libraries like **numpy**, so the python version can more densely sample the space near the nucleus where the fragment sputtering activity is at its highest. Additionally, the radial grid is always started at **2 * R_{coll}**, or twice the collision sphere radius to avoid areas where the model is invalid.

The number of angular grid points N_θ are again configurable, allowing a finer sampling of the space near the outflow axis.

Column Density Calculation

The integration needed to calculate the column density needs values of $n(r)$ where r does not fall on the **radial grid** and so we must use an interpolation scheme to produce $n(r)$ at arbitrary values of r. Both versions use a cubic spline for approximation between grid points, but the domain is limited in the fortran version to $R_{min} < r < R_{max}$.

To extend the approximation of $n(r)$ to all radii, the python version will set $n(r) = n(r_0)$ for $r < R_{coll}$, capping the number density at the value of the grid point closest to the nucleus. Typical values of R_{coll} are $R_{coll} \approxeq 80km$, while typical grid sizes extend to $\approxeq 1,000,000$ km, so the contribution to the column density from this assumed $n(r < R_{coll})$ is small due to the low volume encompassed inside the collision sphere.

No approximation of $n(r)$ in the fortran model is made beyond the edge of the grid, with a simple assumption that $n(r > R_{max}) = 0$. To avoid this sharp cutoff in $n(r)$ at the edge of the grid, the python model uses the total lifetime of the fragment to apply an exponential decay of the outermost grid point.

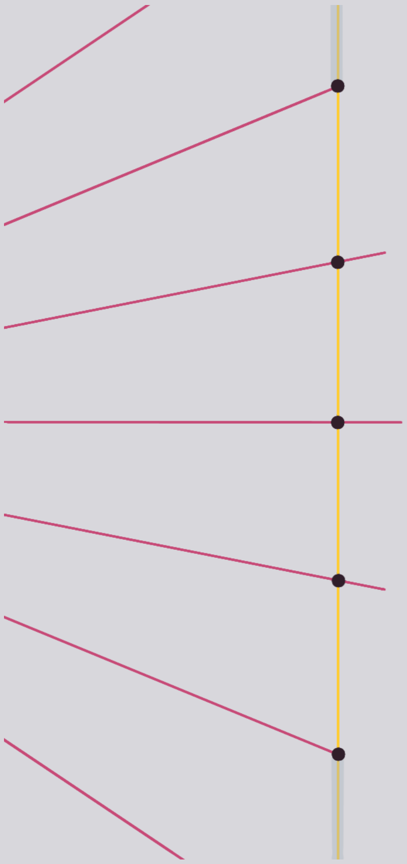


Figure 2: Limited information for column density calculations near the edge of the grid

Figure 2 shows radial grid lines (red) with a column density line of sight path in yellow near the edge of the grid, with shade denoting regions where $r > R_{max}$. The fortran version takes the fragment density $n(r)$ to be zero in the shaded region of the line of sight, while the python version will approximate in this region.

For column densities near the edge of the grid, this exponential fragment decay approximation can contribute to the python column densities being higher than fortran's.

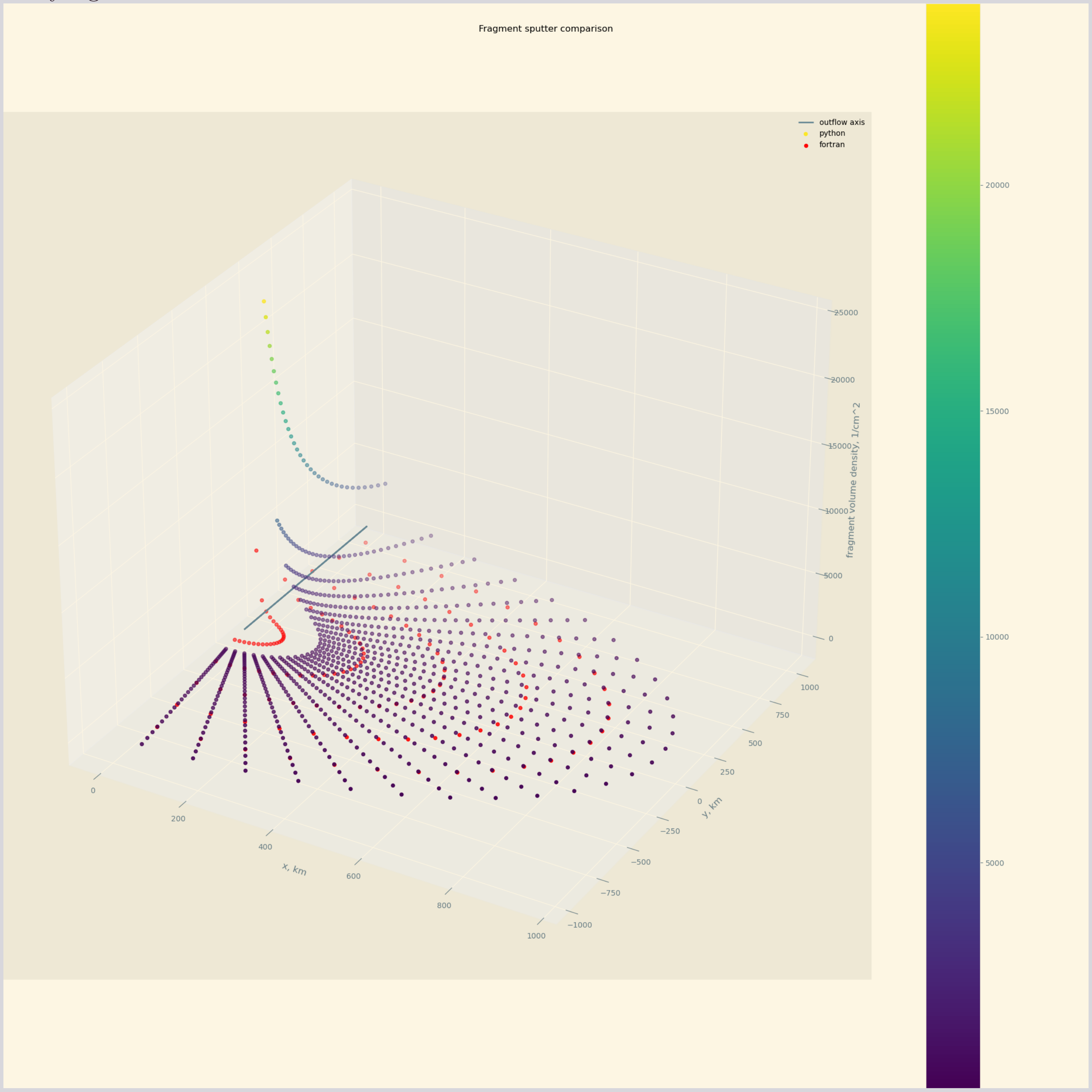
In the cases mentioned earlier that can cause the fortran model to set radial grid points inside the collision sphere, the near-nucleus column densities will be overestimated due to the very large (but invalid) values of $n(r)$ inside the collision sphere.

The following figures are produced by giving both models the same input with a range of parameters to show the effects of fragment lifetime (and therefore grid size). The runs were performed with a python grid that roughly resembled the grid of the fortran model ('matching grids') to isolate the effects of fragment lifetime, and once again with a higher-detail python grid to show that the models are converging on the same results with entirely different grids.

Effect of Matching Grids, Fragment Lifetime 20000 s

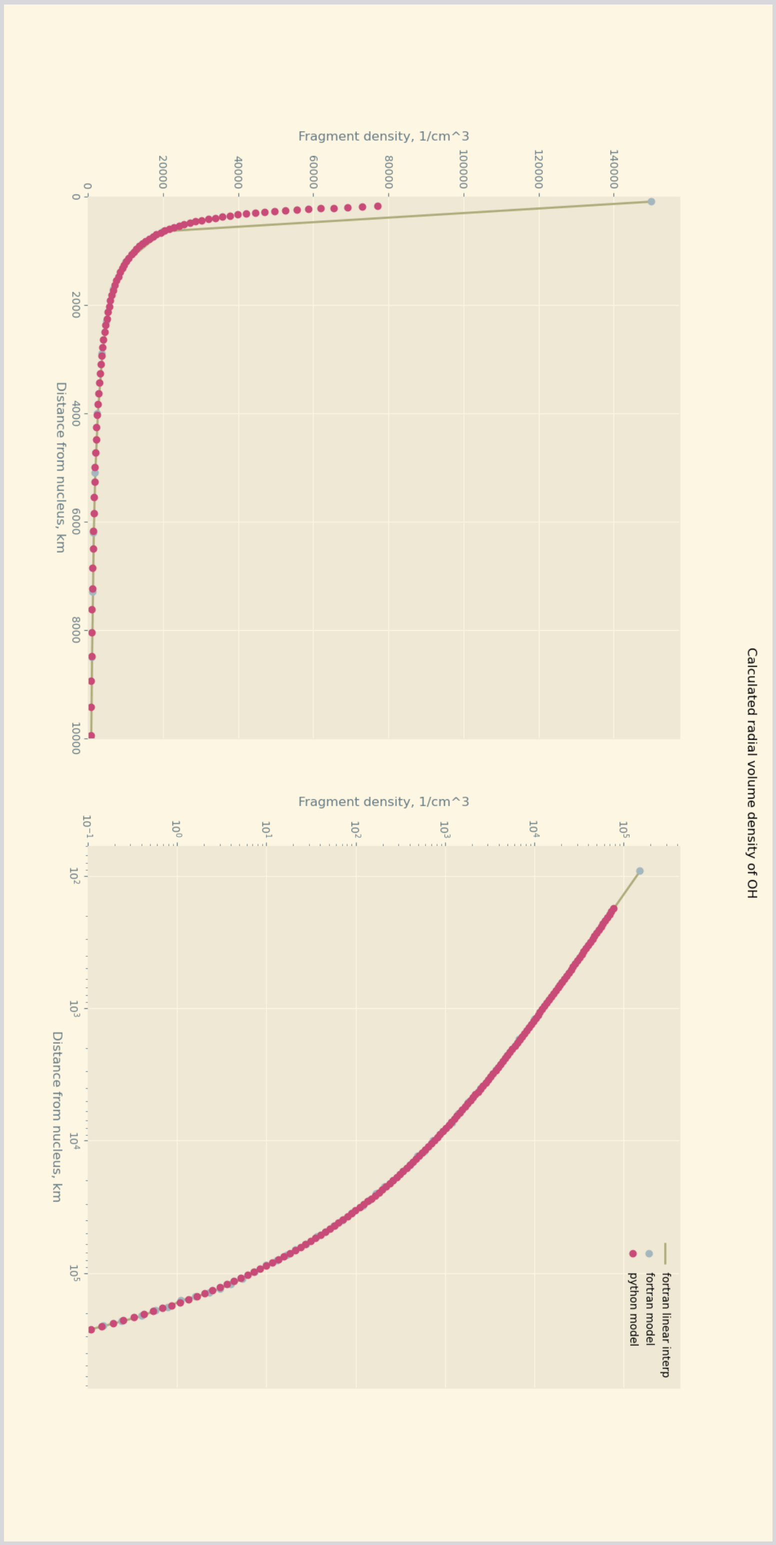
Fragment Sputter

The fragment sputter for both models, with fortran grid in red. Note the first fortran grid point is much closer to the nucleus, while the python grid is much more dense while staying outside the collision sphere (not marked). It may appear as though the python version should be the overestimator here, but this fragment sputter is integrated to produce $n(r)$, which makes the fragment sputter difficult to reason about by sight alone.



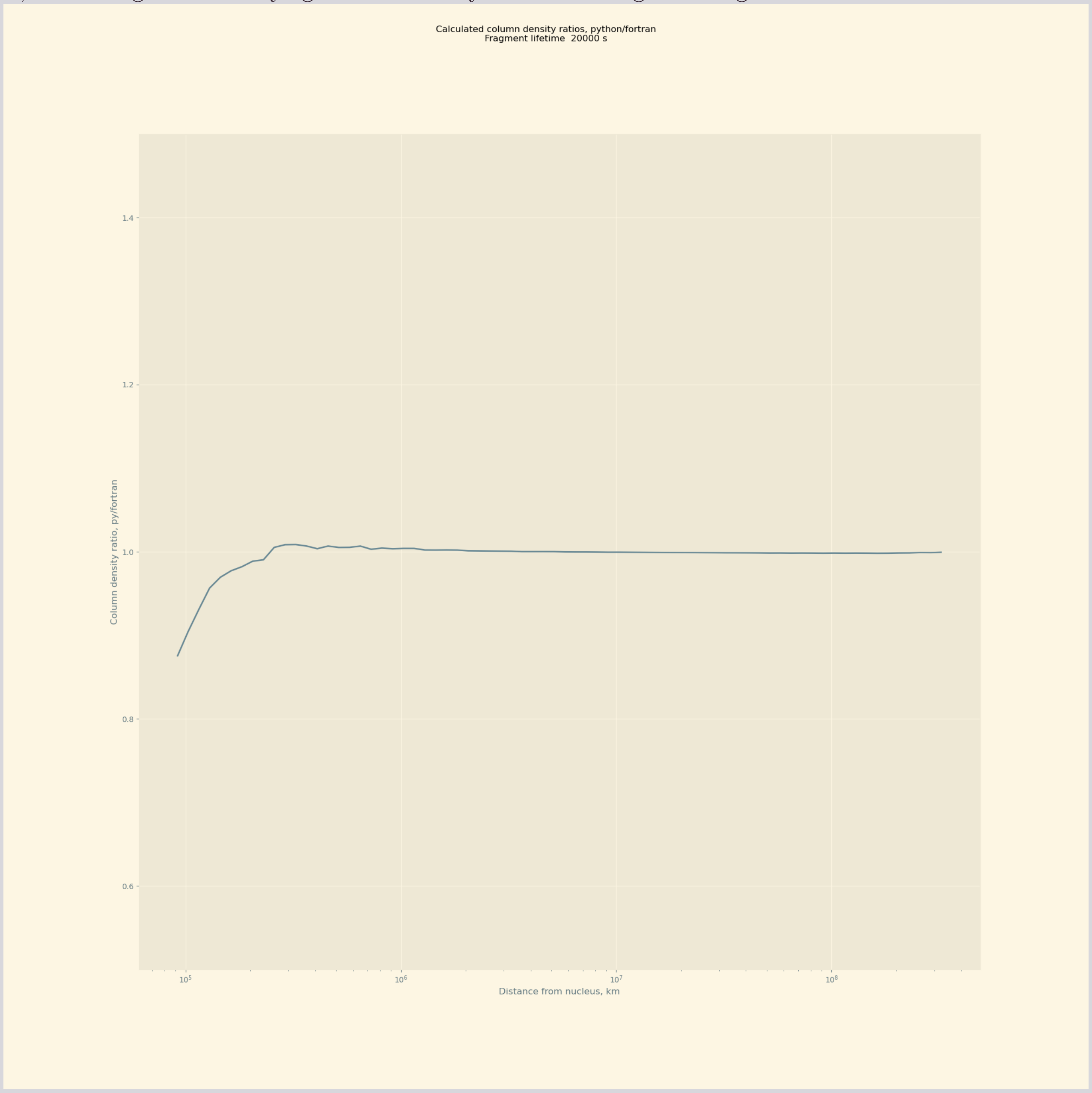
Volume Density

The calculated volume density $n(r)$, showing the overestimation coming from inside the collision sphere.



Column Density

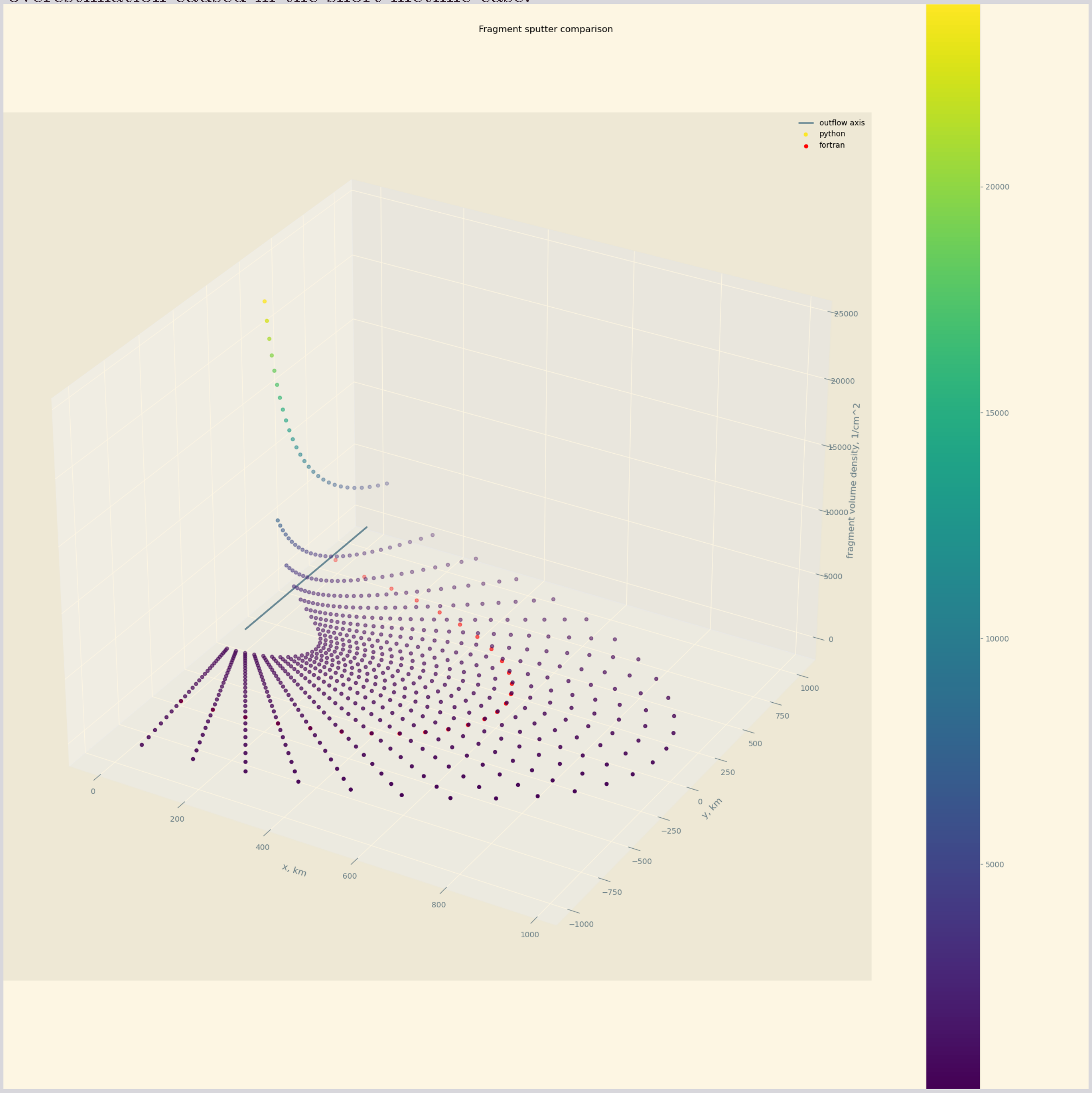
Column density, showing the fortran overestimation near the nucleus. The short fragment lifetime keeps the python version's extra contributions from the approximation beyond the edge of the grid to a minimum, so that agreement stays good all the way out to the edge of the grid.



Effect of Matching Grids, Fragment Lifetime 500000 s

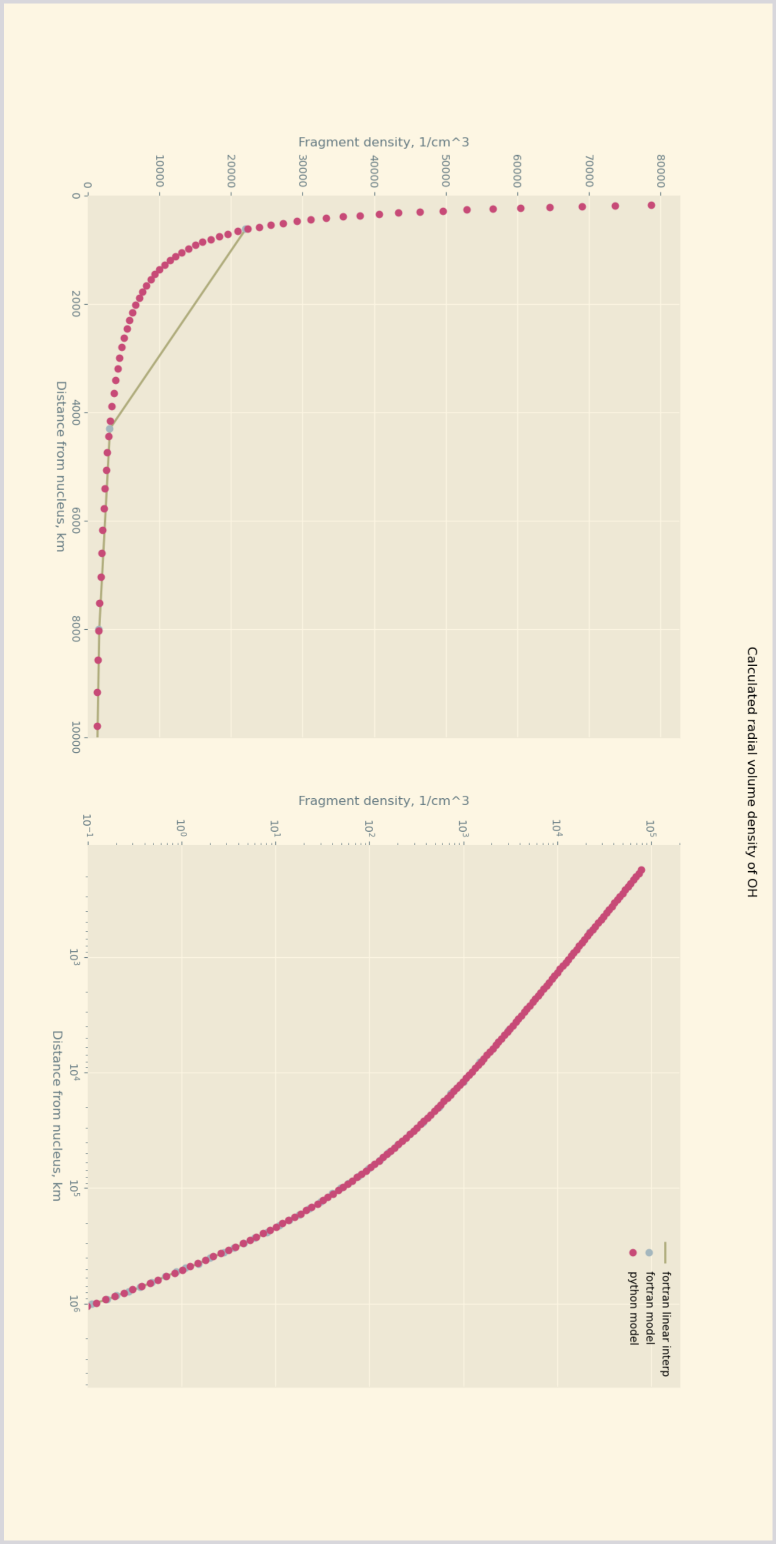
Fragment Sputter

The longer fragment lifetime pulls the inner edge of the fortran grid away from the nucleus, eliminating the overestimation caused in the short lifetime case.



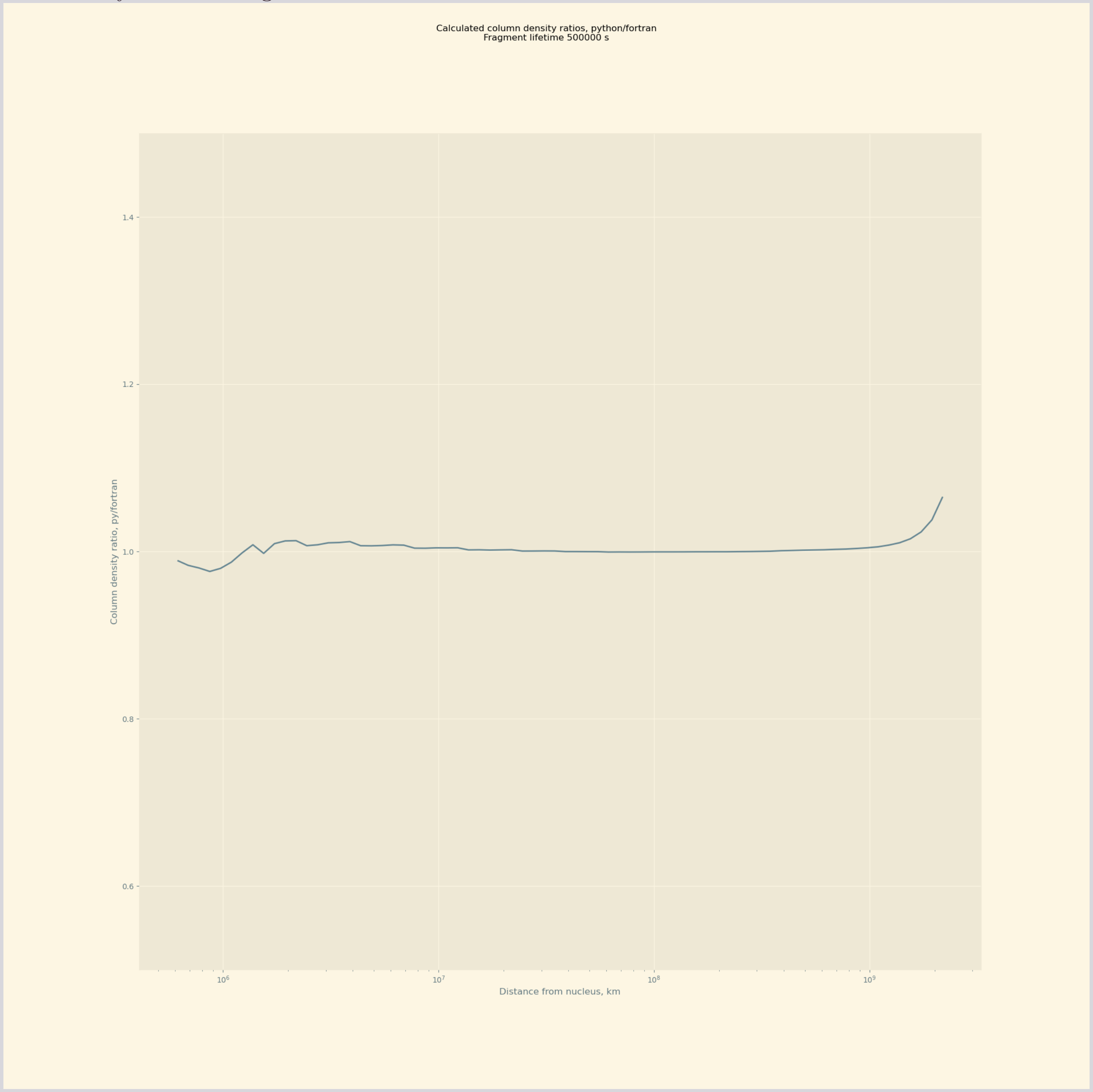
Volume Density

While the fortran points are more sparse near the nucleus, the values line up well with python's, with disagreement coming from the implementation detail of fortran's fixed grid.



Column Density

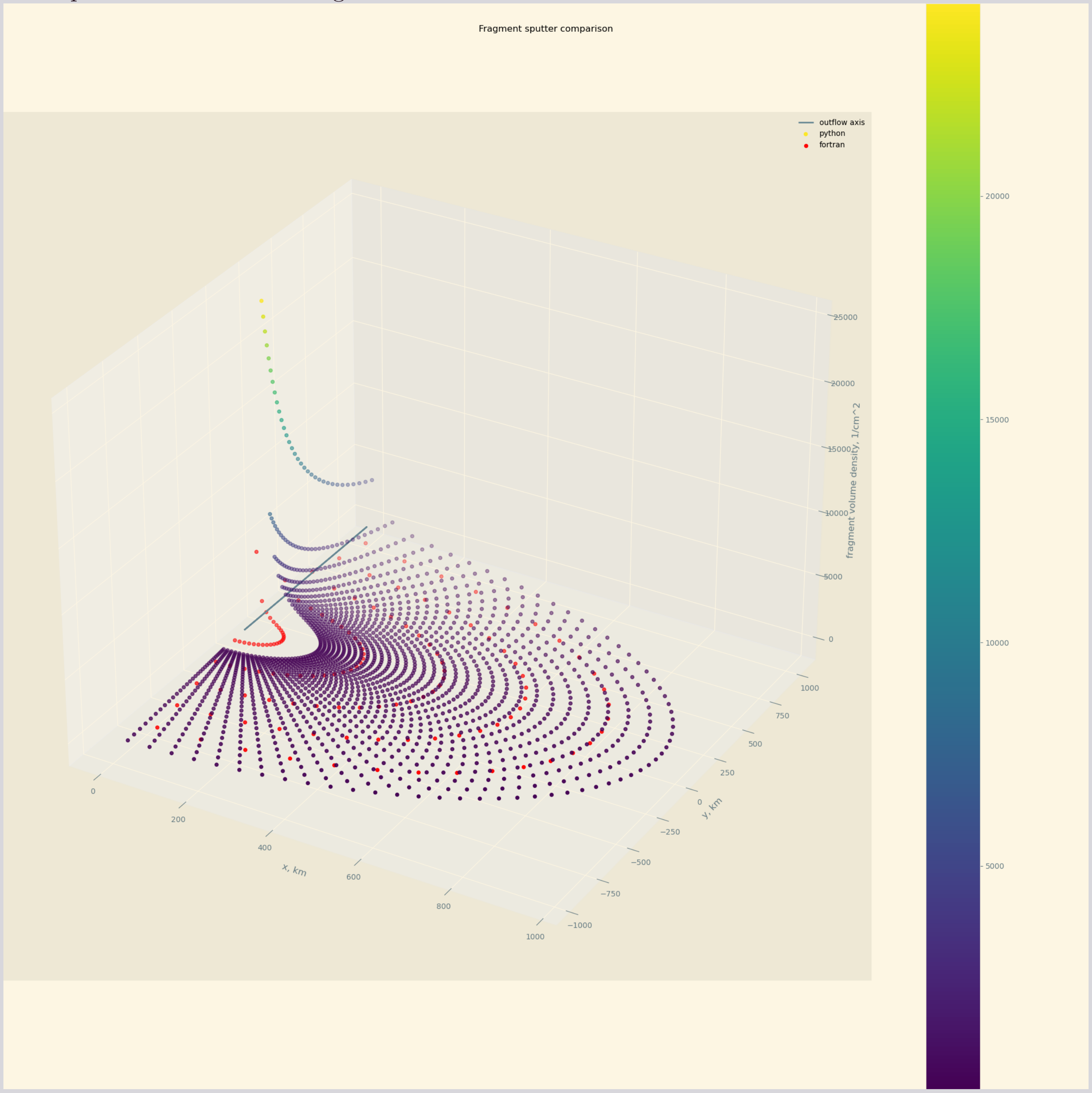
Here we see much better agreement near the nucleus with some variation due to the very different radial grid spacing, which explains the erratic over/under estimation. Further, the outer edge is seeing contributions from python approximating beyond the edge of the grid, which leads to a slightly higher column density for it at large radii.



Effect of HQ Python Grid, Fragment Lifetime 20000 s

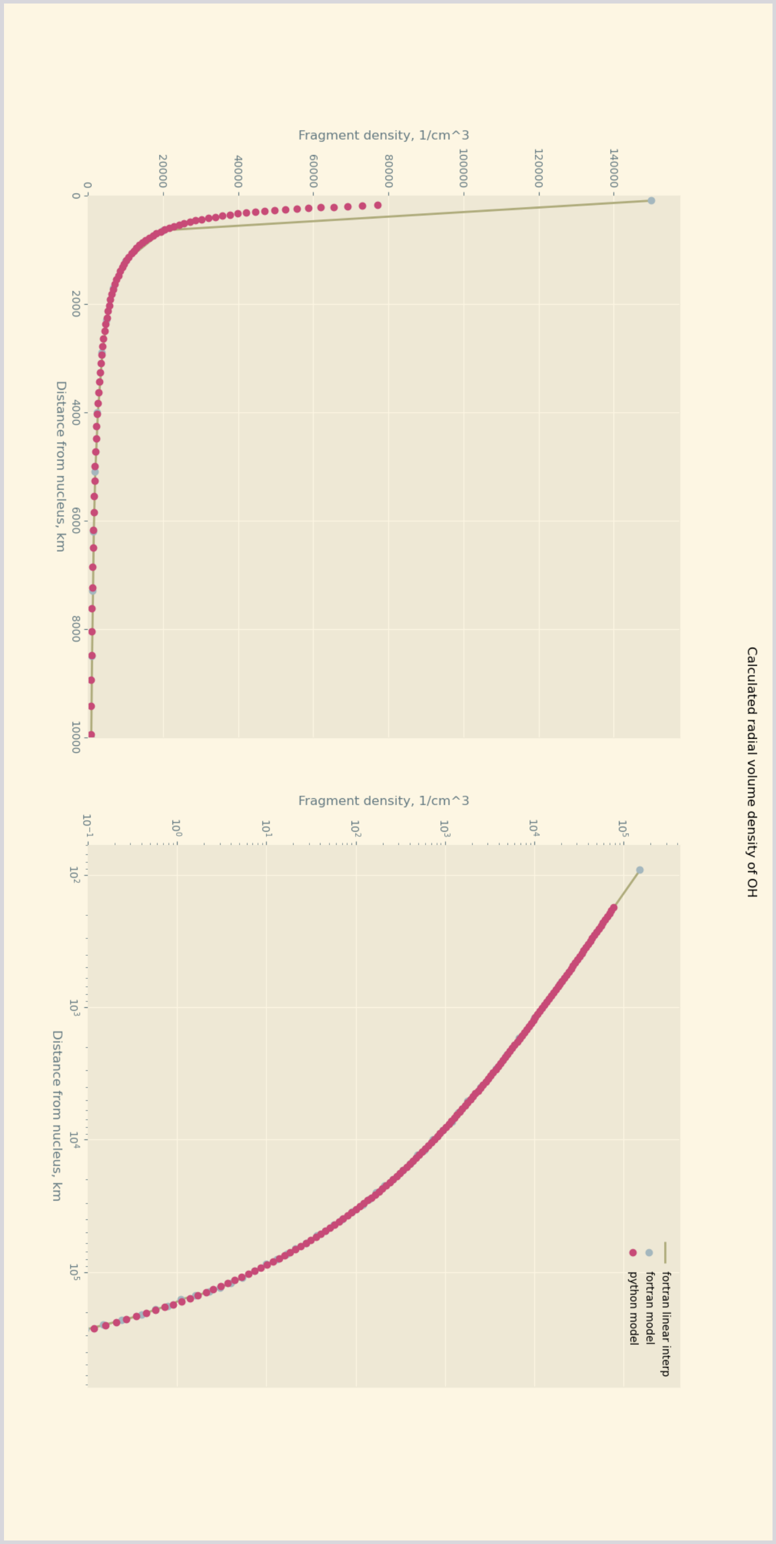
Fragment Sputter

The short fragment lifetime underscores the importance of sampling the outflow axis properly: note the outer edge of the grid near the outflow axis is consistently higher in the python model as it 'catches' more fragments from the outflow axis. This manifests in additional column density near the edge of the grid. We expect disagreement near the nucleus as we can see the fortran model once again peeking inside the collision sphere and overestimating as a result.



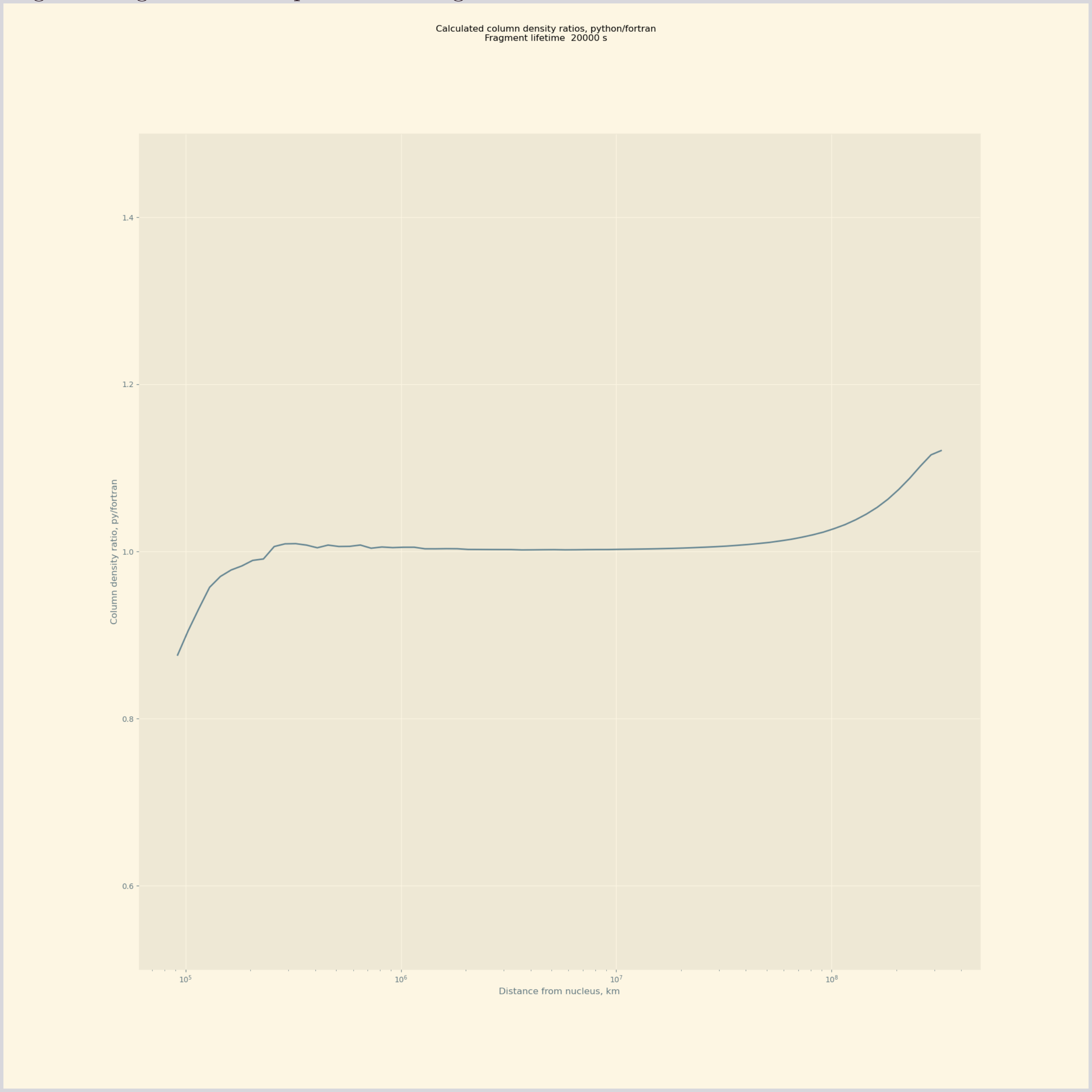
Volume Density

Even with an unmatching grid, the models generally agree outside of the sources of discrepancy already discussed.



Column Density

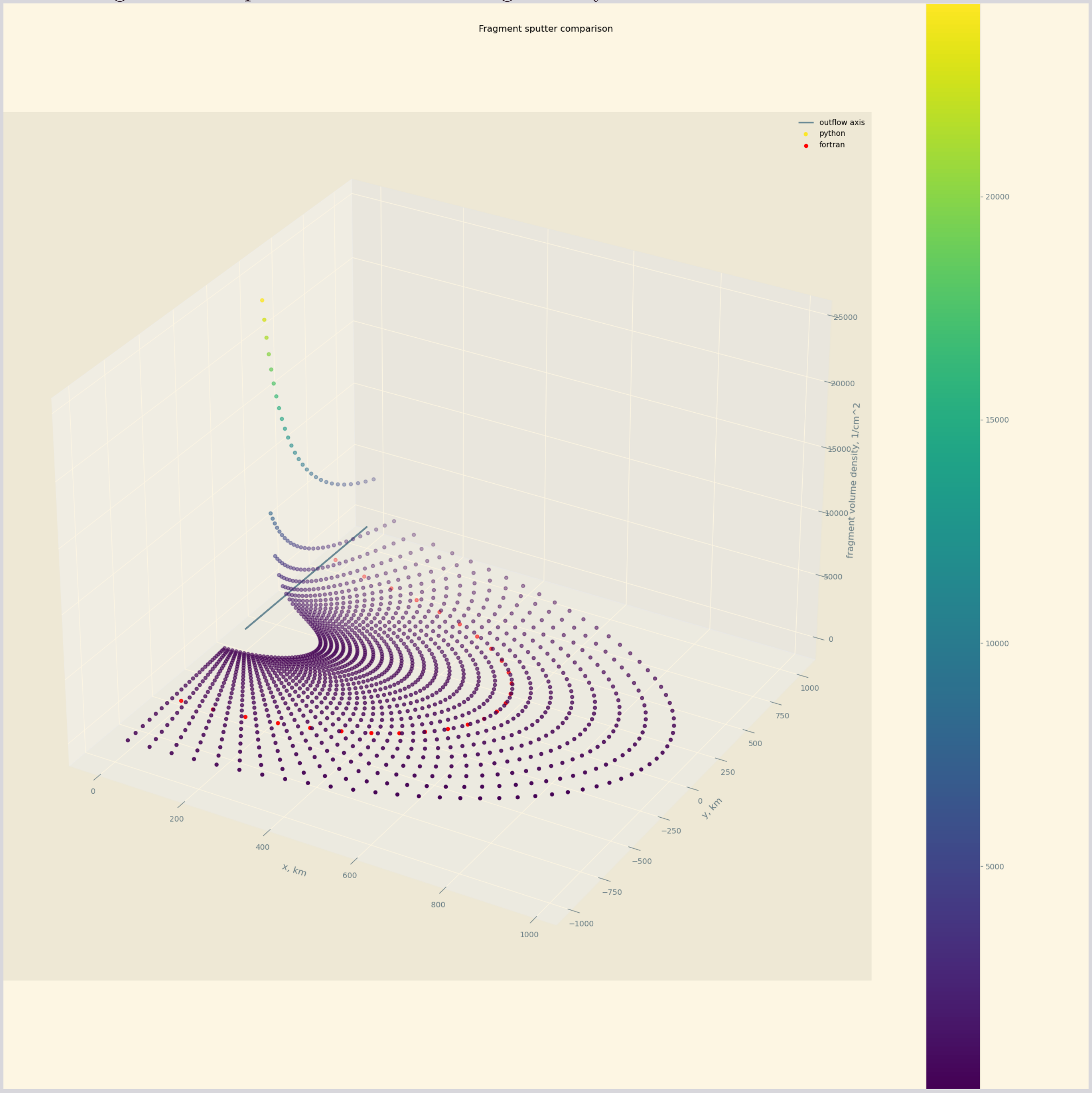
We see the effects of near-nucleus overestimation and the HQ grid picking up additional fragments near the edge of the grid that the sparser fortran grid does not.



Effect of HQ Python Grid, Fragment Lifetime 500000 s

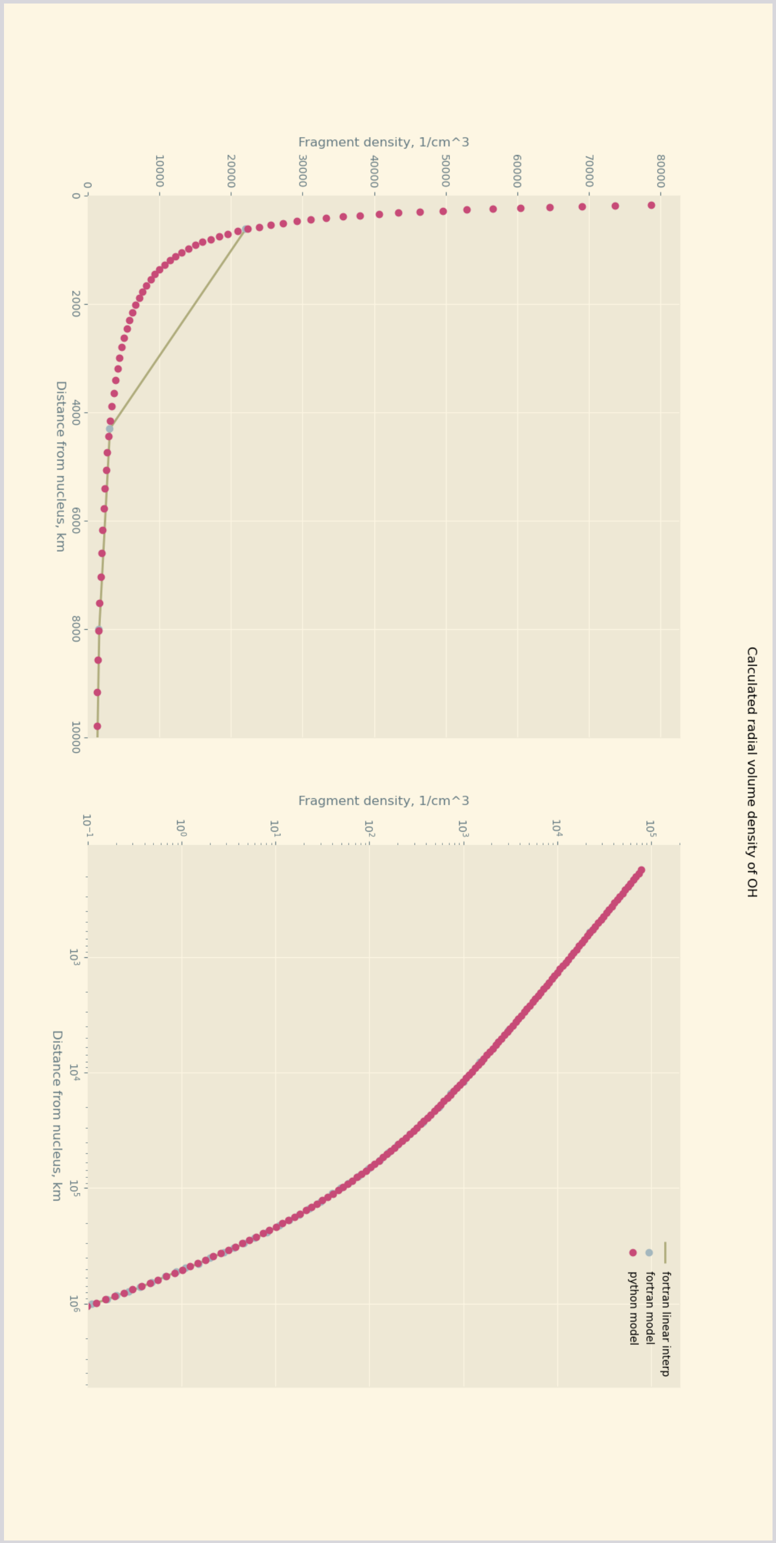
Fragment Sputter

Again the longer lifetime pulls the fortran radial grid away from the nucleus.



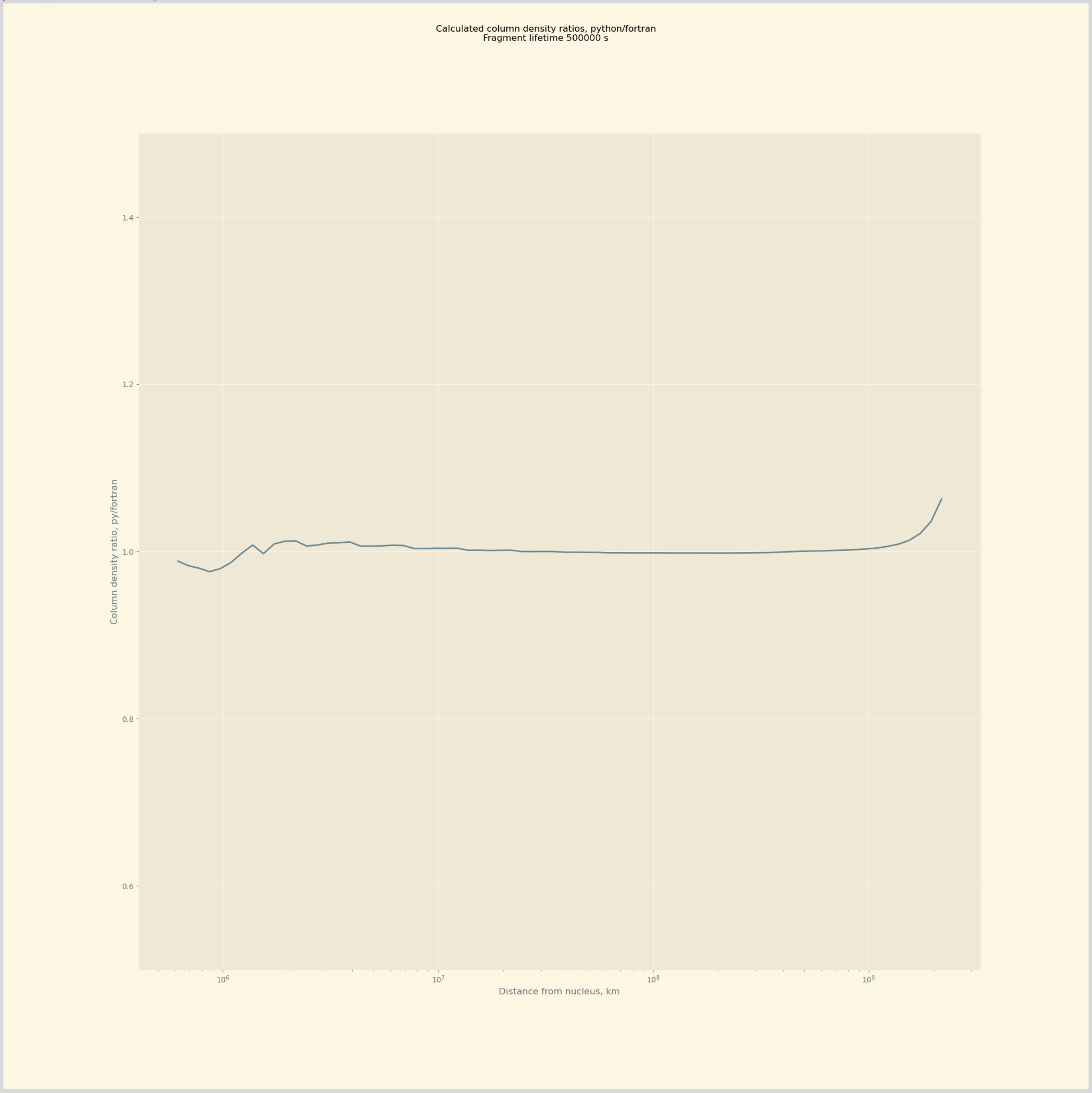
Volume Density

Good agreement in general despite the different gridding.



Column Density

The same effects as the large fragment lifetime on the matching grids: variance in the agreement near the nucleus, with a slight extra contribution near the edge of the grid from python's attempt to estimate $n(r)$ beyond the grid.



Summary

Despite the two models having a different codebase and mildly different approaches, they in general agree, so there is high confidence that the python version is calculating correctly. When they do not agree, the above comparisons and explanations should show that the deviations are from limitations in the fortran model.

Near the Nucleus

The failure of the fortran model to stay outside the collision sphere when fragment lifetimes are short lead to overestimations of $\approx 10 - 20\%$ when calculating column density near the nucleus ($r < 500,000$ km).

Edge of the Grid

Two contributions can cause python to have higher column densities near the edge of the grid.

By approximating beyond the edge of grid where fortran would say there are zero fragments, python will pick up more fragments ($\approx 5\%$). This effect is most noticable near the edge of the grid due to the calculation of the column density as shown in Figure 2.

The second contribution comes when python's angular grid is finer than fortran's, which will more accurately characterize the fragment sputter at large radii. This effect is exaggerated at low fragment lifetimes due to the exponential decay from photodissociation – the sparser fortran angular grid is too far away at large radii to catch fragments to the same precision, leading to a higher column density for the python version.