AGB Star Visualisation - Report

Annalisa Calvi and Jacob Zhang

July 2025

1 Introduction

Our package, `agb-deprojection-methods` (see `https://pypi.org/project/agb-star-deprojection-methods/`), provides methods to help visualise the circumstellar envelopes of AGB stars. It works under the assumption that these circumstellar envelopes are expanding radially.

2 Computing x and y offsets

For the plot, the x, y coordinates have to be centered on the star. The original data provides coordinates in terms of right ascension and declination. To convert these into distances, we use the small-angle approximation $\tan(\theta) \approx \theta$ to get the x and y offsets:

$$\Delta x = \theta D$$
$$\Delta y = \alpha D$$

where D is the distance from the earth to the star and θ, α are the right ascension and declination in radians.

3 Computing v_{sys} and v_{exp}

3.1 Theory

We begin by assuming that the circumstellar envelope forms a sphere of radius R, and that matter on the surface of this sphere is travelling at v_{exp} km/s. Then points (x, y, z) on the surface (in physical space) satisfy $x^2 + y^2 + z^2 = R^2$, and map to $(x, y, v_{\text{exp}}z/R)$ in velocity space. That is, the channel velocity v is equal to $v_{\text{sys}} + v_{\text{exp}}z/R$, so that $z = R(v - v_{\text{sys}})/v_{\text{exp}}$. We then see that $x^2 + y^2 + R^2(v - v_{\text{sys}})^2/v_{\infty}^2 = R^2$. This equation describes an ellipse in (x, y, v).

A cross-section of this ellipse with constant v is a circle with radius $\sqrt{x^2+y^2}$, noting that although x, y can change, x^2+y^2 must be constant. The area, A, of the cross-sectional circle is then given by $\pi(x^2+y^2)$. It follows that $A/\pi + R^2(v-v_{\text{sys}})^2/v_{\text{exp}}^2 = R^2$, and finally,

$$A(v) = \pi R^2 \left(1 - \frac{(v-v_{\text{sys}})^2}{v_{\text{exp}}^2}\right).$$

For $v \notin [v_{\text{sys}} - v_{\text{exp}}, v_{\text{sys}} + v_{\text{exp}}]$ we say $A(v) = 0$. Note that $A(v)$ is defined on a continuous range of values.

Suppose we have, for each discrete velocity channel v, an empirical value $d(v)$ which is proportional to the theoretical value $A(v)$. Then we can estimate v_{sys} and v_{exp} as follows. First, we compute the normalised values, $\hat{d}(v) := \frac{d(v)}{\sum_u d(u)}$. We also normalise $A(v)$, calculating

$$\hat{A}(v) := \frac{A(v)}{\int_{v_{\text{sys}}-v_{\text{exp}}}^{v_{\text{sys}}+v_{\text{exp}}} A(u)\,du} = \frac{3}{4v_{\text{exp}}}\left(1 - \frac{(v-v_{\text{sys}})^2}{v_{\text{exp}}^2}\right).$$

I will point out that $\hat{d}(v)$ is not necessarily equal (or even close) to $\hat{A}(v)$, though they will still be in proportion. This is because $\sum_u d(u)$ is influenced by how many velocity channels there are.

Notice, however, that $\hat{d}$ satisfies the definition of a probability mass function (PMF), and $\hat{A}$ satisfies the definition of a probability density function (PDF). Let X be the discrete random variable with PMF $\hat{d}$, and let Y be the continuous random variable with PDF $\hat{A}$. Recalling that our discrete velocity channels are evenly spaced, we see that X is a discrete approximation of Y. In particular, it is reasonable to set $\mathbb{E}[X] = \mathbb{E}[Y]$ and $\text{Var}(X) = \text{Var}(Y)$, and solve for $v_{\text{sys}}, v_{\text{exp}}$ satisfying these equations. We compute

$$\mathbb{E}[Y] = \int_{v_{\text{sys}}-v_{\text{exp}}}^{v_{\text{sys}}+v_{\text{exp}}} u\hat{A}(u)\,du = v_{\text{sys}}$$

$$\text{Var}(Y) = \int_{v_{\text{sys}}-v_{\text{exp}}}^{v_{\text{sys}}+v_{\text{exp}}} (u-v_{\text{sys}})^2\hat{A}(u)\,du = \frac{1}{5}v_{\text{exp}}^2,$$

so that we can approximate v_{sys} and v_{exp} as

$$v_{\text{sys}} := \mathbb{E}[X] = \sum_v v\hat{d}(v) \tag{1}$$

$$v_{\text{exp}} := \sqrt{5\text{Var(X)}} = \sqrt{5\sum_v (v-v_{\text{sys}})^2\hat{d}(v)}. \tag{2}$$

3.2 Computing relevant empirical values

We tried a few ways of calculating $d(v)$ from the data, and ultimately landed on this method: for each velocity channel v, let $d(v)$ be the *average intensity* of the points *within two beams* of the radius. We make the assumption that this is proportional to the cross-sectional area of the amount of gas around the star in this velocity channel.

3.3 Dealing with noise

The above method of computing $d(v)$ does result in some noise outside the bounds of $[v_{\text{sys}} - v_{\text{exp}}, v_{\text{sys}} + v_{\text{exp}}]$. We can tackle this with the following iterative process:

1. Normalise $d(v)$.
2. Estimate v_{sys}, v_{exp} using (1) and (2).
3. For $v \notin [v_{\text{sys}} - v_{\text{exp}}, v_{\text{sys}} + v_{\text{exp}}]$, set $d(v) = 0$.

Additional noise results in overestimating v_{exp}, as it increases the variance of X. So in the above process, v_{sys} stays relatively stable, and v_{exp} monotonically decreases until convergence. We accept the values of v_{sys} and v_{exp} once we see that they are repeating.

The convergence of this algorithm can be visualised using the method `plot-velocity-vs-intensity`.

4 Computing beta law parameters

The main issue here is computing the radial expansion velocity at each radius. We assume that there is a one-to-one correspondence between:

- the intensity of a data point and its distance from the star (radius); and
- the radius of a data point and its radial velocity.

We then take these steps:

1. At each channel velocity v, let $d(v)$ be the average intensity of the points within one beam of the centre.
2. When calculating v_{sys} and v_{exp}, we assume $d(v) \propto A(v)$. That is, $d(v)$ should approximate a parabola, with zeros at $v_{\text{sys}} - v_{\text{exp}}$ and $v_{\text{sys}} + v_{\text{exp}}$. Let $I(v)$ be the parabola with these zeros, and maximum value set to $\max d(v)$. In particular,
$$I(v) := (\max d(v)) \left(1 - \frac{(v - v_{\text{sys}})^2}{v_{\text{exp}}^2}\right)$$
. So I calculates *intensity* from *channel velocity*.
3. As we are taking the intensity of points in the centre of each frame, their radial expansion velocity u is given by $u = |v|$. We can write $I(u) = (\max d(v)) \left(1 - \frac{(u - v_{\text{sys}})^2}{v_{\text{exp}}^2}\right)$, defined for $u \in [0, v_{\text{exp}}]$.
4. We see that $I(u)$ is a one-to-one correspondence between the intensity of a data point and its radial velocity. It has inverse
$$u(I) = v_{\text{exp}} \sqrt{1 - I/(\max d(v))} + v_{\text{sys}},$$
defined for $I \in [0, \max d(v)]$.

5. We then take the central velocity channel (with velocity closest to the systemic velocity). For each point (x, y) in the central channel, its distance from the star is given by $\sqrt{x^2 + y^2}$. Now, for each radius r, we can compute the intensity $I(r)$ by taking the average intensity over all points (x, y) with $\sqrt{x^2 + y^2} \approx r$.

6. Finally, we can compute radial velocity from the radius by taking $u(r) := u(I(r))$.

Once we have a radial velocity for each radius, we can fit the beta velocity law

$$u_\beta(r) = v_0 + (v_{\text{exp}} - v_0)\left(1 - \frac{r_{dust}}{r}\right)^\beta.$$

In particular, we fit β and r_{dust}, using `scipy.optimise.curve_fit`. and assume $v_0 = 3$ km/s (the speed of sound).

5 Computing radius

In many cases it was convenient to have an approximate radius of the gas cloud. The final method used to calculate a star's radius was to first select the velocity channel corresponding to the systemic velocity of the star. Then, define the following function

$$f(r) = \#\left\{(x,y)|\, x^2 + y^2 < r^2, \text{data} > \mu + 5\sigma\right\} + \#\left\{(x,y)\,|x^2 + y^2 > r^2, \text{data} < \mu + 5\sigma\right\}$$

and choose the radius to be the value of r which maximises this function. The rationale behind this decision is that the correct radius choice would be one that maximises both the amount of significant datapoints within the radius as well as the number of non-significant datapoints outside of the radius.

6 Beam Filter

When reading the beam data, we are given the length of that major axis, a, the length of the minor axis, b, and the principal angle, θ. Let

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$
$$D = \begin{bmatrix} b & 0 \\ 0 & a \end{bmatrix}$$

Let $B := R_\theta D^{-2} R_\theta^T$. Then, for a vector v in right ascension, declination coordinates, v is $v^T B v$ beams away from the centre.

7 Deprojection

Let $u(r)$ be a function mapping the radius of the gas cloud to its radial velocity. Let P denote physical space, V denote velocity space, and $T : P \to V$ denote the projection map. Then P has coordinates (x, y, z), V has coordinates (x, y, u), and we write

$$T(x, y, z) = \left(x, y, \frac{z}{\sqrt{x^2 + y^2 + z^2}} u(\sqrt{x^2 + y^2 + z^2}) \right).$$

7.1 Constant velocity deprojection

For a constant velocity, we have $u(r) = v_{\text{exp}}$, so that

$$T(x, y, z) = \left(x, y, \frac{z}{\sqrt{x^2 + y^2 + z^2}} v_{\text{exp}} \right).$$

In particular, the channel velocity v is given by $v = \frac{z}{\sqrt{x^2+y^2+z^2}} v_{\text{exp}}$. It follows that

$$(v_{\text{exp}}/v)^2 = \frac{x^2 + y^2 + z^2}{z^2},$$

and, as sign(z) =sign(v), by algebraic manipulation we obtain

$$z = v \frac{\sqrt{x^2 + y^2}}{\sqrt{v_{exp}^2 - v^2}}.$$

7.2 Solving for the radius

We now consider a general velocity law $u(r)$. Let $r = \sqrt{x^2 + y^2 + z^2}$. Then

$$(x, y, v) = \left(x, y, \frac{z}{r} u(r) \right).$$

As $z = vr/u(r)$, we have

$$r^2 = x^2 + y^2 + z^2 = x^2 + y^2 + (vr/u(r))^2,$$

so to find r, we need to solve the equation

$$r^2 \left(1 - \frac{v^2}{u(r)^2} \right) - (x^2 + y^2) = 0$$

for some $r > \sqrt{x^2 + y^2}$. Fortunately, the left hand side is monotonic: we have

$$\frac{d}{dr} \left(r^2 \left(1 - \frac{v^2}{u(r)^2} \right) - (x^2 + y^2) \right) = 2r \left(1 - \frac{v^2}{u(r)^2} \right) + 2r^2 \frac{v^2 u'(r)}{u(r)^3} > 0. \tag{3}$$

As it is monotonic, we can use `scipy.optimise.elementwise.find_root` to solve for r.

7.3 Arbitrary velocity law deprojection

To deproject using an arbitrary velocity law $u(r)$, we first solve for r as described above, then set $z = \text{sign}(v)\sqrt{r^2 - x^2 - y^2}$.

It is also possible to find z directly, by solving

$$\frac{z}{\sqrt{x^2+y^2+z^2}} u(\sqrt{x^2+y^2+z^2}) - v = 0.$$

However, we have

$$\begin{aligned}\frac{d}{dz}\left(\frac{z}{\sqrt{x^2+y^2+z^2}} u(\sqrt{x^2+y^2+z^2}) - v\right) &= \frac{z^2}{r^2}u'(r) + \frac{r^2-z^2}{r^2}\frac{u(r)}{r} \\ &= \frac{u(r)}{r}\left(1 - \frac{v^2}{u(r)^2}\right) + \frac{v^2 u'(r)}{u(r)^2}.\end{aligned}$$

While this derivative is still greater than 0, so that the left hand side is monotonic, notice that is equal to the expression in (3) multiplied by $\frac{u(r)}{2r^2}$. That is, it is much smaller than (3). This means that solving for r can be done with greater accuracy and precision than solving for z, even if theoretically they should give the same answer.

8 Expansion and Reprojection

8.1 Method 1 - Transition through spherical coordinates

The new expanded coordinates were calculated in spherical coordinates with the assumption that gas only moves in the radial direction, then were converted back to cartesian coordinates. Assuming a constant expansion velocity, the new coordinates are easily calculated as $(r_0 + v_{exp}\Delta t, \theta, \phi)$. If the expansion were to instead use the beta velocity law, it needs to be done by solving the differential equation

$$\frac{dr}{dt} = u_\beta(r)$$

Which was done numerically using `scipy.integrate.solve_ivp`. However, at large values of r the beta law is asymptotic to the extent that there is a negligible difference between using it versus the constant velocity model. Because of this, a constant velocity model was preferred. Due to the discreteness of each pixel, when data is expanded out, there will be a significant gap in the ends of the expanded cloud. To fill in this missing data, spherical datapoints were added with a radius and intensity equal to their respective mean values, then. To ensure that the total flux is not artificially inflated by the additional points, the data was normalised by multiplying the entire dataset by a factor:

$$\frac{\sum \text{original data}}{\sum \text{current data}}$$

To reproject the gas cloud, the data was divided into discrete velocity channels based on its velocity in the direction of the line of sight. Assuming that all gas at this point expands at the expansion velocity, we have

$$v_c = v_{exp} \frac{z}{\sqrt{x^2 + y^2 + z^2}}$$

8.2 Method 2

When it isn't necessary to add additional data points, it can be more precise and efficient to do fewer calculations, going from a gas cloud in velocity space directly to the expanded gas cloud in velocity space. Let $E_{r'}((x, y, z)) := \frac{r'}{r}(x, y, z)$ be a map that expands a point to have a radius of r'. Let (x', y', v') and (x', y', z') denote the coordinates of the expanded point in velocity space and physical space, while (x, y, v) and (x, y, z) denote the original coordinates of the point. We have

$$(x', y', z') = \left(\frac{r'}{r}x, \frac{r'}{r}y, \frac{r'}{r}z\right)$$
$$(x, y, v) = \left(x, y, \frac{z}{r}u(r)\right)$$
$$(x', y', v') = \left(x', y', \frac{z'}{r'}u(r')\right).$$

Immediately we see that $x' = \frac{r'}{r}x$ and $y' = \frac{r'}{r}y$. Finally, taking v'/v we obtain

$$\frac{v'}{v} = \frac{\frac{z'}{r'}u(r')}{\frac{z}{r}u(r)} = \frac{z'}{z} \cdot \frac{r}{r'} \cdot \frac{u(r')}{u(r)} = \frac{u(r')}{u(r)},$$

so we conclude that

$$(x', y', v') = \left(\frac{r'}{r}x, \frac{r'}{r}y, \frac{u(r')}{u(r)}v\right).$$

We can find r using the method outlined in section 7.2. Then r' is the new radius, which is computed from r, the time period t, and the velocity law $u(r)$. As in method 1, we solve

$$\frac{dr}{dt} = u(r)$$

with initial condition $r(0) = r$, to find $r' = r(t)$.

Note that when we use a constant velocity model, $u(r') = u(r)$ and hence $v' = v$. Then the velocity channel of each point can be kept the same.

Method 2 is used in the `agb-star-deprojection-methods` package, while Method 1 can be found in `test.py`. As a result, the end velocity channels of expanded gas clouds given by the package are not especially reliable, as the few data points can be over-expanded, especially using a constant velocity.

9 Faster interpolation

Because of the workings of `Plotly` and the `StarData` class, for deprojection and expansion methods (which change the coordinates) it was necessary to obtain a grid of data. This is difficult, as our initial data is gridded, and neither deprojection nor expansion sends grids to grids.

One approach was to interpolate gridded data using `scipy.interpolate.griddata`. This was functional but incredibly slow for large numbers of data points. It is generally difficult to interpolate between unstructured data, so we wanted a way to interpolate between gridded data instead. Here is the approach that is taken in the `agb-star-deprojection-methods` package.

1. Find bounds on the transformed (ie. deprojected or expanded) space, and generate a grid of data points within these bounds.

2. Perform the *inverse* operation (projection/de-expansion) on this new grid, and discard points that are out-of-bounds of the original gridded space. This is the *preimage* of the transformed space.

3. Interpolate the original gridded space to the preimage of the transformed space.

4. Identify these points with the transformed space.

This allows us to use `scipy.interpolate.interpn`, which is much faster as it interpolated within a grid.

10 Conclusion

Here is the secret of our deprojection package: it doesn't do any deprojection! Although we have documented methods that we've tried in this report, the best thing to do turned out to be never calculating the deprojection at all. Instead, we take a box of spatial coordinates (x, y, z), and for each coordinate, to find the associated intensity, we compute (x, y, v) (using *forwards* projection), and then get the intensity at (x, y, v) by interpolating. We don't need to deal with approximating solutions to equations, or using ill-conditioned functions; all we need is the forwards projection.