GoFish

Release v1.1.1

Aug 30, 2023

Contents:

1 Background **3**

2 Installation **5**

3 Fishing in *uv* Space **7**

4 Citation **9**

5 Community Guidelines **11**

5.1 API 11
5.2 Coordinate System in `GoFish` 35
5.3 Basics of `GoFish` 44
5.4 Advanced Fishing 62
5.5 Masking with `GoFish` 66
5.6 Searching For (Sub-)Structure in Disks 71

Python Module Index **77**

Index **79**

`GoFish` is a set of Python tools to exploit the known rotation of a protoplanetary disk to shift all emission to a common line center in order to stack them, increasing the signal-to-noise of the spectrum, detecting weaker lines, or super-sampling the spectrum to better resolve the line profile.

CHAPTER 1

Background

The method was first described in Yen et al. (2016), although other groups were using similar methods, such as Teague et al. (2016) and Matra et al. (2017) with varying applications. By exploiting the known rotation structure of the disk we can:

- Extract previously undetected line emission.
- Azimuthally average spectra to get a significant boost in the SNR.
- Super-sample the spectra to boost the spectral resolution of the data.

Details of the above examples can be found in the tutorials.

In Teague et al. (2018a) and Teague et al. (2018b), this method was inverted to use bright line emission to infer the rotation profile of the gas. You can use the functionality of `GoFish` with that of `eddy` (Teague 2019) to perform similar analyses.

Note: This documentation was written with a view to being used with ALMA data. However, this method works equally well with any PPV data obtained with any IFU instrument.

CHAPTER 2

Installation

Simply use PyPI with:

```
pip install gofish
```

which should install the necessary dependencies. If you have any trouble installing, please raise an issue.

If the installation went to plan you should be able to run the tutorial notebooks.

CHAPTER 3

Fishing in *uv* Space

`GoFish` works in the image plane, which allows the user flexibility in masking certain spatial regions. However, with this comes the complication of complex spatial correlations due to the highly non-linear imaging process.

We would strongly recommend using `VISIBLE` (Loomis et al. 2017), which is a match-filtering approach to finding weak line emission. This has the significant advantage of not requiring any imaging as it works directly on the measurement sets and avoids any issues with correlated noise.

CHAPTER 4

Citation

If you use *GoFish* as part of your research, please cite the JOSS article:

```
@article{GoFish,
    doi = {10.21105/joss.01632},
    url = {https://doi.org/10.21105/joss.01632},
    year = {2019},
    month = {sep},
    publisher = {The Open Journal},
    volume = {4},
    number = {41},
    pages = {1632},
    author = {Richard Teague},
    title = {GoFish: Fishing for Line Observations in Protoplanetary Disks},
    journal = {The Journal of Open Source Software}
}
```

as well as any of the above referenced papers for the method.

CHAPTER 5

Community Guidelines

GoFish is being actively developed on GitHub. If you find any bugs, or want to suggest any improvements, please follow the contributing guide and open an issue.

5.1 API

class gofish.**imagecube**(*path*, *FOV=None*, *velocity_range=None*, *verbose=True*, *primary_beam=None*, *bunit=None*, *pixel_scale=None*)

Base class containing all the FITS data. Must be a 3D cube containing two spatial and one velocity axis for and spectral shifting. A 2D 'cube' can be used to make the most of the deprojection routines. These can easily be made from CASA using the `exportfits()` command.

Parameters

- **path** (*str*) – Relative path to the FITS cube.
- **FOV** (*Optional[float]*) – Clip the image cube down to a specific field-of-view spanning a range `FOV`, where `FOV` is in [arcsec].
- **velocity_range** (*Optional[tuple]*) – A tuple of minimum and maximum velocities to clip the velocity range to.
- **verbose** (*Optional[bool]*) – Whether to print out warning messages.
- **primary_beam** (*Optional[str]*) – Path to the primary beam as a FITS file to apply the correction.
- **bunit** (*Optional[str]*) – If no *bunit* header keyword is found, use this value, e.g., 'Jy/beam'.
- **pixel_scale** (*Optional[float]*) – If no axis information is found in the header, use this value for the pixel scaling in [arcsec], assuming an image centered on 0.0".

average_spectrum(*r_min*, *r_max*, *inc*, *PA*, *mstar*, *dist*, *dr=None*, *PA_min=None*, *PA_max=None*, *exclude_PA=False*, *abs_PA=False*, *x0=0.0*, *y0=0.0*, *z0=None*, *psi=None*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *z_func=None*, *resample=1*, *beam_spacing=False*, *mask_frame='disk'*, *unit='Jy/beam'*, *mask=None*, *skip_empty_annuli=True*, *shadowed=False*, *empirical_uncertainty=False*, *include_spectral_decorrelation=True*, *velocity_resolution=1.0*, *vrad_func=None*)

Return the averaged spectrum over a given radial range, returning a spectrum in units of [Jy/beam] or [K] using the Rayleigh-Jeans approximation. As the Keplerian rotation of the disk needs to be accounted for, a stellar mass in [Msun] and distance in [pc] must be provided.

The *resample* parameter allows you to resample the spectrum at a different velocity spacing (by providing a float argument) or averaging of an integer number of channels (by providing an integer argument). For example, `resample=3`, will return a velocity axis which has been supersampled such that a channel is three times as narrow as the intrinsic width. Instead, `resample=50.0` will result in a velocity axis with a channel spacing of 50 m/s.

The third variable returned is the standard error on the mean of each velocity bin, i.e. the standard deviation of that velocity bin divided by the square root of the number of samples.

Parameters

- **r_min** (*float*) – Inner radius in [arcsec] of the region to average.
- **r_max** (*float*) – Outer radius in [arcsec] of the region to average.
- **inc** (*float*) – Inclination of the disk in [degrees].
- **PA** (*float*) – Position angle of the disk in [degrees], measured east-of-north towards the redshifted major axis.
- **mstar** (*float*) – Stellar mass in [Msun].
- **dist** (*float*) – Distance to the source in [pc].
- **dr** (*Optional[float]*) – Width of the annuli to split the integrated region into in [arcsec]. Default is quater of the beam major axis.
- **PA_min** (*Optional[float]*) – Minimum polar angle of the segment of the annulus in [degrees]. Note this is the polar angle, not the position angle.
- **PA_max** (*Optional[float]*) – Maximum polar angle of the segment of the annulus in [degrees]. Note this is the polar angle, not the position angle.
- **exclude_PA** (*Optional[bool]*) – If `True`, exclude the provided polar angle range rather than include it.
- **abs_PA** (*Optional[bool]*) – If `True`, take the absolute value of the polar angle such that it runs from 0 [deg] to 180 [deg].
- **x0** (*Optional[float]*) – Source center offset along the x-axis in [arcsec].
- **y0** (*Optional[float]*) – Source center offset along the y-axis in [arcsec].
- **z0** (*Optional[float]*) – Emission height in [arcsec] at a radius of 1".
- **psi** (*Optional[float]*) – Flaring angle of the emission surface.
- **z_func** (*Optional[function]*) – A function which provides $z(r)$. Note that no checking will occur to make sure this is a valid function.
- **resample** (*Optional[float/int]*) – Resampling parameter for the deprojected spectrum. An integer specifies an average of that many channels, while a float specifies the desired channel width. Default is `resample=1`.

- **beam_spacing** (*Optional[bool]*) – When extracting the annuli, whether to choose spatially independent pixels or not.
- **PA_min** – Minimum polar angle to include in the annulus in [degrees]. Note that the polar angle is measured in the disk-frame, unlike the position angle which is measured in the sky-plane.
- **PA_max** – Maximum polar angleto include in the annulus in [degrees]. Note that the polar angle is measured in the disk-frame, unlike the position angle which is measured in the sky-plane.
- **exclude_PA** – Whether to exclude pixels where `PA_min <= PA_pix <= PA_max`.
- **mask_frame** (*Optional[str]*) – Which frame the radial and azimuthal mask is specified in, either `'disk'` or `'sky'`.
- **unit** (*Optional[str]*) – Units for the spectrum, either `'Jy/beam'` or `'K'`. Note that the conversion to Kelvin assumes the Rayleigh-Jeans approximation which is typically invalid at sub-mm wavelengths.
- **mask** (*Optional[ndarray]*) – Either a 2D or 3D mask to use when averaging the spectra. This will be used in addition to any geometrically defined mask.
- **assume_correlated** (*Optional[bool]*) – Whether to treat the spectra which are stacked as correlated, by default this is `True`. If `False`, the uncertainty will be estimated using Poisson statistics, otherwise the uncertainty is just the standard deviation of each velocity bin.
- **skip_empty_annuli** (*Optional[bool]*) – If `True`, skip any annuli which are empty (i.e. their masks have zero pixels in). If `False`, any empty masks will raise a `ValueError`.
- **empirical_uncertainty** (*Optional[bool]*) – If `True`, use an empirical measure of the uncertainty based on an iterative sigma clipping described in `imagecube.estimate_uncertainty`.
- **include_spectral_decorrlation** (*Optional[bool]*) – If `True`, take account of the spectral decorrelation when estimating uncertainties on the average spectrum and return a spectral correlation length too. Defaults to `True`.
- **velocity_resolution** (*Optional[float]*) – Velocity resolution of the data as a fraction of the channel spacing. Defaults to `1.0`.
- **vrad_func** (*Optional[callable]*) – Define any radial velocity component here. Must return a radial velocity in [m/s] for a given radial position in [arcsec].

Returns The velocity axis of the spectrum, `velax`, in [m/s], the averaged spectrum, `spectrum`, and the variance of the velocity bin, `scatter`. The latter two are in units of either [Jy/beam] or [K] depending on the `unit`.

static estimate_uncertainty(*a*, *nsigma=3.0*, *niter=20*)

Estimate the noise by iteratively sigma-clipping. For each iteraction the `a` array is masked above `abs(a) > nsigma * std` and the standard deviation, `std` calculated. This is repeated until either convergence or for `niter` iteractions. In some cases, usually with low `nsigma` values, the `std` will approach zero and all `a` values are masked, resulting in an NaN. In this case, the function will return the last finite value.

Parameters

- **a** (*array*) – Array of data from which to estimate the uncertainty.
- **nsigma** (*Optional[float]*) – Factor of the standard devitation above which to mask `a` values.

- **niter** (*Optional[int]*) – Number of iterations to halt after if convergence is not reached.

Returns Standard deviation of the sigma-clipped data.

Return type std (float)

integrated_spectrum(*r_min*, *r_max*, *inc*, *PA*, *mstar=None*, *dist=None*, *dr=None*, *x0=0.0*, *y0=0.0*, *z0=None*, *psi=None*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *z_func=None*, *resample=1*, *beam_spacing=False*, *PA_min=None*, *PA_max=None*, *exclude_PA=False*, *abs_PA=False*, *mask=None*, *mask_frame='disk'*, *empirical_uncertainty=False*, *skip_empty_annuli=True*, *shadowed=False*, *velocity_resolution=1.0*, *vrad_func=None*, *include_spectral_decorrelation=True*)

Return the integrated spectrum over a given radial range, returning a spectrum in units of [Jy]. If a stellar mass and distance are specified then this will split the data into concentric annuli, correcting each annulus for the Keplerian rotation of the disk to generate a more precise measurement. If no stellar mass or distance are given then a simple spatial integration will be used.

The *resample* parameter allows you to resample the spectrum at a different velocity spacing (by providing a float argument) or averaging of an integer number of channels (by providing an integer argument). For example, `resample=3`, will return a velocity axis which has been supersampled such that a channel is three times as narrow as the intrinsic width. Instead, `resample=50.0` will result in a velocity axis with a channel spacing of 50 m/s.

Note: The third variable returned is the scatter in each velocity bin and not the uncertainty on the bin mean as the data is not strictly independent due to spectral and spatial correlations in the data. If you want to assume uncorrelated data to get a better estimate of the uncertainty, set `assumed_correlated=False`.

Parameters

- **r_min** (*float*) – Inner radius in [arcsec] of the region to integrate.
- **r_max** (*float*) – Outer radius in [arcsec] of the region to integrate.
- **inc** (*float*) – Inclination of the disk in [degrees].
- **PA** (*float*) – Position angle of the disk in [degrees], measured east-of-north towards the redshifted major axis.
- **mstar** (*Optional[float]*) – Stellar mass in [Msun].
- **dist** (*Optional[float]*) – Distance to the source in [pc].
- **dr** (*Optional[float]*) – Width of the annuli to split the integrated region into in [arcsec]. Default is quater of the beam major axis.
- **x0** (*Optional[float]*) – Source center offset along the x-axis in [arcsec].
- **y0** (*Optional[float]*) – Source center offset along the y-axis in [arcsec].
- **z0** (*Optional[float]*) – Emission height in [arcsec] at a radius of 1".
- **psi** (*Optional[float]*) – Flaring angle of the emission surface.
- **z_func** (*Optional[function]*) – A function which provides $z(r)$. Note that no checking will occur to make sure this is a valid function.
- **resample** (*Optional[float/int]*) – Resampling parameter for the deprojected spectrum. An integer specifies an average of that many channels, while a float specifies the desired channel width. Default is `resample=1`.

- **beam_spacing** (*Optional[bool]*) – When extracting the annuli, whether to choose spatially independent pixels or not.
- **PA_min** (*Optional[float]*) – Minimum polar angle to include in the annulus in [degrees]. Note that the polar angle is measured in the disk-frame, unlike the position angle which is measured in the sky-plane.
- **PA_max** (*Optional[float]*) – Maximum polar angleto include in the annulus in [degrees]. Note that the polar angle is measured in the disk-frame, unlike the position angle which is measured in the sky-plane.
- **exclude_PA** (*Optional[bool]*) – Whether to exclude pixels where `PA_min <= PA_pix <= PA_max`.
- **abs_PA** (*Optional[bool]*) – If `True`, take the absolute value of the polar angle such that it runs from 0 [deg] to 180 [deg].
- **mask** (*Optional[ndarray]*) – Either a 2D or 3D mask to use when averaging the spectra. This will be used in addition to any geometrically defined mask.
- **mask_frame** (*Optional[str]*) – Which frame the radial and azimuthal mask is specified in, either `'disk'` or `'sky'`.
- **assume_correlated** (*Optional[bool]*) – Whether to treat the spectra which are stacked as correlated, by default this is `True`. If `False`, the uncertainty will be estimated using Poisson statistics, otherwise the uncertainty is just the standard deviation of each velocity bin.
- **skip_empty_annuli** (*Optional[bool]*) – If `True`, skip any annuli which are empty (i.e. their masks have zero pixels in). If `False`, any empty masks will raise a `ValueError`.
- **vrad_func** (*Optional[callable]*) – Define any radial velocity component here. Must return a radial velocity in [m/s] for a given radial position in [arcsec].

Returns The velocity axis of the spectrum, `velax`, in [m/s], the integrated spectrum, `spectrum`, and the variance of the velocity bin, `scatter`. The latter two are in units of [Jy].

radial_spectra(*rvals=None*, *rbins=None*, *dr=None*, *x0=0.0*, *y0=0.0*, *inc=0.0*, *PA=0.0*, *z0=None*, *psi=None*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *z_func=None*, *mstar=1.0*, *dist=100.0*, *resample=1*, *beam_spacing=False*, *r_min=None*, *r_max=None*, *PA_min=None*, *PA_max=None*, *exclude_PA=None*, *abs_PA=False*, *mask_frame='disk'*, *mask=None*, *unit='Jy/beam'*, *shadowed=False*, *skip_empty_annuli=True*, *empirical_uncertainty=False*, *vrad_func=None*)

Return shifted and stacked spectra, over a given spatial region in the disk. The averaged spectra can be rescaled by using the `unit` argument for which the possible units are:

- `'mJy/beam'`
- `'Jy/beam'`
- `'mK'`
- `'K'`
- `'mJy'`
- `'Jy'`

where `'mJy'` or `'Jy'` will integrate the emission over the defined spatial range assuming that the averaged spectrum is the same for all pixels in that region.

In addition to a spatial region specified by the usual geometrical mask properties (`r_min`, `r_max`, `PA_min`, `PA_max`), a user-defined can be included, either as a 2D array (such that it is constant as a function of velocity), or as a 3D array, for example a CLEAN mask, for velocity-specific masks.

There are two ways to return the uncertainties for the spectra. The default are the straight *statistical* uncertainties which are propagated through from the `annulus` class. Sometimes these can appear to give a poor description of the true variance of the data. In this case the user can use `empirical_uncertainty=True` which will use an iterative sigma-clipping approach to estimate the uncertainty by the variance in line-free regions of the spectrum.

Note: Calculation of the empirircal uncertainty is experimental. Use the results with caution and if anything looks suspicious, please contact me.

Parameters

- **rvals** (*Optional[floats]*) – Array of bin centres for the profile in [arcsec]. You need only specify one of `rvals` and `rbins`.
- **rbins** (*Optional[floats]*) – Array of bin edges for the profile in [arcsec]. You need only specify one of `rvals` and `rbins`.
- **dr** (*Optional[float]*) – Width of the radial bins in [arcsec] to use if neither `rvals` nor `rbins` is set. Default is 1/4 of the beam major axis.
- **x0** (*Optional[float]*) – Source center offset along the x-axis in [arcsec].
- **y0** (*Optional[float]*) – Source center offset along the y-axis in [arcsec].
- **inc** (*Optional[float]*) – Inclination of the disk in [degrees].
- **PA** (*Optional[float]*) – Position angle of the disk in [degrees], measured east-of-north towards the redshifted major axis.
- **z0** (*Optional[float]*) – Emission height in [arcsec] at a radius of 1”.
- **psi** (*Optional[float]*) – Flaring angle of the emission surface.
- **z_func** (*Optional[function]*) – A function which provides $z(r)$. Note that no checking will occur to make sure this is a valid function.
- **mstar** (*Optional[float]*) – Stellar mass in [Msun].
- **dist** (*Optional[float]*) – Distance to the source in [pc].
- **resample** (*Optional[float/int]*) – Resampling parameter for the deprojected spectrum. An integer specifies an average of that many channels, while a float specifies the desired channel width. Default is `resample=1`.
- **beam_spacing** (*Optional[bool]*) – When extracting the annuli, whether to choose spatially independent pixels or not. Default it not.
- **r_min** (*Optional[float]*) – Inner radius in [arcsec] of the region to integrate. The value used will be greater than or equal to `r_min`.
- **r_max** (*Optional[float]*) – Outer radius in [arcsec] of the region to integrate. The value used will be less than or equal to `r_max`.
- **PA_min** (*Optional[float]*) – Minimum polar angle to include in the annulus in [degrees]. Note that the polar angle is measured in the disk-frame, unlike the position angle which is measured in the sky-plane.

- **PA_max** (*Optional[float]*) – Maximum polar angleto include in the annulus in [degrees]. Note that the polar angle is measured in the disk-frame, unlike the position angle which is measured in the sky-plane.
- **exclude_PA** (*Optional[bool]*) – Whether to exclude pixels where `PA_min <= PA_pix <= PA_max`. Default is `False`.
- **abs_PA** (*Optional[bool]*) – If `True`, take the absolute value of the polar angle such that it runs from 0 [deg] to 180 [deg].
- **mask_frame** (*Optional[str]*) – Which frame the radial and azimuthal mask is specified in, either `'disk'` or `'sky'`.
- **mask** (*Optional[arr]*) – A 2D or 3D mask to include in addition to the geometrical mask. If 3D, must match the shape of the `data` array, while a 2D mask will be interpretted as a velocity independent mask and must match `data[0].shape`.
- **unit** (*Optional[str]*) – Desired unit of the output spectra.
- **shadowed** (*Optional[bool]*) – If `True`, use a slower but more accurate deprojection algorithm which will handle shadowed pixels due to sub-structure.
- **skip_empty_annuli** (*Optional[bool]*) – If `True`, skip any annuli which are found to have no finite spectra in them, returning `NaN` for that annlus. Otherwise raise a `ValueError`.
- **empirical_uncertainty** (*Optional[bool]*) – Whether to calculate the uncertainty on the spectra empirically (`True`) or using the statistical uncertainties.

Returns Four arrays. An array of the bin centers, `rvals`, an array of the velocity axis, `velax`, and the averaged spectra, `spectra` and their associated uncertainties, `scatter`. The latter two will have shapes of `(rvals.size, velax.size)` and wil be in units given by the `unit` argument.

radial_profile(*rvals=None*, *rbins=None*, *dr=None*, *unit='Jy/beam m/s'*, *x0=0.0*, *y0=0.0*, *inc=0.0*, *PA=0.0*, *z0=0.0*, *psi=1.0*, *r_cavity=0.0*, *r_taper=<sphinx.ext.autodoc.importer._MockObject object>*, *q_taper=1.0*, *z_func=None*, *mstar=0.0*, *dist=100.0*, *resample=1*, *beam_spacing=False*, *r_min=None*, *r_max=None*, *PA_min=None*, *PA_max=None*, *exclude_PA=False*, *abs_PA=False*, *mask_frame='disk'*, *mask=None*, *velo_range=None*, *assume_correlated=True*, *shadowed=False*)

Generate a radial profile from shifted and stacked spectra. There are different ways to collapse the spectrum into a single value using the `unit` argument. Possible units for the flux (density) are:

- `'mJy/beam'`
- `'Jy/beam'`
- `'mK'`
- `'K'`
- `'mJy'`
- `'Jy'`

where `'/beam'` is equivalent to `'/pix'` if not beam information is found in the FITS header. For the velocity we have:

- `'m/s'`
- `'km/s'`
- `''`

with the empty string resulting in the peak value of the spectrum. For example, `unit='K'` will return the peak brightness temperature as a function of radius, while `unit='K km/s'` will return the velocity integrated brightness temperature as a function of radius.

All conversions from [Jy/beam] to [K] are performed using the Rayleigh-Jeans approximation. For other units, or to develop more sophisticated statistics for the collapsed line profiles, use the `radial_spectra` function which will return the shifted and stacked line profiles.

Parameters

- **rvals** (*Optional[floats]*) – Array of bin centres for the profile in [arcsec]. You need only specify one of `rvals` and `rbins`.
- **rbins** (*Optional[floats]*) – Array of bin edges for the profile in [arcsec]. You need only specify one of `rvals` and `rbins`.
- **dr** (*Optional[float]*) – Width of the radial bins in [arcsec] to use if neither `rvals` nor `rbins` is set. Default is 1/4 of the beam major axis.
- **unit** (*Optional[str]*) – Unit for the y-axis of the profile, as in the function description.
- **x0** (*Optional[float]*) – Source center offset along the x-axis in [arcsec].
- **y0** (*Optional[float]*) – Source center offset along the y-axis in [arcsec].
- **inc** (*Optional[float]*) – Inclination of the disk in [degrees].
- **PA** (*Optional[float]*) – Position angle of the disk in [degrees], measured east-of-north towards the redshifted major axis.
- **z0** (*Optional[float]*) – Emission height in [arcsec] at a radius of 1".
- **psi** (*Optional[float]*) – Flaring angle of the emission surface.
- **z_func** (*Optional[function]*) – A function which provides $z(r)$. Note that no checking will occur to make sure this is a valid function.
- **mstar** (*Optional[float]*) – Stellar mass in [Msun].
- **dist** (*Optional[float]*) – Distance to the source in [pc].
- **resample** (*Optional[float/int]*) – Resampling parameter for the deprojected spectrum. An integer specifies an average of that many channels, while a float specifies the desired channel width. Default is `resample=1`.
- **beam_spacing** (*Optional[bool]*) – When extracting the annuli, whether to choose spatially independent pixels or not.
- **r_min** (*Optional[float]*) – Inner radius in [arcsec] of the region to integrate. The value used will be greater than or equal to `r_min`.
- **r_max** (*Optional[float]*) – Outer radius in [arcsec] of the region to integrate. The value used will be less than or equal to `r_max`.
- **PA_min** (*Optional[float]*) – Minimum polar angle to include in the annulus in [degrees]. Note that the polar angle is measured in the disk-frame, unlike the position angle which is measured in the sky-plane.
- **PA_max** (*Optional[float]*) – Maximum polar angleto include in the annulus in [degrees]. Note that the polar angle is measured in the disk-frame, unlike the position angle which is measured in the sky-plane.
- **exclude_PA** (*Optional[bool]*) – Whether to exclude pixels where `PA_min <= PA_pix <= PA_max`.

- **abs_PA** (*Optional[bool]*) – If `True`, take the absolute value of the polar angle such that it runs from 0 [deg] to 180 [deg].
- **mask_frame** (*Optional[str]*) – Which frame the radial and azimuthal mask is specified in, either `'disk'` or `'sky'`.
- **mask** (*Optional[array]*) – A user-specified 2D or 3D array to mask the data with prior to shifting and stacking.
- **velo_range** (*Optional[tuple]*) – A tuple containing the spectral range to integrate if required for the provided `unit`. Can either be a string, including units, or as channel integers.
- **shadowed** (*Optional[bool]*) – If `True`, use a slower algorithm for deprojecting the pixel coordinates into disk-center coordiantes which can handle shadowed pixels.

Returns Arrays of the bin centers in [arcsec], the profile value in the requested units and the associated uncertainties.

shifted_cube(*inc*, *PA*, *x0=0.0*, *y0=0.0*, *z0=None*, *psi=None*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *z_func=None*, *mstar=None*, *dist=None*, *vmod=None*, *r_min=None*, *r_max=None*, *fill_val=<sphinx.ext.autodoc.importer._MockObject object>*, *save=False*, *shadowed=False*)

Apply the velocity shift to each pixel and return the cube, or save as as new FITS file. This would be useful if you want to create moment maps of the data where you want to integrate over a specific velocity range without having to worry about the Keplerian rotation in the disk.

Parameters

- **inc** (*float*) – Inclination of the disk in [degrees].
- **PA** (*float*) – Position angle of the disk in [degrees], measured east-of-north towards the redshifted major axis.
- **x0** (*Optional[float]*) – Source center offset along the x-axis in [arcsec].
- **y0** (*Optional[float]*) – Source center offset along the y-axis in [arcsec].
- **z0** (*Optional[float]*) – Emission height in [arcsec] at a radius of 1".
- **psi** (*Optional[float]*) – Flaring angle of the emission surface.
- **z_func** (*Optional[callable]*) – A function which returns the emission height in [arcsec] for a radius given in [arcsec].
- **mstar** (*Optional[float]*) – Stellar mass in [Msun].
- **dist** (*Optional[float]*) – Distance to the source in [pc].
- **v0** (*Optional[callable]*) – A function which returns the projected line of sight velocity in [m/s] for a radius given in [m/s].
- **r_min** (*Optional[float]*) – The inner radius in [arcsec] to shift.
- **r_max** (*Optional[float]*) – The outer radius in [arcsec] to shift.

Returns The shifted data cube.

keplerian(*inc*, *PA*, *mstar*, *dist*, *x0=0.0*, *y0=0.0*, *vlsr=0.0*, *z0=None*, *psi=None*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *z_func=None*, *r_min=None*, *r_max=None*, *cylindrical=False*, *shadowed=False*)

Projected Keplerian velocity profile in [m/s]. For positions outside `r_min` and `r_max` the values will be set to NaN.

Parameters

- **inc** (*float*) – Inclination of the disk in [degrees].
- **PA** (*float*) – Position angle of the disk in [degrees], measured east-of-north towards the redshifted major axis.
- **mstar** (*float*) – Stellar mass in [Msun].
- **dist** (*float*) – Distance to the source in [pc].
- **x0** (*Optional[float]*) – Source center offset along the x-axis in [arcsec].
- **y0** (*Optional[float]*) – Source center offset along the y-axis in [arcsec].
- **vlsr** (*Optional[float]*) – Systemic velocity in [m/s].
- **z0** (*Optional[float]*) – Emission height in [arcsec] at a radius of 1".
- **psi** (*Optional[float]*) – Flaring angle of the emission surface.
- **r_cavity** (*Optional[float]*) – Edge of the inner cavity for the emission surface in [arcsec].
- **r_taper** (*Optional[float]*) – Characteristic radius in [arcsec] of the exponential taper to the emission surface.
- **q_taper** (*Optional[float]*) – Exponent of the exponential taper of the emission surface.
- **z_func** (*Optional[callable]*) – A function which returns the emission height in [arcsec] for a radius given in [arcsec].
- **r_min** (*Optional[float]*) – The inner radius in [arcsec] to model.
- **r_max** (*Optional[float]*) – The outer radius in [arcsec] to model.
- **cylindrical** (*Optional[bool]*) – If `True`, force cylindrical rotation, i.e. ignore the height in calculating the velocity.
- **shadowed** (*Optional[bool]*) – If `True`, use a slower algorithm for deprojecting the pixel coordinates into disk-center coordiantes which can handle shadowed pixels.

find_center(*dx=None*, *dy=None*, *Nx=None*, *Ny=None*, *mask=None*, *v_min=None*, *v_max=None*, *spectrum='avg'*, *SNR='peak'*, *normalize=True*, ***kwargs*)

Find the source center (assuming the disk is azimuthally symmetric) by calculating the SNR of the averaged spectrum by varying the source center, `(x0, y0)`.

Parameters

- **dx** (*Optional[float]*) – Maximum offset to consider in the *x* direction in [arcsec]. Default is one beam FWHM.
- **dy** (*Optional[float]*) – Maximum offset to consider in the *y* direction in [arcsec]. Default is one beam FWHM.
- **Nx** (*Optional[int]*) – Number of samples to take along the *x* direction. Default results in roughtly pixel spacing.
- **Ny** (*Optional[int]*) – Number of samples to take along the *y* direction. Default results in roughtly pixel spacing.
- **mask** (*Optional[array]*) – Boolean mask of channels to use when calculating the integrated flux or RMS noise.
- **v_min** (*Optional[float]*) – Minimum velocity in [m/s] to consider if an explicit mask is not provided.

- **v_max** (*Optional[float]*) – Maximum velocity in [m/s] to consider if an explicit mask is not provided.
- **spectrum** (*Optional[str]*) – Type of spectrum to consider, either the integrated spectrum with `'int'`, or the average spectrum with `'avg'`.
- **SNR** (*Optional[str]*) – Type of signal-to-noise definition to use. Either `'int'` to use the integrated flux density as the signal, or `'peak'` to use the maximum value.
- **normalize** (*Optional[bool]*) – Whether to normalize the SNR map relative to the SNR at `(x0, y0) = (0, 0)`.

Returns The axes of the grid search, `x0s` and `y0s`, as well as the 2D array of SNR values, `SNR`.

get_vlos(*rvals=None*, *rbins=None*, *r_min=None*, *r_max=None*, *PA_min=None*, *PA_max=None*, *exclude_PA=False*, *abs_PA=False*, *x0=0.0*, *y0=0.0*, *inc=0.0*, *PA=0.0*, *z0=None*, *psi=None*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *z_func=None*, *mstar=None*, *dist=None*, *mask=None*, *mask_frame='disk'*, *beam_spacing=True*, *fit_vrad=True*, *annulus_kwargs=None*, *get_vlos_kwargs=None*, *shadowed=False*)

Infer the rotational and radial velocity profiles from the data following the approach described in **'Teague et al. (2018)'_**. The cube will be split into annuli, with the *get_vlos()* function from *annulus* being used to infer the rotational (and radial) velocities.

Parameters

- **rvals** (*Optional[floats]*) – Array of bin centres for the profile in [arcsec]. You need only specify one of `rvals` and `rbins`.
- **rbins** (*Optional[floats]*) – Array of bin edges for the profile in [arcsec]. You need only specify one of `rvals` and `rbins`.
- **r_min** (*float*) – Minimum midplane radius of the annuli in [arcsec].
- **r_max** (*float*) – Maximum midplane radius of the annuli in [arcsec].
- **PA_min** (*Optional[float]*) – Minimum polar angle of the segment of the annulus in [degrees]. Note this is the polar angle, not the position angle.
- **PA_max** (*Optional[float]*) – Maximum polar angle of the segment of the annulus in [degrees]. Note this is the polar angle, not the position angle.
- **exclude_PA** (*Optional[bool]*) – If `True`, exclude the provided polar angle range rather than include.
- **abs_PA** (*Optional[bool]*) – If `True`, take the absolute value of the polar angle such that it runs from 0 [deg] to 180 [deg].
- **x0** (*Optional[float]*) – Source right ascension offset [arcsec].
- **y0** (*Optional[float]*) – Source declination offset [arcsec].
- **inc** (*Optional[float]*) – Source inclination [deg].
- **PA** (*Optional[float]*) – Source position angle [deg]. Measured between north and the red-shifted semi-major axis in an easterly direction.
- **z0** (*Optional[float]*) – Aspect ratio at 1" for the emission surface. To get the far side of the disk, make this number negative.
- **psi** (*Optional[float]*) – Flaring angle for the emission surface.
- **z_func** (*Optional[function]*) – A function which provides z(r). Note that no checking will occur to make sure this is a valid function.

- **mstar** (*Optional[float]*) – Stellar mass of the central star in [Msun] to calculate the starting positions for the MCMC. If specified, must also provide `dist`.
- **dist** (*Optional[float]*) – Source distance in [pc] to use to calculate the starting positions for the MCMC. If specified, must also provide `mstar`.
- **mask** (*Optional[ndarray]*) – A 2D array mask which matches the shape of the data.
- **mask_frame** (*Optional[str]*) – Coordinate frame for the mask. Either `'disk'` or `'sky'`. If `disk` coordinates are used then the inclination and position angle of the mask are set to zero.
- **beam_spacing** (*Optional[bool/float]*) – If True, randomly sample the annulus such that each pixel is at least a beam FWHM apart. A number can also be used in place of a boolean which will describe the number of beam FWHMs to separate each sample by.
- **fit_vrad** (*Optional[bool]*) – Whether to include radial velocities in the optimization.
- **annulus_kwargs** (*Optional[dict]*) – Kwargs to pass to `get_annulus`.
- **get_vlos_kwargs** (*Optional[dict]*) – Kwargs to pass to `annulus.get_vlos`.

Returns The radial sampling of the annuli, `rpnts`, and a list of the values returned from `annulus.get_vlos`.

get_annulus(*r_min*, *r_max*, *PA_min=None*, *PA_max=None*, *exclude_PA=False*, *abs_PA=False*, *x0=0.0*, *y0=0.0*, *inc=0.0*, *PA=0.0*, *z0=None*, *psi=None*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *z_func=None*, *mask=None*, *mask_frame='disk'*, *beam_spacing=True*, *annulus_kwargs=None*, *shadowed=False*)

Return an annulus (or section of), of spectra and their polar angles. Can select spatially independent pixels within the annulus, however as this is random, each draw will be different.

Parameters

- **r_min** (*float*) – Minimum midplane radius of the annulus in [arcsec].
- **r_max** (*float*) – Maximum midplane radius of the annulus in [arcsec].
- **PA_min** (*Optional[float]*) – Minimum polar angle of the segment of the annulus in [degrees]. Note this is the polar angle, not the position angle.
- **PA_max** (*Optional[float]*) – Maximum polar angle of the segment of the annulus in [degrees]. Note this is the polar angle, not the position angle.
- **exclude_PA** (*Optional[bool]*) – If `True`, exclude the provided polar angle range rather than include.
- **abs_PA** (*Optional[bool]*) – If `True`, take the absolute value of the polar angle such that it runs from 0 [deg] to 180 [deg].
- **x0** (*Optional[float]*) – Source right ascension offset [arcsec].
- **y0** (*Optional[float]*) – Source declination offset [arcsec].
- **inc** (*Optional[float]*) – Source inclination [deg].
- **PA** (*Optional[float]*) – Source position angle [deg]. Measured between north and the red-shifted semi-major axis in an easterly direction.
- **z0** (*Optional[float]*) – Aspect ratio at 1" for the emission surface. To get the far side of the disk, make this number negative.
- **psi** (*Optional[float]*) – Flaring angle for the emission surface.

- **z_func** (*Optional[function]*) – A function which provides z(r). Note that no checking will occur to make sure this is a valid function.
- **mask** (*Optional[ndarray]*) – A 2D array mask which matches the shape of the data.
- **mask_frame** (*Optional[str]*) – Coordinate frame for the mask. Either `'disk'` or `'sky'`. If `disk` coordinates are used then the inclination and position angle of the mask are set to zero.
- **beam_spacing** (*Optional[bool/float]*) – If True, randomly sample the annulus such that each pixel is at least a beam FWHM apart. A number can also be used in place of a boolean which will describe the number of beam FWHMs to separate each sample by.

Returns If `annulus=True`, will return an `eddy.annulus` instance, otherwise will be an array containing the polar angles of each spectrum in [degrees] and the array of spectra, ordered in increasing polar angle.

get_mask(*r_min=None*, *r_max=None*, *exclude_r=False*, *PA_min=None*, *PA_max=None*, *exclude_PA=False*, *abs_PA=False*, *mask_frame='disk'*, *x0=0.0*, *y0=0.0*, *inc=0.0*, *PA=0.0*, *z0=None*, *psi=None*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *z_func=None*, *shadowed=False*)

Returns a 2D mask for pixels in the given region. The mask can be specified in either disk-centric coordinates, `mask_frame='disk'`, or on the sky, `mask_frame='sky'`. If sky-frame coordinates are requested, the geometrical parameters (`inc`, `PA`, `z0`, etc.) are ignored, however the source offsets, `x0`, `y0`, are still considered.

Parameters

- **r_min** (*Optional[float]*) – Minimum midplane radius of the annulus in [arcsec]. Defaults to minimum deprojected radius.
- **r_max** (*Optional[float]*) – Maximum midplane radius of the annulus in [arcsec]. Defaults to the maximum deprojected radius.
- **exclude_r** (*Optional[bool]*) – If `True`, exclude the provided radial range rather than include.
- **PA_min** (*Optional[float]*) – Minimum polar angle of the segment of the annulus in [degrees]. Note this is the polar angle, not the position angle.
- **PA_max** (*Optional[float]*) – Maximum polar angle of the segment of the annulus in [degrees]. Note this is the polar angle, not the position angle.
- **exclude_PA** (*Optional[bool]*) – If `True`, exclude the provided polar angle range rather than include it.
- **abs_PA** (*Optional[bool]*) – If `True`, take the absolute value of the polar angle such that it runs from 0 [deg] to 180 [deg].
- **x0** (*Optional[float]*) – Source center offset along the x-axis in [arcsec].
- **y0** (*Optional[float]*) – Source center offset along the y-axis in [arcsec].
- **inc** (*Optional[float]*) – Inclination of the disk in [degrees].
- **PA** (*Optional[float]*) – Position angle of the disk in [degrees], measured east-of-north towards the redshifted major axis.
- **z0** (*Optional[float]*) – Emission height in [arcsec] at a radius of 1".
- **psi** (*Optional[float]*) – Flaring angle of the emission surface.
- **z_func** (*Optional[function]*) – A function which provides $z(r)$. Note that no checking will occur to make sure this is a valid function.

Returns A 2D array mask matching the shape of a channel.

radial_sampling(*rbins=None*, *rvals=None*, *dr=None*)

Return bins and bin center values. If the desired bin edges are known, will return the bin edges and vice versa. If neither are known will return default binning with the desired spacing.

Parameters

- **rbins** (*Optional[list]*) – List of bin edges.
- **rvals** (*Optional[list]*) – List of bin centers.
- **dr** (*Optional[float]*) – Spacing of bin centers in [arcsec]. Defaults to a quarter of the beam major axis.

Returns List of bin edges. rpnts (list): List of bin centres.

Return type rbins (list)

background_residual(*x0=0.0*, *y0=0.0*, *inc=0.0*, *PA=0.0*, *z0=None*, *psi=None*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *z_func=None*, *r_min=None*, *r_max=None*, *PA_min=None*, *PA_max=None*, *exclude_PA=False*, *abs_PA=False*, *mask_frame='disk'*, *interp1d_kw=None*, *background_only=False*, *shadowed=False*)

Return the residual from an azimuthally avearged background. This is most appropriate for exploring azimuthally asymmetric emission in either the zeroth moment (integrated intensity) or the peak brightness temperature maps. As such, this function only works for 2D data.

The coordinates provided will be used to both build the azimuthally averaged profile (using the `radial_profile` function) and then project this onto the sky-plane. Any masking parameters used here will only be used when creating the azimuthally spectrum, but the residual with cover the entire data.

Parameters

- **x0** (*Optional[float]*) – Source right ascension offset [arcsec].
- **y0** (*Optional[float]*) – Source declination offset [arcsec].
- **inc** (*Optional[float]*) – Source inclination [deg].
- **PA** (*Optional[float]*) – Source position angle [deg]. Measured between north and the red-shifted semi-major axis in an easterly direction.
- **z0** (*Optional[float]*) – Aspect ratio at 1" for the emission surface. To get the far side of the disk, make this number negative.
- **psi** (*Optional[float]*) – Flaring angle for the emission surface.
- **z_func** (*Optional[function]*) – A function which provides z(r). Note that no checking will occur to make sure this is a valid function.
- **r_min** (*Optional[float]*) – Inner radius in [arcsec] of the region to integrate. The value used will be greater than or equal to `r_min`.
- **r_max** (*Optional[float]*) – Outer radius in [arcsec] of the region to integrate. The value used will be less than or equal to `r_max`.
- **PA_min** (*Optional[float]*) – Minimum polar angle to include in the annulus in [degrees]. Note that the polar angle is measured in the disk-frame, unlike the position angle which is measured in the sky-plane.
- **PA_max** (*Optional[float]*) – Maximum polar angleto include in the annulus in [degrees]. Note that the polar angle is measured in the disk-frame, unlike the position angle which is measured in the sky-plane.

- **exclude_PA** (*Optional[bool]*) – Whether to exclude pixels where `PA_min <= PA_pix <= PA_max`.
- **abs_PA** (*Optional[bool]*) – If `True`, take the absolute value of the polar angle such that it runs from 0 [deg] to 180 [deg].
- **mask_frame** (*Optional[str]*) – Which frame the radial and azimuthal mask is specified in, either `'disk'` or `'sky'`.
- **interp1d_kw** (*Optional[dict]*) – Kwargs to pass to `scipy.interpolate.interp1d`.
- **background_only** (*Optional[bool]*) – If True, return only the azimuthally averaged background rather than the residual. Default is `False`.

Returns

Residual between the data and the azimuthally averaged background. If `background_only = True` then this is just the azimuthally averaged background.

Return type residual (array)

disk_coords(*x0=0.0*, *y0=0.0*, *inc=0.0*, *PA=0.0*, *z0=None*, *psi=None*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *z_func=None*, *force_positive_surface=False*, *force_negative_surface=False*, *frame='cylindrical'*, *shadowed=False*, *extend=2.0*, *oversample=2.0*, *griddata_kw=None*)

Get the disk coordinates given certain geometrical parameters and an emission surface. The emission surface is most simply described as a power law profile,

$$z(r) = z_0 \times \left(\frac{r}{1''}\right)^{\psi}$$

where `z0` and `psi` can be provided by the user. With the increase in spatial resolution afforded by interferometers such as ALMA there are a couple of modifications that can be used to provide a better match to the data.

An inner cavity can be included with the `r_cavity` argument which makes the transformation:

$$\tilde{r} = \max(0, r - r_{\rm cavity})$$

Note that the inclusion of a cavity will mean that other parameters, such as `z0`, would need to change as the radial axis has effectively been shifted.

To account for the drop in emission surface in the outer disk where the gas surface density decreases there are two descriptions. The preferred way is to include an exponential taper to the power law profile,

$$z_{\rm tapered}(r) = z(r) \times \exp\left(-\left[\frac{r}{r_{\rm taper}}\right]^{q_{\rm taper}}\right)$$

where both `r_taper` and `q_taper` values must be set.

It is also possible to override this parameterization and directly provide a user-defined `z_func`. This allow for highly complex surfaces to be included. If this is provided, the other height parameters are ignored.

In some cases, the projection of the emission surface can lead to regions of the disk being 'shadowed' by itself along the line of sight to the observer. The default method for calculating the disk coordinates does not take this into account in preference to speed. If you work with some emission surface that suffers from this, you can use the `shadowed=True` argument which will use a more precise method to calculate the disk coordinates. This can be much slower for large cubes as it requires rotating large grids.

Parameters

- **x0** (*Optional[float]*) – Source right ascension offset [arcsec].

- **y0** (*Optional[float]*) – Source declination offset [arcsec].
- **inc** (*Optional[float]*) – Source inclination [deg].
- **PA** (*Optional[float]*) – Source position angle [deg]. Measured between north and the red-shifted semi-major axis in an easterly direction.
- **z0** (*Optional[float]*) – Aspect ratio at 1" for the emission surface. To get the far side of the disk, make this number negative.
- **psi** (*Optional[float]*) – Flaring angle for the emission surface.
- **r_cavity** (*Optional[float]*) – Edge of the inner cavity for the emission surface in [arcsec].
- **r_taper** (*Optional[float]*) – Characteristic radius in [arcsec] of the exponential taper to the emission surface.
- **q_taper** (*Optional[float]*) – Exponent of the exponential taper of the emission surface.
- **z_func** (*Optional[function]*) – A function which provides $z(r)$. Note that no checking will occur to make sure this is a valid function.
- **force_positive_surface** (*Optional[bool]*) – Force the emission surface to be positive, default is `False`.
- **force_negative_surface** (*Optionanl[bool]*) – Force the emission surface to be negative, default is `False`.
- **frame** (*Optional[str]*) – Frame of reference for the returned coordinates. Either `'polar'` or `'cartesian'`.
- **shadowed** (*Optional[bool]*) – Use a slower, but more precise, method for deprojecting the pixel coordinates if the emission surface is shadowed.

Returns Three coordinate arrays, either the cylindrical coordaintes, `(r, theta, z)` or cartestian coordinates, `(x, y, z)`, depending on `frame`.

sky_to_disk(*coords*, *x0=0.0*, *y0=0.0*, *inc=0.0*, *PA=0.0*, *coord_type='cartesian'*, *coord_type_out='cartesian'*)

Deproject the sky plane coordinates into midplane disk-frame coordinates. Accounting for non-zero emission heights in progress.

Parameters

- **coords** (*tuple*) – A tuple of the sky-frame coordinates to transform. Must be either cartestian or cylindrical frames specified by the `coord_type` argument. All spatial coordinates should be given in [arcsec], while all angular coordinates should be given in [radians].
- **x0** (*Optional[float]*) – Source right ascension offset in [arcsec].
- **y0** (*Optional[float]*) – Source declination offset in [arcsec].
- **inc** (*float*) – Inclination of the disk in [deg].
- **PA** (*float*) – Position angle of the disk, measured Eastwards to the red-shifted major axis from North in [deg].
- **coord_type** (*Optional[str]*) – The coordinate type of the sky-frame coordinates, either `'cartesian'` or `'cylindrical'`.
- **coord_type_out** (*Optional[str]*) – The coordinate type of the returned disk-frame coordinates, either `'cartesian'` or `'cylindrical'`.

Returns Two arrays representing the deprojection of the input coordinates into the disk-frame, `x_disk` and `y_disk` if `coord_type_out='cartesian'` otherwise `r_disk` and `t_disk`.

disk_to_sky(*coords*, *inc*, *PA*, *x0=0.0*, *y0=0.0*, *coord_type='cylindrical'*, *return_idx=False*)
Project disk-frame coordinates onto the sky plane. Can return either the coordinates (the default return), useful for plotting, or the pixel indices (if `return_idx=True`) which can be used to extract a spectrum at a particular location.

Parameters

- **coords** (*tuple*) – A tuple of the disk-frame coordinates to transform. Must be either cartestian, cylindrical or spherical frames, specified by the `frame` argument. If only two coordinates are given, the input is assumed to be 2D. All spatial coordinates should be given in [arcsec], while all angular coordinates should be given in [radians].
- **inc** (*float*) – Inclination of the disk in [deg].
- **PA** (*float*) – Position angle of the disk, measured Eastwards to the red-shifted major axis from North in [deg].
- **x0** (*Optional[float]*) – Source right ascension offset in [arcsec].
- **y0** (*Optional[float]*) – Source declination offset in [arcsec].
- **coord_type** (*Optional[str]*) – Coordinate system used for the disk coordinates, either `'cartesian'`, `'cylindrical'` or `'spherical'`.
- **return_idx** (*Optional[bool]*) – If true, return the index of the nearest pixel to each on-sky position.

Returns Two arrays representing the projection of the input coordinates onto the sky, `x_sky` and `y_sky`, unless `return_idx` is `True`, in which case the arrays are the indices of the nearest pixels on the sky.

velocity_to_restframe_frequency(*velax=None*, *vlsr=0.0*)
Return restframe frequency [Hz] of the given velocity [m/s].

restframe_frequency_to_velocity(*nu*, *vlsr=0.0*)
Return velocity [m/s] of the given restframe frequency [Hz].

spectral_resolution(*dV=None*)
Convert velocity resolution in [m/s] to [Hz].

velocity_resolution(*dnu*)
Convert spectral resolution in [Hz] to [m/s].

keplerian_mask(*inc*, *PA*, *dist*, *mstar*, *vlsr*, *x0=0.0*, *y0=0.0*, *z0=0.0*, *psi=1.0*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *dV0=300.0*, *dVq=-0.5*, *r_min=0.0*, *r_max=4.0*, *nbeams=None*, *tolerance=0.01*, *restfreqs=None*, *max_dz0=0.2*, *return_type='float'*)
Generate a make based on a Keplerian velocity model. Original code from `https://github.com/richteague/keplerian_mask`. Unlike with the original code, the mask will be built on the same cube grid as the attached data. Multiple lines can be considered at once by providing a list of the rest frequencies of the line.

Unlike other functions, this does not accept `z_func`.

Parameters

- **inc** (*float*) – Inclination of the disk in [deg].
- **PA** (*float*) – Position angle of the disk, measured Eastwards to the red-shifted major axis from North in [deg].

- **dist** (*float*) – Distance to source in [pc].
- **mstar** (*float*) – Stellar mass in [Msun].
- **vlsr** (*float*) – Systemic velocity in [m/s].
- **x0** (*Optional[float]*) – Source right ascension offset in [arcsec].
- **y0** (*Optional[float]*) – Source declination offset in [arcsec].
- **z0** (*Optional[float]*) – Aspect ratio at 1" for the emission surface. To get the far side of the disk, make this number negative.
- **psi** (*Optional[float]*) – Flaring angle for the emission surface.
- **r_cavity** (*Optional[float]*) – Edge of the inner cavity for the emission surface in [arcsec].
- **r_taper** (*Optional[float]*) – Characteristic radius in [arcsec] of the exponential taper to the emission surface.
- **q_taper** (*Optional[float]*) – Exponent of the exponential taper of the emission surface.
- **dV0** (*Optional[float]*) – Line Doppler width at 1" in [m/s].
- **dVq** (*Optional[float]*) – Powerlaw exponent for the Doppler-width dependence.
- **r_min** (*Optional[float]*) – Inner radius to consider in [arcsec].
- **r_max** (*Optional[float]*) – Outer radius to consider in [arcsec].
- **nbeams** (*Optional[float]*) – Size of convolution kernel to smooth the mask by.
- **tolerance** (*Optional[float]*) – After smoothing, the limit used to decide if a pixel is masked or not. Lower values will include more pixels.
- **restfreqs** (*Optional[list]*) – Rest frequency (or list of rest frequencies) in [Hz] to allow for multiple (hyper-)fine components.
- **max_dz0** (*Optional[float]*) – The maximum step size between different `z0` values used for the different emission heights.
- **return_type** (*Optional[str]*) – The value type used for the returned mask, the default is `'float'`.

Returns The Keplerian mask with the desired value type.

Return type ndarry

print_beam()
Print the beam properties.

beams_per_pix
Number of beams per pixel.

pix_per_beam
Number of pixels in a beam.

FOV
Field of view.

frequency(*vlsr=0.0*, *unit='GHz'*)
A *velocity_to_restframe_frequency* wrapper with unit conversion.

Parameters

- **vlsr** (*optional[float]*) – Sytemic velocity in [m/s].
- **unit** (*optional[str]*) – Unit for the output axis.

Returns 1D array of frequency values.

frequency_offset(*nu0=None*, *vlsr=0.0*, *unit='MHz'*)
Return the frequency offset relative to *nu0* for easier plotting.

Parameters

- **nu0** (*optional[float]*) – Reference restframe frequency in [Hz].
- **vlsr** (*optional[float]*) – Sytemic velocity in [m/s].
- **unit** (*optional[str]*) – Unit for the output axis.

Returns 1D array of frequency values.

jybeam_to_Tb_RJ(*data=None*, *nu=None*)
[Jy/beam] to [K] conversion using Rayleigh-Jeans approximation.

jybeam_to_Tb(*data=None*, *nu=None*)
[Jy/beam] to [K] conversion using the full Planck law.

Tb_to_jybeam_RJ(*data=None*, *nu=None*)
[K] to [Jy/beam] conversion using Rayleigh-Jeans approxmation.

Tb_to_jybeam(*data=None*, *nu=None*)
[K] to [Jy/beam] conversion using the full Planck law.

estimate_RMS(*N=5*, *r_in=0.0*, *r_out=10000000000.0*)
Estimate RMS of the cube based on first and last *N* channels and a circular area described by an inner and outer radius.

Parameters

- **N** (*int*) – Number of edge channels to include.
- **r_in** (*float*) – Inner edge of pixels to consider in [arcsec].
- **r_out** (*float*) – Outer edge of pixels to consider in [arcsec].

Returns The RMS based on the requested pixel range.

Return type RMS (float)

print_RMS(*N=5*, *r_in=0.0*, *r_out=10000000000.0*)
Print the estimated RMS in Jy/beam and K (using RJ approx.).

correct_PB(*path*)
Correct for the primary beam given by `path`.

shift_image(*x0=0.0*, *y0=0.0*, *data=None*)
Shift the source center of the provided data by `d0` [arcsec] and `y0` [arcsec] in the x- and y-directions, respectively. The shifting is performed with `scipy.ndimage.shift` which uses a third-order spline interpolation.

Parameters

- **x0** (*Optional[float]*) – Shfit along the x-axis in [arcsec].
- **y0** (*Optional[float]*) – Shifta long the y-axis in [arcsec].
- **data** (*Optional[ndarray]*) – Data to shift.

Returns The shifted array.

Return type ndarray

rotate_image(*PA*, *data=None*)

Rotate the image such that the red-shifted axis aligns with the x-axis.

Parameters

- **PA** (*float*) – Position angle of the disk, measured to the red-shifted major axis of the disk, anti-clockwise from North, in [deg].
- **data** (*Optional[ndarray]*) – Data to rotate if not the attached data.

Returns The rotated array.

Return type ndarray

rms

RMS of the cube based on the first and last 5 channels.

extent

Cube field of view for use with Matplotlib's `imshow`.

get_spectrum(*coords*, *x0=0.0*, *y0=0.0*, *inc=0.0*, *PA=0.0*, *frame='sky'*, *coord_type='cartesian'*, *area=0.0*, *beam_weighting=False*, *return_mask=False*)

Return a spectrum at a position defined by a coordinates given either in sky-frame position (`frame='sky'`) or a disk-frame location (`frame='disk'`). The coordinates can be either in cartesian or cylindrical frames set by `coord_type`.

By default the returned spectrum is extracted at the pixel closest to the provided coordinates. If `area` is set to a positive value, then a beam-shaped area is averaged over, where `area` sets the size of this region in number of beams. For example `area=2.0` will result in an average over an area twice the size of the beam.

If an average is averaged over, you can also weight the pixels by the beam response with `beam_weighting=True`. This will reduce the weight of pixels that are further away from the beam center.

Finally, to check that you're extracting what you think you are, you can return the mask (and weights) used for the extraction with `return_mask=True`. Note that if `beam_weighting=False` then all `weights` will be 1.

TODO: Check that the returned uncertainties are reasonable.

Parameters

- **coords** (*tuple*) – The coordinates from where you want to extract a spectrum. Must be a length 2 tuple.
- **x0** (*Optional[float]*) – RA offset in [arcsec].
- **y0** (*Optional[float]*) – Dec offset in [arcsec].
- **inc** (*Optional[float]*) – Inclination of source in [deg]. Only required for `frame='disk'`.
- **PA** (*Optional[float]*) – Position angle of source in [deg]. Only required for `frame='disk'`.
- **frame** (*Optional[str]*) – The frame that the `coords` are given. Either `'disk'` or `'sky'`.
- **coord_type** (*Optional[str]*) – The type of coordinates given, either `'cartesian'` or `'cylindrical'`.

- **area** (*Optional[float]*) – The area to average over in units of the beam area. Note that this take into account the beam aspect ratio and position angle. For a single pixel extraction use `area=0.0`.
- **beam_weighting** (*Optional[bool]*) – Whether to use the beam response function to weight the averaging of the spectrum.
- **return_mask** (*Optional[bool]*) – Whether to return the mask and weights used to extract the spectrum.

Retuns (if `return_mask=False`): x, y, dy (arrays): The velocity axis, extracted spectrum and associated uncertainties.

(if `return_mask=True`): mask, weights (arrays): Arrays of the mask used to extract the spectrum and the weighted used for the averaging.

convolve_with_beam(*data*, *scale=1.0*, *circular=False*, *convolve_kwargs=None*)

Convolve the attached data with a 2D Gaussian kernel matching the synthesized beam. This can be scaled with `scale`, or forced to be circular (taking the major axis as the radius of the beam).

Parameters

- **data** (*ndarray*) – The data to convolve. Must be either 2D or 3D.
- **scale** (*Optional[float]*) – Factor to scale the synthesized beam by.
- **circular** (*Optional[bool]*) – Force a cicular kernel. If `True1`, the kernel will adopt the scaled major axis of the beam to use as the radius.
- **convolve_kwargs** (*Optional[dict]*) – Keyword arguments to pass to `astropy.convolution.convolve`.

Returns Data convolved with the requested kernel.

Return type ndarray

cross_section(*x0=0.0*, *y0=0.0*, *PA=0.0*, *mstar=1.0*, *dist=100.0*, *vlsr=None*, *grid=True*, *grid_spacing=None*, *downsample=1*, *cylindrical_rotation=False*, *clip_noise=True*, *min_npnts=5*, *statistic='mean'*, *mask_velocities=None*)

Return the cross section of the data following Dutrey et al. (2017). This yields `I_nu(r, z)`. If `grid=True` then this will be gridded using `scipy.interpolate.griddata` onto axes with the same pixel spacing as the attached data.

Reference:

Dutrey et al. (2017): https://ui.adsabs.harvard.edu/abs/2017A%26A...607A.130D

Parameters

- **x0** (*Optional[float]*) – Source right ascension offset [arcsec].
- **y0** (*Optional[float]*) – Source declination offset [arcsec].
- **PA** (*Optional[float]*) – Position angle of the disk in [deg].
- **mstar** (*Optional[float]*) – Mass of the central star in [Msun].
- **dist** (*Optional[float]*) – Distance to the source in [pc].
- **vlsr** (*Optional[float]*) – Systemic velocity in [m/s]. If `None`, assumes the central velocity.
- **grid** (*Optional[bool]*) – Whether to grid the coordinates to a regular grid. Default is `True`.

- **grid_spacing** (*Optional[float]*) – The spacing, in [arcsec], for the R and Z grids. If `None` is provided, will use pixel spacing.
- **downsample** (*Optional[int]*) – If provided, downsample the coordinates to grid by this factor to speed up the interpolation for large datasets. Default is `1`.
- **cylindrical_rotation** (*Optional[bool]*) – If `True`, assume that the Keplerian rotation decreases with height above the midplane.
- **clip_noise** (*Optional[bool]*) – If `True`, remove all pixels which fall below 3 times the standard deviation of the two edge channels. If the argument is a `float`, use this as the clip level.
- **min_npnts** (*Optional[int]*) – Number of minimum points in each bin for the average. Default is 5.
- **statistic** (*Optional[str]*) – Statistic to calculate for each bin. Note that the uncertainty returned will only make sense with `'max'`, `'mean'` or `'median'`.
- **mask_velocities** (*Optional[list of tuples]*) – List of `(v_min, v_max)` tuples to mask (i.e. remove from the averaging).

Returns Either two 1D arrays containing `(r, z, I_nu)`, or, if `grid=True`, two 1D arrays with the `r` and `z` axes and two 2D array of `I_nu` and `dI_nu`.

Return type ndarray

spiral_coords(*r_p*, *t_p*, *m=None*, *r_min=None*, *r_max=None*, *mstar=1.0*, *T0=20.0*, *Tq=-0.5*, *dist=100.0*, *clockwise=True*, *frame_out='cartesian'*)

Spiral coordinates from Bae & Zhaohuan (2018a). In order to recover the linear spirals from Rafikov (2002), use m >> 1. :param r_p: Orbital radius of the planet in [arcsec]. :type r_p: float :param t_p: Polar angle of planet relative to the red-shifted

major axis of the disk in [radians].

Parameters

- **m** (*optional[int]*) – Azimuthal wavenumber of the spiral. If not specified, will assume the dominant term based on the rotation and temperature profiles.
- **r_min** (*optional[float]*) – Inner radius of the spiral in [arcsec].
- **r_max** (*optional[float]*) – Outer radius of the spiral in [arcsec].
- **mstar** (*optioanl[float]*) – Stellar mass of the central star in [Msun] to calculate the rotation profile.
- **T0** (*optional[float]*) – Gas temperature in [K] at 1 arcsec.
- **Tq** (*optional[float]*) – Exoponent of the radial gas temperature profile.
- **dist** (*optional[float]*) – Source distance in [pc] used to scale [arcsec] to [au] in the calculation of the rotation profile.
- **clockwise** (*optional[bool]*) – Direction of the spiral.
- **frame_out** (*optional[str]*) – Coordinate frame of the returned values, either 'cartesian' or 'cylindrical'.

Returns Coordinates of the spiral in either cartestian or cylindrical frame.

Return type ndarray

plot_center(*x0s*, *y0s*, *SNR*, *normalize=True*)
Plot the array of SNR values.

plot_teardrop(*inc*, *PA*, *mstar*, *dist*, *ax=None*, *rvals=None*, *rbins=None*, *dr=None*, *x0=0.0*, *y0=0.0*, *z0=None*, *psi=None*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *z_func=None*, *resample=1*, *beam_spacing=False*, *r_min=None*, *r_max=None*, *PA_min=None*, *PA_max=None*, *exclude_PA=False*, *abs_PA=False*, *mask_frame='disk'*, *mask=None*, *unit='Jy/beam'*, *pcolormesh_kwargs=None*, *shadowed=False*, *vrad_func=None*)
Make a *teardrop* plot. For argument descriptions see `radial_spectra`. For all properties related to `pcolormesh`, include them in `pcolormesh_kwargs` as a dictionary, e.g.

pcolormesh_kwargs = dict(cmap='inferno', vmin=0.0, vmax=1.0)

This will override any of the default style parameters.

plot_beam(*ax*, *x0=0.1*, *y0=0.1*, ***kwargs*)
Plot the sythensized beam on the provided axes.

Parameters

- **ax** (*matplotlib axes instance*) – Axes to plot the FWHM.
- **x0** (*float*) – Relative x-location of the marker.
- **y0** (*float*) – Relative y-location of the marker.
- **kwargs** (*dic*) – Additional kwargs for the style of the plotting.

plot_surface(*x0=0.0*, *y0=0.0*, *inc=0.0*, *PA=0.0*, *z0=None*, *psi=None*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *z_func=None*, *shadowed=False*, *r_max=None*, *fill=None*, *ax=None*, *contour_kwargs=None*, *imshow_kwargs=None*, *return_fig=True*)
Overplot the assumed emission surface.

Parameters

- **x0** (*Optional[float]*) – Source right ascension offset [arcsec].
- **y0** (*Optional[float]*) – Source declination offset [arcsec].
- **inc** (*Optional[float]*) – Source inclination [deg].
- **PA** (*Optional[float]*) – Source position angle [deg]. Measured between north and the red-shifted semi-major axis in an easterly direction.
- **z0** (*Optional[float]*) – Aspect ratio at 1" for the emission surface. To get the far side of the disk, make this number negative.
- **psi** (*Optional[float]*) – Flaring angle for the emission surface.
- **r_cavity** (*Optional[float]*) – Edge of the inner cavity for the emission surface in [arcsec].
- **r_taper** (*Optional[float]*) – Characteristic radius in [arcsec] of the exponential taper to the emission surface.
- **q_taper** (*Optional[float]*) – Exponent of the exponential taper of the emission surface.
- **z_func** (*Optional[function]*) – A function which provides $z(r)$. Note that no checking will occur to make sure this is a valid function.
- **shadowed** (*Optional[bool]*) – If `True`, use the slower, but more robust method for deprojecting pixel values.

- **r_max** (*Optional[float]*) – Maximum radius in [arcsec] to plot the emission surface out to.
- **fill** (*Optional[str]*) – A string to execute (be careful!) to fill in the emission surface using `rvals`, `tvals` and `zvals` as returned by `disk_coords`. For example, to plot the radial values use `fill='rvals'`. To plot the projection of rotational velocities, use `fill='rvals * np.cos(tvals)'`.
- **ax** (*Optional[matplotlib axis instance]*) – Axis to plot onto. contour_kwargs (Optional[dict]): Kwargs to pass to `matplolib.contour` to overplot the mesh.
- **return_fig** (*Optional[bool]*) – If `True`, return the figure for additional plotting.

Returns The `ax` instance.

plot_maximum(*ax=None*, *imshow_kwargs=None*)

Plot the maximum along the spectral axis.

Parameters

- **ax** (*Optional[matplotlib axis instance]*) – Axis to use for plotting.
- **imshow_kwargs** (*Optional[dict]*) – Kwargs to pass to imshow.

Returns The axis with the maximum plotted.

plot_mask(*ax*, *r_min=None*, *r_max=None*, *exclude_r=False*, *PA_min=None*, *PA_max=None*, *exclude_PA=False*, *abs_PA=False*, *mask_frame='disk'*, *mask=None*, *x0=0.0*, *y0=0.0*, *inc=0.0*, *PA=0.0*, *z0=None*, *psi=None*, *r_cavity=None*, *r_taper=None*, *q_taper=None*, *z_func=None*, *mask_color='k'*, *mask_alpha=0.5*, *contour_kwargs=None*, *contourf_kwargs=None*, *shadowed=False*)

Plot the boolean mask on the provided axis to check that it makes sense.

Parameters

- **ax** (*matplotib axis instance*) – Axis to plot the mask.
- **r_min** (*Optional[float]*) – Minimum midplane radius of the annulus in [arcsec]. Defaults to minimum deprojected radius.
- **r_max** (*Optional[float]*) – Maximum midplane radius of the annulus in [arcsec]. Defaults to the maximum deprojected radius.
- **exclude_r** (*Optional[bool]*) – If `True`, exclude the provided radial range rather than include.
- **PA_min** (*Optional[float]*) – Minimum polar angle of the segment of the annulus in [degrees]. Note this is the polar angle, not the position angle.
- **PA_max** (*Optional[float]*) – Maximum polar angle of the segment of the annulus in [degrees]. Note this is the polar angle, not the position angle.
- **exclude_PA** (*Optional[bool]*) – If `True`, exclude the provided polar angle range rather than include it.
- **abs_PA** (*Optional[bool]*) – If `True`, take the absolute value of the polar angle such that it runs from 0 [deg] to 180 [deg].
- **x0** (*Optional[float]*) – Source center offset along the x-axis in [arcsec].
- **y0** (*Optional[float]*) – Source center offset along the y-axis in [arcsec].
- **inc** (*Optional[float]*) – Inclination of the disk in [degrees].

- **PA** (*Optional[float]*) – Position angle of the disk in [degrees], measured east-of-north towards the redshifted major axis.
- **z0** (*Optional[float]*) – Emission height in [arcsec] at a radius of 1".
- **psi** (*Optional[float]*) – Flaring angle of the emission surface.
- **z_func** (*Optional[function]*) – A function which provides $z(r)$. Note that no checking will occur to make sure this is a valid function.
- **mask_color** (*Optional[str]*) – Color used for the mask lines.
- **mask_alpha** (*Optional[float]*) – The alpha value of the filled contour of the masked regions. Setting `mask_alpha=0.0` will remove the filling.
- **contour_kwargs** (*Optional[dict]*) – Kwargs to pass to contour for drawing the mask.

Returns The matplotlib axis instance.

Return type ax

5.2 Coordinate System in GoFish

5.2.1 0. Get the Data

For this we will use the CS (3-2) emission in TW Hya, originally published in Teague et al. (2018). The emission is already clearly detected, but acts as a good starting point. You can download the data from Harvard Dataverse, making sure to place it in the directory where this notebook is.

[1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from gofish import imagecube
```

[2]:
```
cube = imagecube('TWHya_CS_32.fits', FOV=10.0)
```

5.2.2 1. Definitions

Within `GoFish` we use the inclination, i, and position angle, PA, to define two *rotations* which transform a disk coordinate system, $(x_{\rm disk}, y_{\rm disk}, z_{\rm disk})$, into the on-sky coordinates $(x_{\rm sky}, y_{\rm sky})$. Note that because we only see things projected on the sky, the transformation from the disk-frame to the sky-frame is simple, while the return transformation is trickier if we know the disk structure is 3D. This is discussed more later on.

To begin with, the inclination of a disk is defined as the inverse cosine of the ratio of the major and minor axes of the disk, $i = \cos^{-1}(a_{\rm maj} \,/\, a_{\rm min})$. You can think of this as a rotation around the major axis of the disk. The position angle is defined as the angle from north to the *red-shifted* major axis of the disk in an easterly (counter-clockwise) direction. You can think about this as a rotation around the line-of-sight axis in an anticlockwise direction. Note that some literature defintions for PA can vary, for example taking the closest major axis such that PA $\in [0, \pi)$, so make sure to check the definitions!

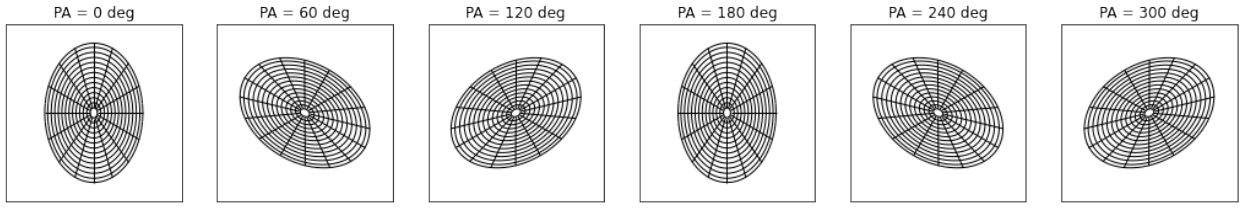
There are two main functions within `GoFish` that will help with transforming between disk- and sky reference frames, `disk_coords`, which calculates the disk-frame coordinates for each pixel for a given viewing geometry, and `disk_to_sky`, which projects the disk-frame coordinates into on-sky positions.

1.1 A 2D Example

As an example, we can see how the disk structure varies for different $(i,\ \mathrm{PA})$ values. For a given viewing geometry, the function `disk_coords` will return three arrays representing the cylindrical disk-frame coordinates corresponding to each on-sky pixel. To demonstrate how this works, we can use the `plot_surface` function within `GoFish` to quickly see how the projected disk geometry looks like.

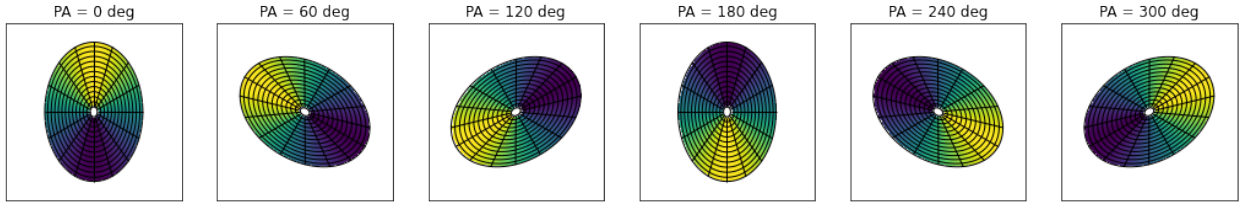
Holding $i = 45°$ and varying PA, we can see how the disk rotates.

[3]:
```
fig, axs = plt.subplots(ncols=6, figsize=(18, 3))

for a, ax in enumerate(axs):

    inc = 45.0
    PA = a * 60.0

    cube.plot_surface(inc=inc, PA=PA, r_max=4.0, ax=ax)
    ax.tick_params(left=0, bottom=0)
    ax.set_xticklabels([])
    ax.set_yticklabels([])

    ax.set_title('PA = {:.0f} deg'.format(PA))
```



The difference between say PA $= 60°$ and PA $= 240°$ can be more easily seen when we fill in the plots with the polar angle, θ. Within `GoFish` this is usually called `tvals`, which runs from $-\pi$ to π, with $\theta = 0$ along the red-shfited major axis. We can add a line into the above command to fill in the plots with $\cos(\theta)$ to highlight the red- and blue-shifted sides of the disk.

[4]:
```
fig, axs = plt.subplots(ncols=6, figsize=(18, 3))

for a, ax in enumerate(axs):

    inc = 45.0
    PA = a * 60.0

    cube.plot_surface(inc=inc, PA=PA, r_max=4.0, fill='np.cos(tvals)', ax=ax)
    ax.tick_params(left=0, bottom=0)
    ax.set_xticklabels([])
    ax.set_yticklabels([])

    ax.set_title('PA = {:.0f} deg'.format(PA))
```



5.2.3 2. 3D Disks

With the high spatial resolution afford by ALMA, it is now possible to resolve the vertical structure of the disk as well (see Rosenfeld et al. 2013 for a nice demonstration of this for HD 163296). Given that we have access to this information, we want to use that in our model too.

Given that we are often just concerned with a thin emission region within a 3D disk, in `GoFish` we describe a 3D structure as a 2D surface within that 3D space, defined in disk-centric cylindrical coordinates as,

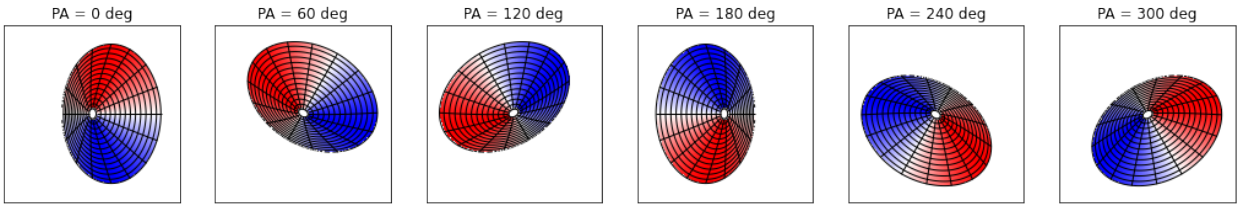
$$z(r) = z_0 \times \left(\frac{r - r_{\rm cavity}}{1''}\right)^{\psi} \times \exp\left(-\left[\frac{r - r_{\rm cavity}}{r_{\rm taper}}\right]^{q_{\rm taper}}\right)$$

where the second term acts as a 'correction' term in order to account for the drop in emission surface expected due to the drop in gas surface density at large radii. `r_cavity` can be used to describe an inner cavity, for example as in PDS 70. By default $r_{\rm cavity} = 0''$ and $r_{\rm taper} = \infty$, such that by including a $(z_0,\ \psi)$ pair, we can start to build concical ($\psi = 1$) or flared ($\psi > 1$) emission surfaces.

5.2.4 2.1 A Simple 3D Example

Using the above code, we can now consider a slightly flared disk with $(z_0,\ \psi) = (0.3,\ 1.2)$.

[5]:
```
fig, axs = plt.subplots(ncols=6, figsize=(18, 3))

for a, ax in enumerate(axs):

    inc = 45.0
    PA = a * 60.0

    cube.plot_surface(inc=inc, PA=PA, z0=0.3, psi=1.2, r_max=4.0, ax=ax,
                      imshow_kwargs=dict(cmap='bwr'), fill='np.cos(tvals)')
    ax.tick_params(left=0, bottom=0)
    ax.set_xticklabels([])
    ax.set_yticklabels([])

    ax.set_title('PA = {:.0f} deg'.format(PA))
```



It is quite clear here that depending on the viewing geometry, one side of the disk appears shorter than the other. This is because the elevated emission surface contributed to the perceived inclination of one side of the disk, such that along the minor axis of the disk,

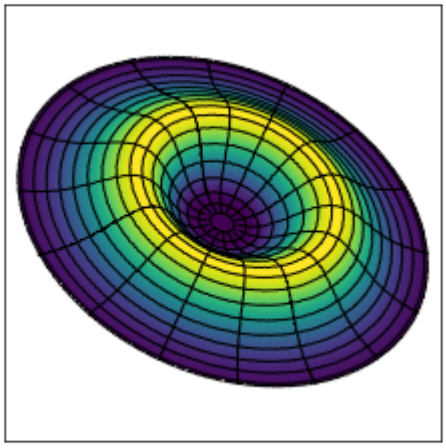
$$i_{\rm perceived} \approx i_{\rm surface} \pm i_{\rm disk} = \tan^{-1}(z\,/\,r) \pm i_{\rm disk},$$

where the side *closer* to the observer has an *increased* inclination, while the side *further* from the observer has a *reduced* inclination.

5.2.5 2.2 A Complex 3D Example

When one has very good angular resolution, it is possible to use a more complex emission surface.

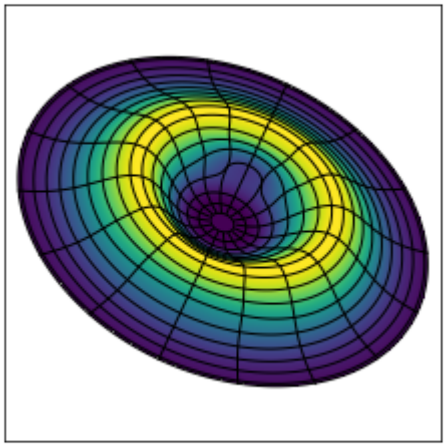
[6]:
```
fig = cube.plot_surface(inc=45.0, PA=65.0,
                        z0=0.8, psi=2.5,
                        r_taper=1.7, q_taper=2.0,
                        r_cavity=0.5, r_max=5.0,
                        fill='zvals',
                        shadowed=True)

fig.axes[0].set_xticklabels([])
fig.axes[0].set_yticklabels([])
fig.axes[0].tick_params(left=0, bottom=0)
```



Note here that we have used the `shadowed=True` argument. This is because bits of the disk are 'shadowed' or hidden along the line of sight. Typically `GoFish` will calculate the sky-to-disk transform iteratively which works well for simple emission surfaces and is fast. When more complex emission surfaces are needed, it can employ a more robust approach which builds a 3D model which it then rotates given the viewing geometry, and then projects that onto the sky. Any function which requires some transform should accept the `shadowed` argument.

To demonstrate the difference, we can turn this off.

[7]:
```
fig = cube.plot_surface(inc=45.0, PA=65.0,
                        z0=0.8, psi=2.5,
                        r_taper=1.7, q_taper=2.0,
                        r_cavity=0.5, r_max=5.0,
                        fill='zvals',
                        shadowed=False)
fig.axes[0].set_xticklabels([])
fig.axes[0].set_yticklabels([])
fig.axes[0].tick_params(left=0, bottom=0)
```



Close, but not quite right...

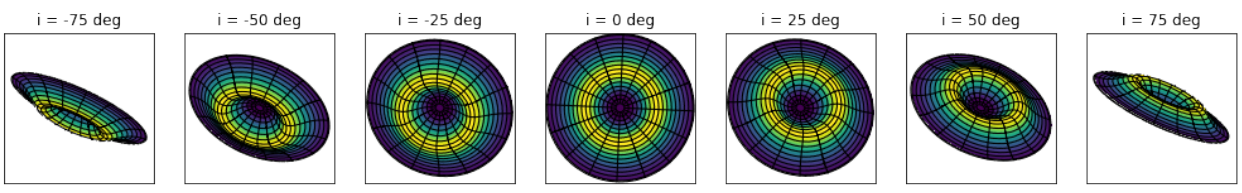
5.2.6 3. Disk Rotation And Total Orientation

Now that it is possible to resolve separate sides of the disk, we also need to consider the complete orientation of the disk. Consider a face-on disk. If we were to incline this disk (apply a rotation around the major axis), then we can rotate it in two directions: in one case the northern edge moves *towards* the observer, in the other, the southern edge does. For a 2D disk, these two are identical and this is why generally literature values of disks spane $i \in [0°, 90°)$.

3.1 Negative Inclinations

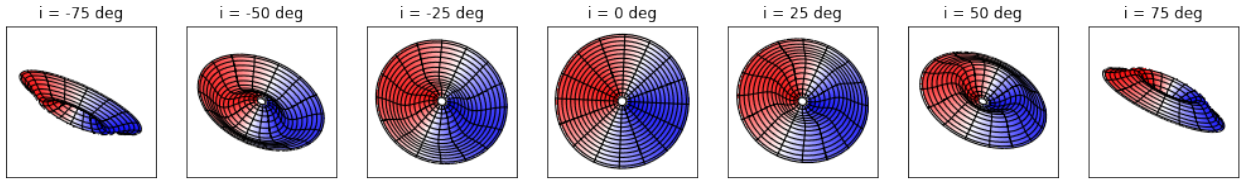
However, for us it makes a difference which direction that rotation is. This can be changed by considering $i \in (-90°, 90°)$. This allows for the front surface to be completely specified.

[8]:
```
fig, axs = plt.subplots(ncols=7, figsize=(17.5, 2.5))

for a, ax in enumerate(axs):

    inc = np.linspace(-75, 75, len(axs))[a]

    fig = cube.plot_surface(inc=inc, PA=65.0,
                            z0=0.8, psi=2.5,
                            r_taper=1.7, q_taper=2.0,
                            r_cavity=0.5, r_max=5.0,
                            fill='zvals', ax=ax,
                            shadowed=True)
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    ax.tick_params(left=0, bottom=0)
    ax.set_title('i = {:.0f} deg'.format(inc))
```



With this somewhat extreme vertical structure, this highlights the issue of rotation. Making the same plot, but filling the background with a pseudo-velocity structure projected along the line of sight:

[9]:
```
fig, axs = plt.subplots(ncols=7, figsize=(17.5, 2.5))

for a, ax in enumerate(axs):

    inc = np.linspace(-75, 75, len(axs))[a]

    fig = cube.plot_surface(inc=inc, PA=65.0,
                            z0=0.8, psi=2.5,
                            r_taper=1.7, q_taper=2.0,
                            r_cavity=0.5, r_max=4.5,
                            fill='rvals**-0.1 * np.cos(tvals)',
                            imshow_kwargs=dict(cmap='bwr'),
                            ax=ax, shadowed=True)
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    ax.tick_params(left=0, bottom=0)
    ax.set_title('i = {:.0f} deg'.format(inc))
```



This makes it clear that $i > 0$ represents clockwise rotation, while $i < 0$ is anti-clockwise rotation. In the extreme case of $i = \pm 90°$, you can imagine that these are opposite sides of the disk.

3.2 Bottom Side of the Disk

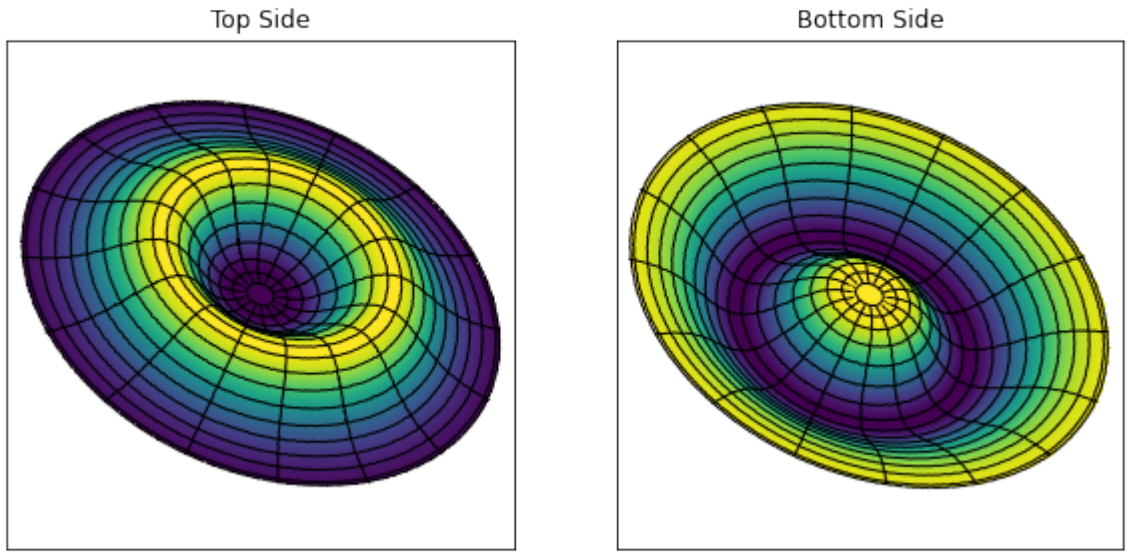
You can also access the bottom side of the disk by flipping the sign of z_0,

[10]:
```
fig, axs = plt.subplots(ncols=2, figsize=(10, 5))

fig = cube.plot_surface(inc=45.0, PA=65.0,
                        z0=0.8, psi=2.5,
                        r_taper=1.7, q_taper=2.0,
                        r_cavity=0.5, r_max=5.0,
                        fill='zvals', ax=axs[0],
                        shadowed=True)
axs[0].set_xticklabels([])
axs[0].set_yticklabels([])
axs[0].set_title('Top Side')
axs[0].tick_params(left=0, bottom=0)

fig = cube.plot_surface(inc=45.0, PA=65.0,
                        z0=-0.8, psi=2.5,
                        r_taper=1.7, q_taper=2.0,
                        r_cavity=0.5, r_max=5.0,
                        fill='zvals', ax=axs[1],
                        shadowed=True)
axs[1].set_xticklabels([])
axs[1].set_yticklabels([])
```

(continues on next page)

(continued from previous page)

```
axs[1].set_title('Bottom Side')
axs[1].tick_params(left=0, bottom=0)
```



NOTE - There's currently an issue in the code that for highly inclined sources with large $z(r)$ profiles the front side of the disk may be shadowed incorrectly. A fix in underway.

5.2.7 4. User-Defined Surfaces

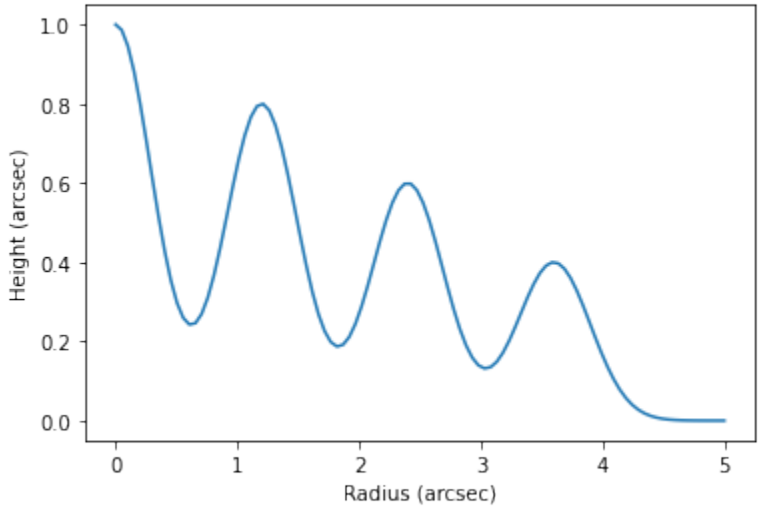
Sometimes it would be useful to use an emperically derived emission surface (following Pinte et al., 2018, for example). This can be achived by providing `GoFish` with a function, `zfunc`, which returns the emission height in arcseconds for a given midplane radius in arcseconds.

4.1 Example of a User-Defined Surface

First define an emission surface with some bumps and wiggles.

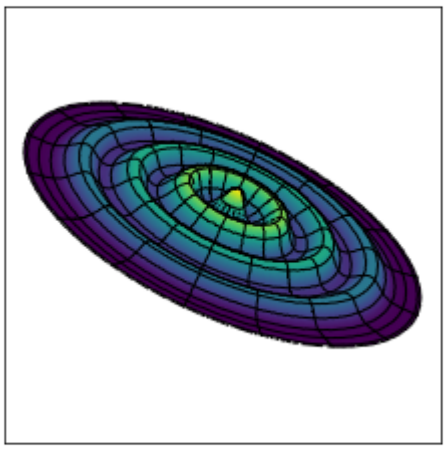
[11]:
```
def z_func(r):
    z = np.zeros(r.shape)
    for i in range(4):
        z += (1.0 - i * 0.2) * np.exp(-0.5 * ((r - i * 1.2) / 0.3)**2.0)
    return np.clip(z, a_min=0.0, a_max=None)

r = np.linspace(0, 5, 100)
z = z_func(r)

fig, ax = plt.subplots()
ax.plot(r, z)
ax.set_xlabel('Radius (arcsec)')
ax.set_ylabel('Height (arcsec)')
```

[11]:
```
Text(0, 0.5, 'Height (arcsec)')
```



Then provide this to `disk_coords`, `disk-to-sky` or `plot_surface`.

[12]:
```
fig = cube.plot_surface(inc=65.0, PA=65.0, z_func=z_func,
                        r_max=5.0, fill='zvals', shadowed=True)
fig.axes[0].set_xticklabels([])
fig.axes[0].set_yticklabels([])
fig.axes[0].tick_params(left=0, bottom=0)
```



5.2.8 5. Disk-to-Sky Transformations

Sometimes it is useful to project things defined in the disk-frame onto the sky. For this we can use the `disk_to_sky` transformation.

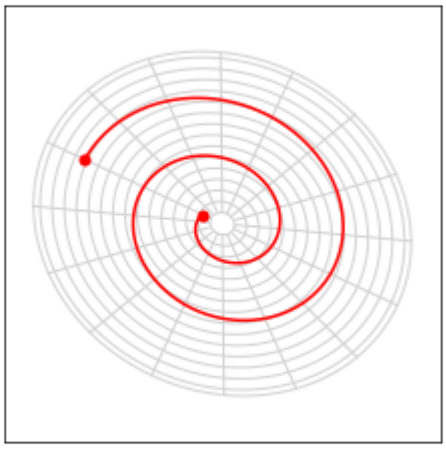
5.1 Midplane Spiral

[13]:
```
#define the viewing geometry

inc = 30.0
PA = 65.0
```

(continues on next page)

(continued from previous page)

```
# draw a background mesh
fig = cube.plot_surface(inc=inc, PA=PA, r_max=4.5,
                        contour_kwargs=dict(colors='lightgray', zorder=-10))
ax = fig.axes[0]

# define a midplane spiral in (arcsec, radians)

r = np.linspace(0.5, 3.5, 1000)
t = np.linspace(0, 4 * np.pi, r.size)

# calculate the transform

x_sky, y_sky = cube.disk_to_sky((r, t), inc=inc, PA=PA, coord_type='cylindrical')

# plot

ax.plot(x_sky, y_sky, color='r')
ax.scatter(x_sky[0], y_sky[0], color='r', lw=0.0)
ax.scatter(x_sky[-1], y_sky[-1], color='r', lw=0.0)

ax.set_xticklabels([])
ax.set_yticklabels([])
ax.tick_params(left=0, bottom=0)
```



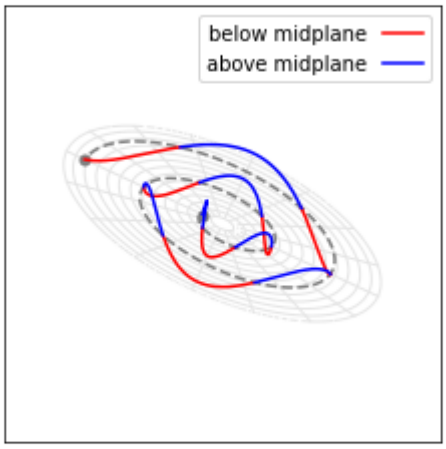
5.2 Buoyant Spiral

[14]:
```
#define the viewing geometry

inc = 65.0
PA = 65.0

# draw a background mesh
fig = cube.plot_surface(inc=inc, PA=PA, r_max=4.0,
                        contour_kwargs=dict(colors='0.9', zorder=-10))
ax = fig.axes[0]

# define a midplane spiral in (arcsec, radians)
```

(continues on next page)

(continued from previous page)

```
r = np.linspace(0.5, 3.5, 10000)
t = np.linspace(0, 4 * np.pi, r.size)
z = 0.5 * np.sin(3.0 * t)

# calculate the transform for the midplane spiral

x_sky, y_sky = cube.disk_to_sky((r, t), inc=inc, PA=PA, coord_type='cylindrical')
l = ax.plot(x_sky, y_sky, color='gray', ls='--')
ax.scatter(x_sky[0], y_sky[0], color=l[0].get_color(), lw=0.0)
ax.scatter(x_sky[-1], y_sky[-1], color=l[0].get_color(), lw=0.0)

# overplot the buoyant spiral

x_sky, y_sky = cube.disk_to_sky((r, t, z), inc=inc, PA=PA, coord_type='cylindrical')
l = ax.plot(np.where(z < 0, x_sky, np.nan), np.where(z < 0, y_sky, np.nan),
            color='red', label='below midplane')

x_sky, y_sky = cube.disk_to_sky((r, t, z), inc=inc, PA=PA, coord_type='cylindrical')
l = ax.plot(np.where(z > 0, x_sky, np.nan), np.where(z > 0, y_sky, np.nan),
            color='blue', label='above midplane')

ax.legend(markerfirst=False)
ax.set_xticklabels([])
ax.set_yticklabels([])
ax.tick_params(left=0, bottom=0)
```



5.3 Basics of `GoFish`

This Notebook will walk through the very basic steps of how to load up data, set up the rotation profile and start extracting spectra.

5.3.1 Get the Data

For this we will use the CS (3-2) emission in TW Hya, originally published in Teague et al. (2018). The emission is already clearly detected, but acts as a good starting point to learn how to use `GoFish`. You can download the data

from Harvard Dataverse, making sure to place it in the directory where this notebook is, or automatically with the cell below.

[1]:
```
import os
if not os.path.exists('TWHya_CS_32.fits'):
    !wget -O TWHya_CS_32.fits -q https://dataverse.harvard.edu/api/access/datafile/:
↪persistentId?persistentId=doi:10.7910/DVN/LO2QZM/KYZFUQ
```

5.3.2 Load the Data

Loading the data is as simple as firing up `GoFish` and passing it the path of the fits cube to `imagecube`.

[2]:
```
import matplotlib.pyplot as plt
from gofish import imagecube
import numpy as np
```

[3]:
```
cube = imagecube('TWHya_CS_32.fits', FOV=10.0)
```

You can use the `FOV` argument here to trim down your image cube to a field of view which may speed up calculations.

5.3.3 Coordinate System

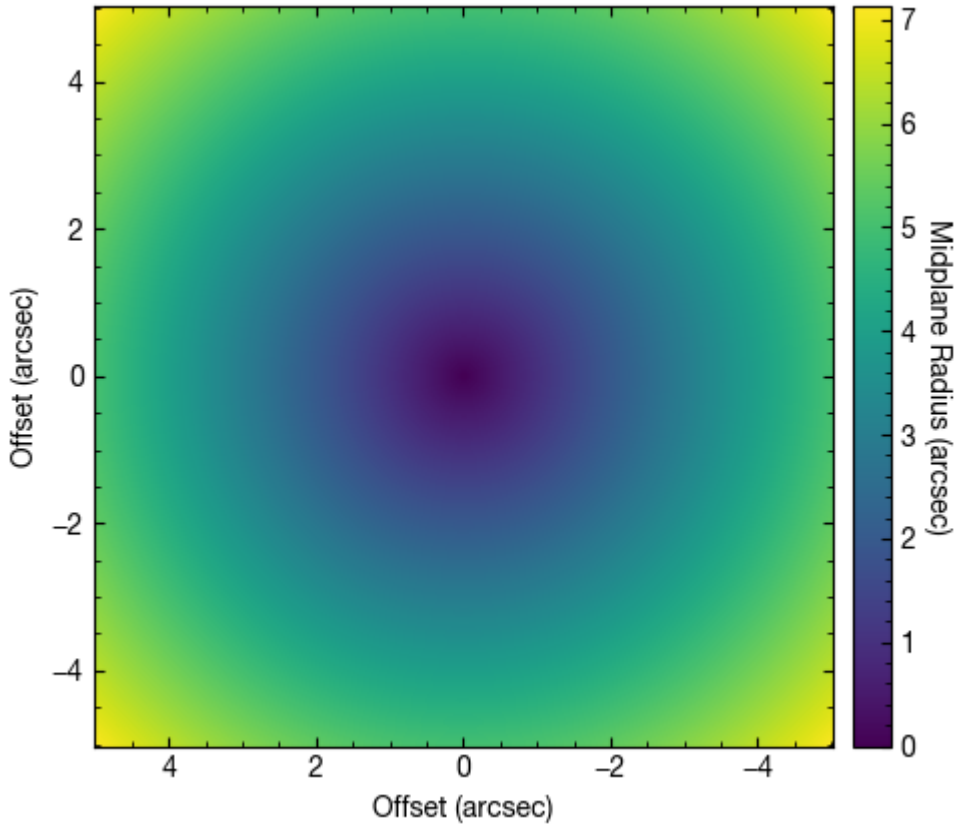
We can deproject from a pixel position, $(x_{\rm pix}, y_{\rm pix})$, to a disk-frame coordinate, (r, ϕ). These transforms are done through the `disk_coords` function which takes geometrical properties of the disk and returns the disk coordinates.

2D Disks

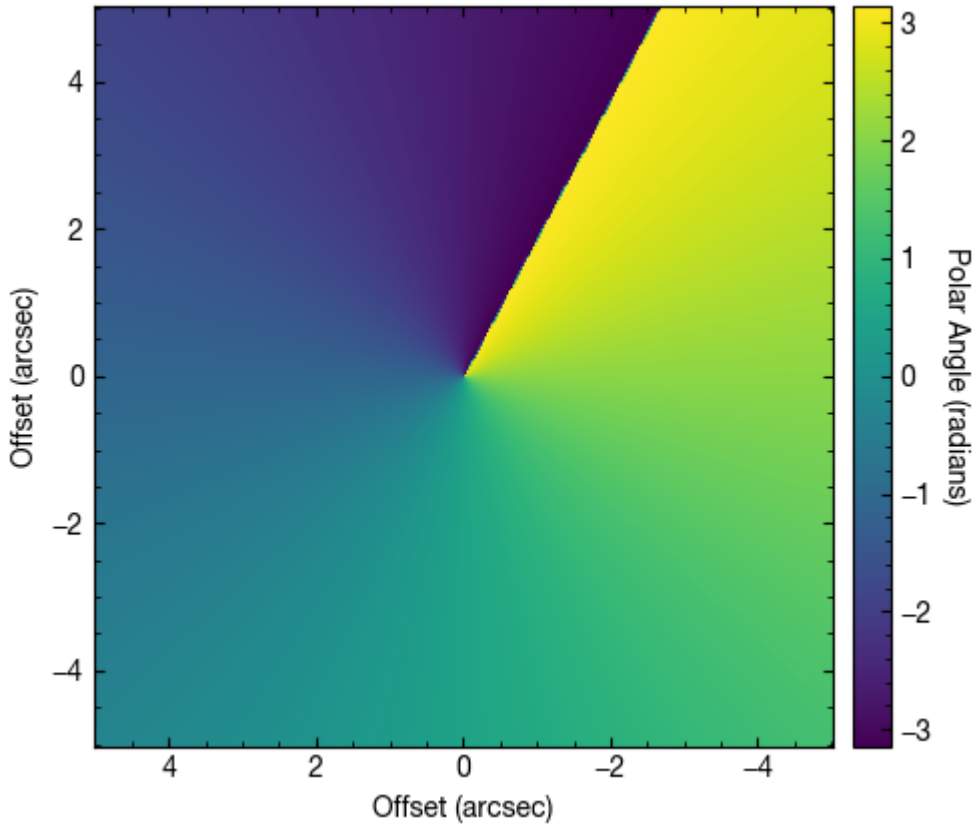
For TW Hya we know that `i=5.0` and `PA=152.0` (both in degrees), so we can return the (r, ϕ) values for each pixel. Note that in `GoFish`, position angle is defined as the angle between North and the *red-shifted* disk major axis in an eastwards (anti-clockwise) direciton. We also assume the source is centered in the image such that `x0=0.0` and `y0=0.0`.

[4]:
```
rvals, tvals, _ = cube.disk_coords(x0=0.0, y0=0.0, inc=5.0, PA=152.0)
```

[5]:
```
fig, ax = plt.subplots()
im = ax.imshow(rvals, origin='lower', extent=cube.extent, vmin=0.0)
cb = plt.colorbar(im, pad=0.02)
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
cb.set_label('Midplane Radius (arcsec)', rotation=270, labelpad=13)
```



[6]:
```
fig, ax = plt.subplots()
im = ax.imshow(tvals, origin='lower', extent=cube.extent)
cb = plt.colorbar(im, pad=0.02)
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
cb.set_label('Polar Angle (radians)', rotation=270, labelpad=13)
```



The polar angle, θ, in is radians, and runs from $-\pi$ to π in an eastward direction with $\theta = 0$ aligning with the red-shifted major axis of the disk. Note that this is *not* the same as the position angle, particualrly when the disk is highly inclined.

3D Disks

You'll notice that when calling `disk_coords` we left space for three returned parameters. The third is z, the height above the midplane. By default, we assume that the disk is a razor-thin 2D disk. However, with the high spatial resolution afford by ALMA, it is now possible to resolve the vertical structure of the disk as well (see Rosenfeld et al. 2013 for a nice demonstration of this for HD 163296).

For this, we assume that the emission surface is described via,

$$z(r) = z_0 \times \left(\frac{r - r_{\rm cavity}}{1''}\right)^{\psi} \times \exp\left(-\left[\frac{r - r_{\rm cavity}}{r_{\rm taper}}\right]^{q_{\rm taper}}\right)$$

where the second term acts as a 'correction' term in order to account for the drop in emission surface expected due to the drop in gas surface density at large radii. `r_cavity` can be used to describe an inner cavity, for example as in PDS 70.

As an example, we can model a conical disk with $\psi = 1$ (taking $\psi > 1$ will give a flared emission surface). We increase the inclination to make the changes due to the height more noticable. Here we use the `plot_surface` function to have an idea of what the surface looks like.

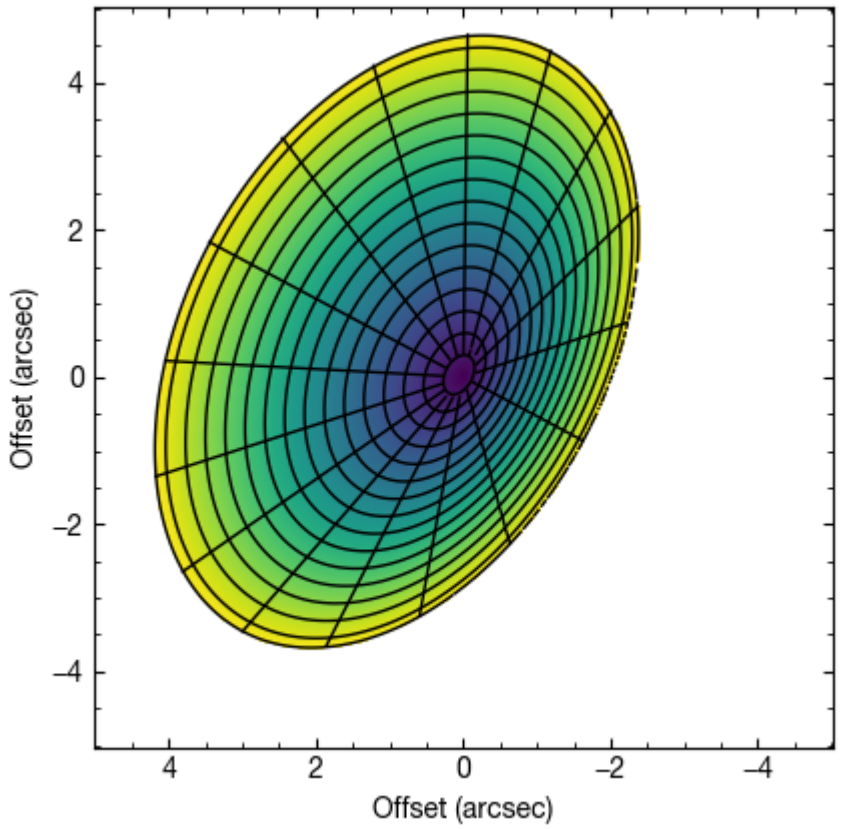
[7]:
```
fig = cube.plot_surface(inc=50.0, PA=152.0, z0=0.3, psi=1.0,
                        r_max=4.5, fill='zvals', return_fig=True)
```

(continues on next page)

(continued from previous page)

```
ax = fig.axes[0]
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
```

[7]:
```
Text(0, 0.5, 'Offset (arcsec)')
```

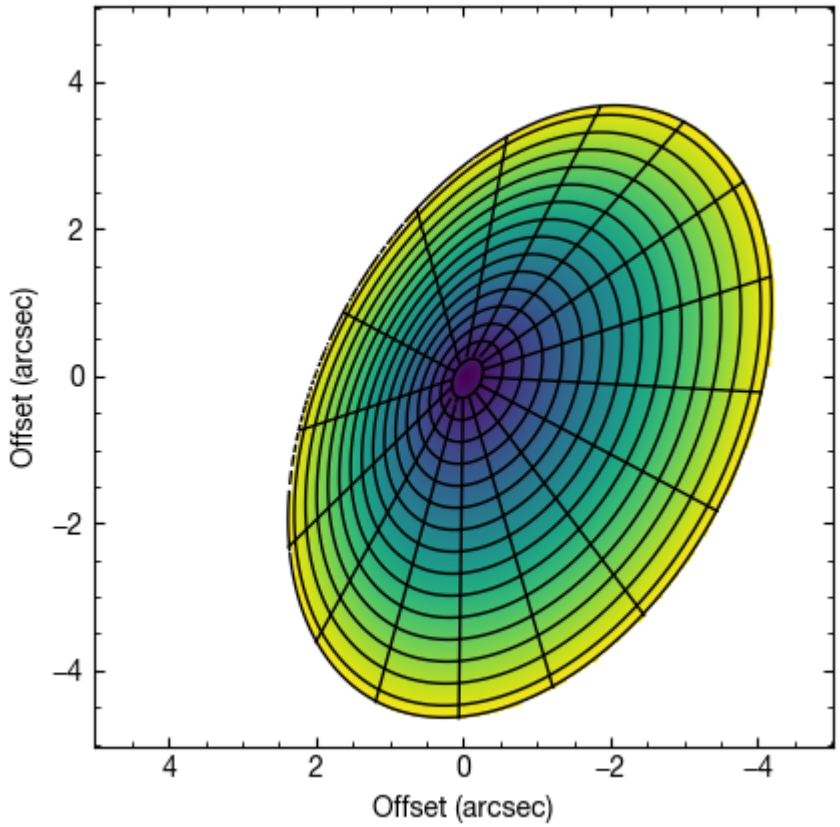


Note that in the above we've used the `fill` argument to fill the background color with `zvals`, the emission height.

We can also assume the disk is tilted in the opposite direction by flipping the sign of the inclination.

[8]:
```
fig = cube.plot_surface(inc=-50.0, PA=152.0, z0=0.3, psi=1.0,
                        r_max=4.5, fill='zvals', return_fig=True)

ax = fig.axes[0]
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
```

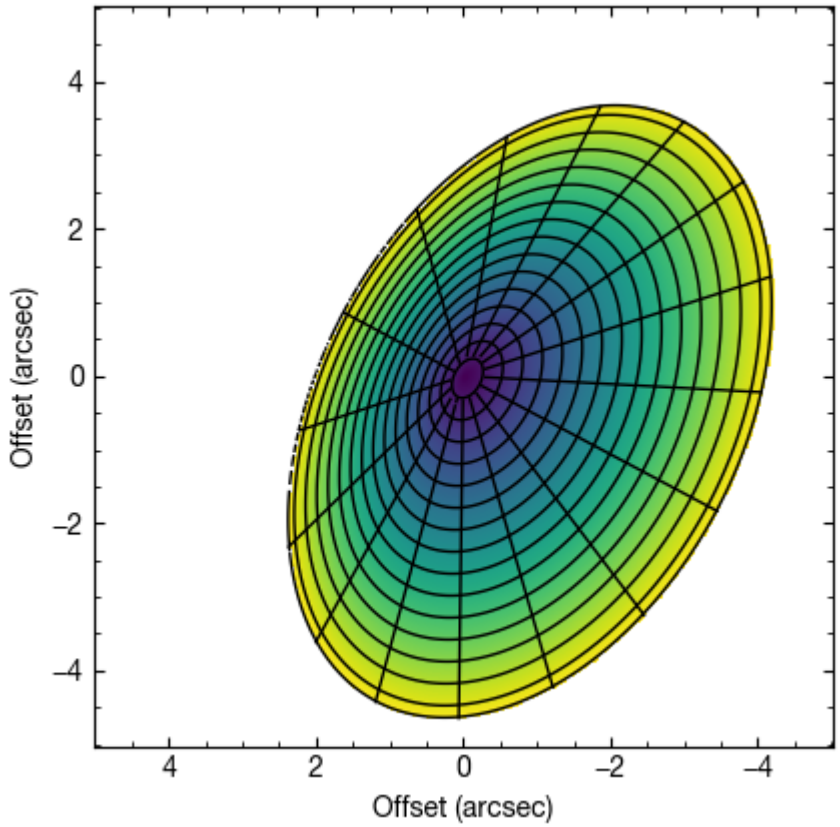
[8]:
```
Text(0, 0.5, 'Offset (arcsec)')
```



Note: To comply with standard orbital conventions, i should vary only between 0° and 180°. Thus you can include $i > 90°$, resulting in the correct orientation.

[9]:
```
fig = cube.plot_surface(inc=130.0, PA=152.0, z0=0.3, psi=1.0,
                        r_max=4.5, fill='zvals', return_fig=True)

ax = fig.axes[0]
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
```

[9]:
```
Text(0, 0.5, 'Offset (arcsec)')
```

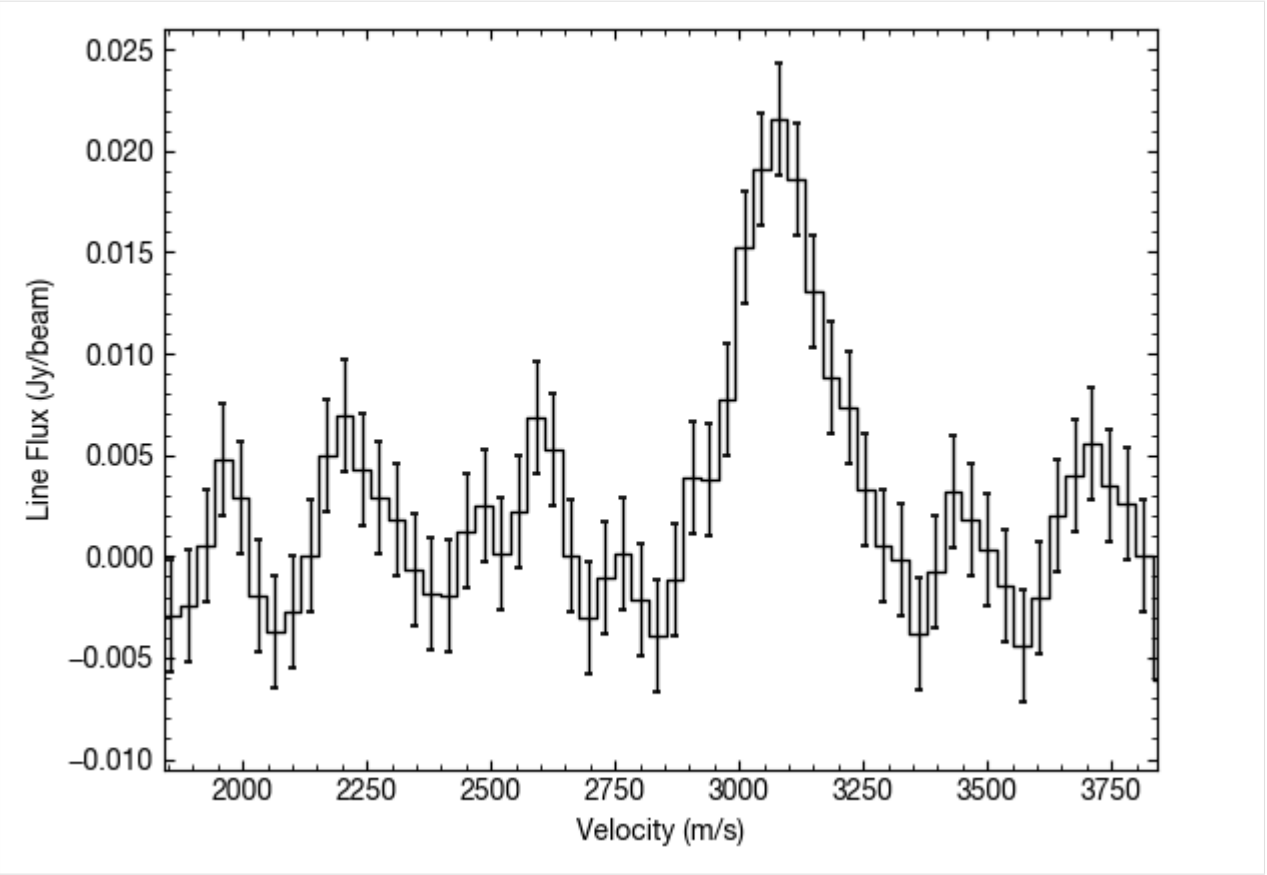


5.3.4 Extracting a Spectrum

With an idea of how to specify the coordinate systems, we can use this to help us extract spectra from various locations in the disk. This is made easy with the `get_spectrum` function - simply provide it the coordinates where you want the spectrum and voila!

[10]:
```
x, y, dy = cube.get_spectrum(coords=(1.5, -1.0), frame='sky', coord_type='cartesian')

fig, ax = plt.subplots()
ax.errorbar(x, y, dy, fmt=' ', capsize=1.25, capthick=1.25, color='k', lw=1.0)
ax.step(x, y, where='mid', color='k', lw=1.0)
ax.set_xlabel('Velocity (m/s)')
ax.set_ylabel('Line Flux (Jy/beam)')
ax.set_xlim(1.84e3, 3.84e3)
```

[10]:
```
(1840.0, 3840.0)
```

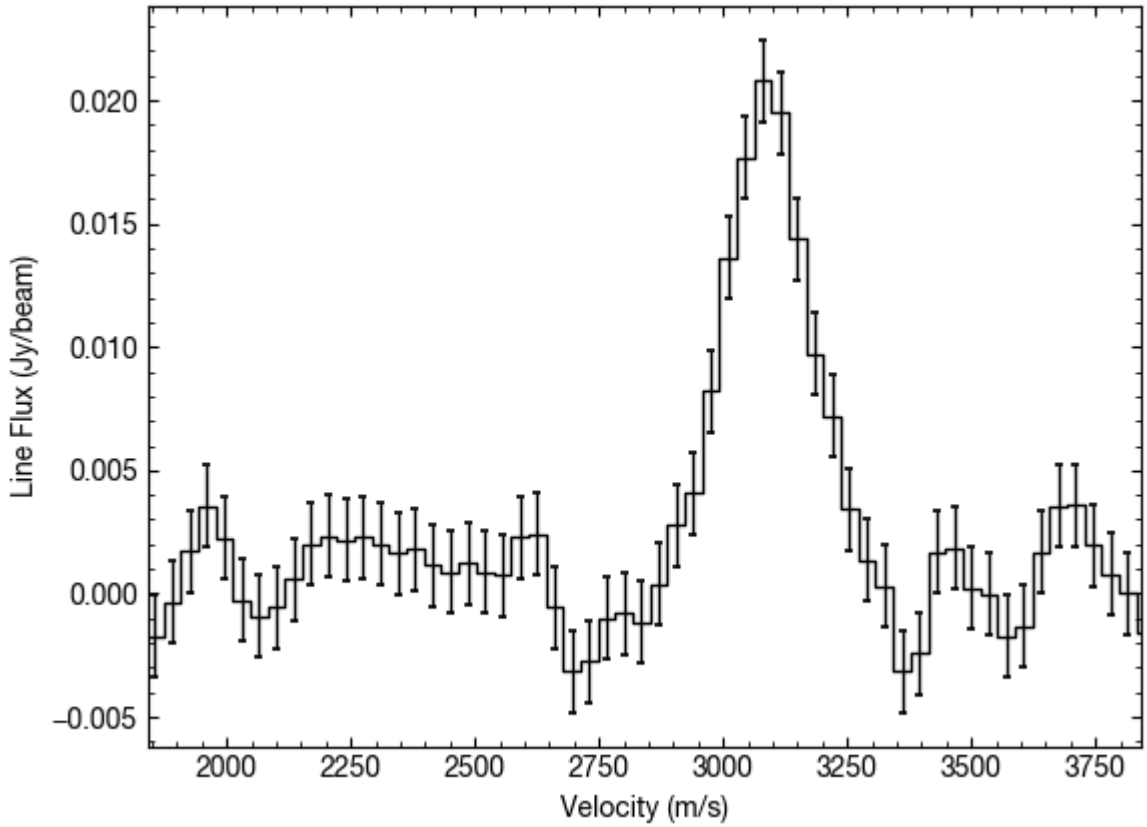


In the above we've extracted a spectrum at an offset from the image center of $(1.5'', -1.0'')$.

By default this is just a single pixel, but we can average over a larger area with the `area` argument. This specifies the area you want to average over as a fraction of the beam area. For example we can average over an area twice the size of the beam with `area=2.0`.

[11]:
```
x, y, dy = cube.get_spectrum(coords=(1.5, -1.0), frame='sky', coord_type='cartesian',␣
↪area=2.0)

fig, ax = plt.subplots()
ax.errorbar(x, y, dy, fmt=' ', capsize=1.25, capthick=1.25, color='k', lw=1.0)
ax.step(x, y, where='mid', color='k', lw=1.0)
ax.set_xlabel('Velocity (m/s)')
ax.set_ylabel('Line Flux (Jy/beam)')
ax.set_xlim(1.84e3, 3.84e3)
```

[11]:
```
(1840.0, 3840.0)
```



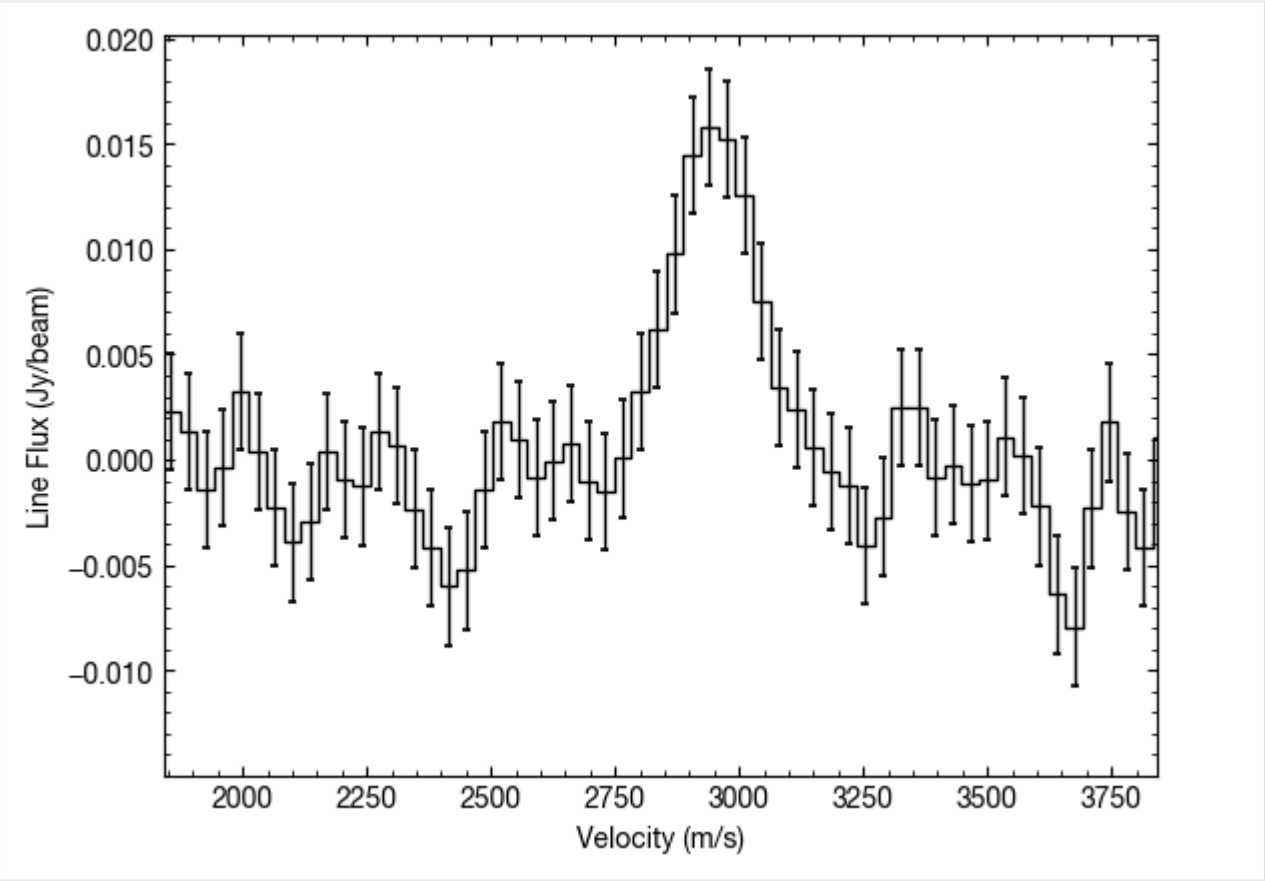
While this does limit some of the noise, remember that you're averaging over an area where the line profile will probably change, particularly if there's large gradients in the source properties.

Sometimes it's more useful to specify the location in the disk frame of reference, such as if you know where there's an interesting feature. We can do this by swapping `frame='disk'`. Again we can specify the coordinates in either cartesian, `coord_type='cartesian'`, or cylindrical frames, `coord_type='cylindrical'`, although disk-frame coordintes are more often than not specified in cylindrical form. Remember from above that `GoFish` assumes that $\phi = 0$ is the red-shifted major axis and ϕ will increase in a clockwise direction on the sky.

Here we extract a spectrum at a $2''$ offset along the south-west minor axis.

[12]:
```
x, y, dy = cube.get_spectrum(coords=(2.0, np.pi / 2.0), frame='disk', coord_type=
↪'cylindrical')

fig, ax = plt.subplots()
ax.errorbar(x, y, dy, fmt=' ', capsize=1.25, capthick=1.25, color='k', lw=1.0)
ax.step(x, y, where='mid', color='k', lw=1.0)
ax.set_xlabel('Velocity (m/s)')
ax.set_ylabel('Line Flux (Jy/beam)')
ax.set_xlim(1.84e3, 3.84e3)
```

[12]:
```
(1840.0, 3840.0)
```



5.3.5 Extracing a Disk Averaged Spectrum

Now that we know how to deproject the data, we can use this to map a Keplerian velocity field onto the disk and calculated the projected line of sight velocity, v_0, given by,

$$v_0(r,\,\phi) = \sqrt{\frac{GM_{\rm star}r^2}{(r^2+z^2)^{3/2}}}\,\cos(\phi)\,\sin(i),$$

where we are expected to know $M_{\rm star}$ ($0.88\,M_{\rm sun}$ for TW Hya), and the distance to the source (59.5 pc).

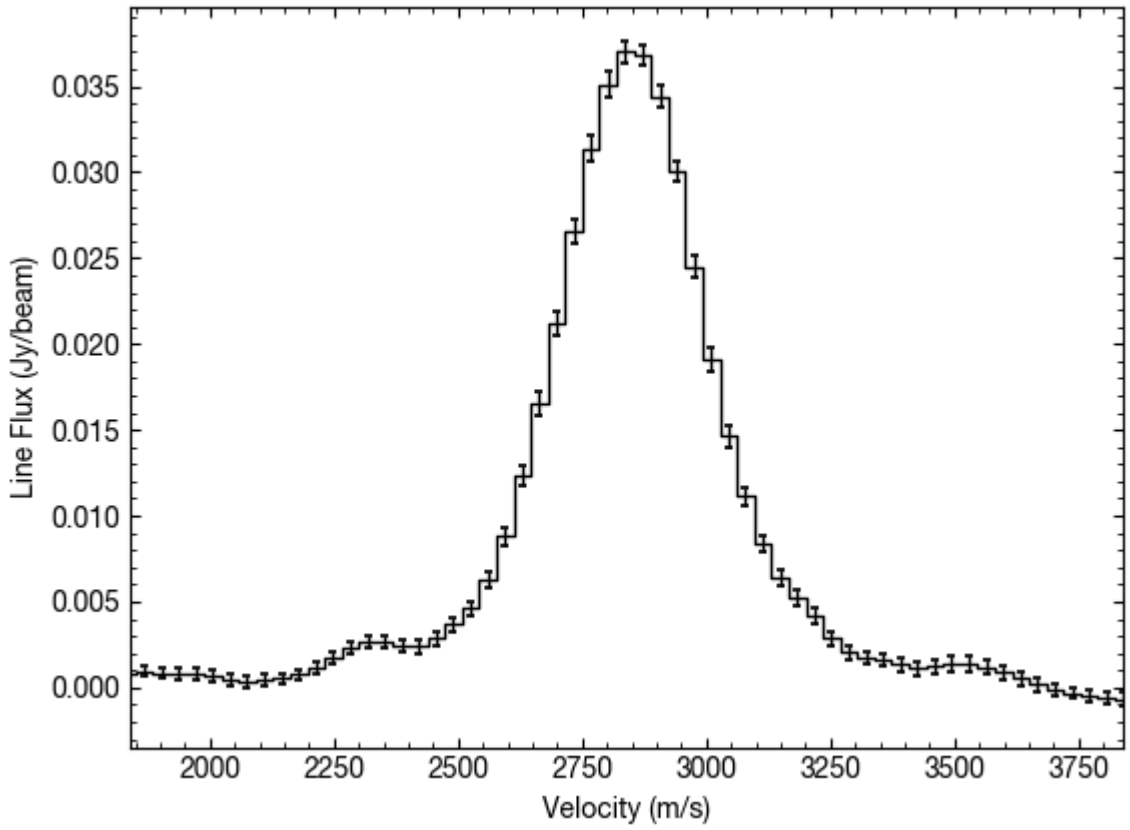
To apply this we use the `averaged_spectrum` function, which is provided the geometrical properties of the disk to calculate v_0, before applying these shifts to concentric annuli of the data before combining them, weighted by the area of each annulus. We must also tell it what radial range to consider through the `r_min` and `r_max` parameters, both given in arcseconds.

[13]:
```
x, y, dy = cube.average_spectrum(r_min=0.0, r_max=1.0, inc=5.0,
                                 PA=152., mstar=0.88, dist=59.5)
```

Plotting the data gives a nice spectrum:

[14]:
```
fig, ax = plt.subplots()
ax.errorbar(x, y, dy, fmt=' ', capsize=1.25, capthick=1.25, color='k', lw=1.0)
ax.step(x, y, where='mid', color='k', lw=1.0)
ax.set_xlabel('Velocity (m/s)')
ax.set_ylabel('Line Flux (Jy/beam)')
ax.set_xlim(1.84e3, 3.84e3)
```

[14]:
```
(1840.0, 3840.0)
```

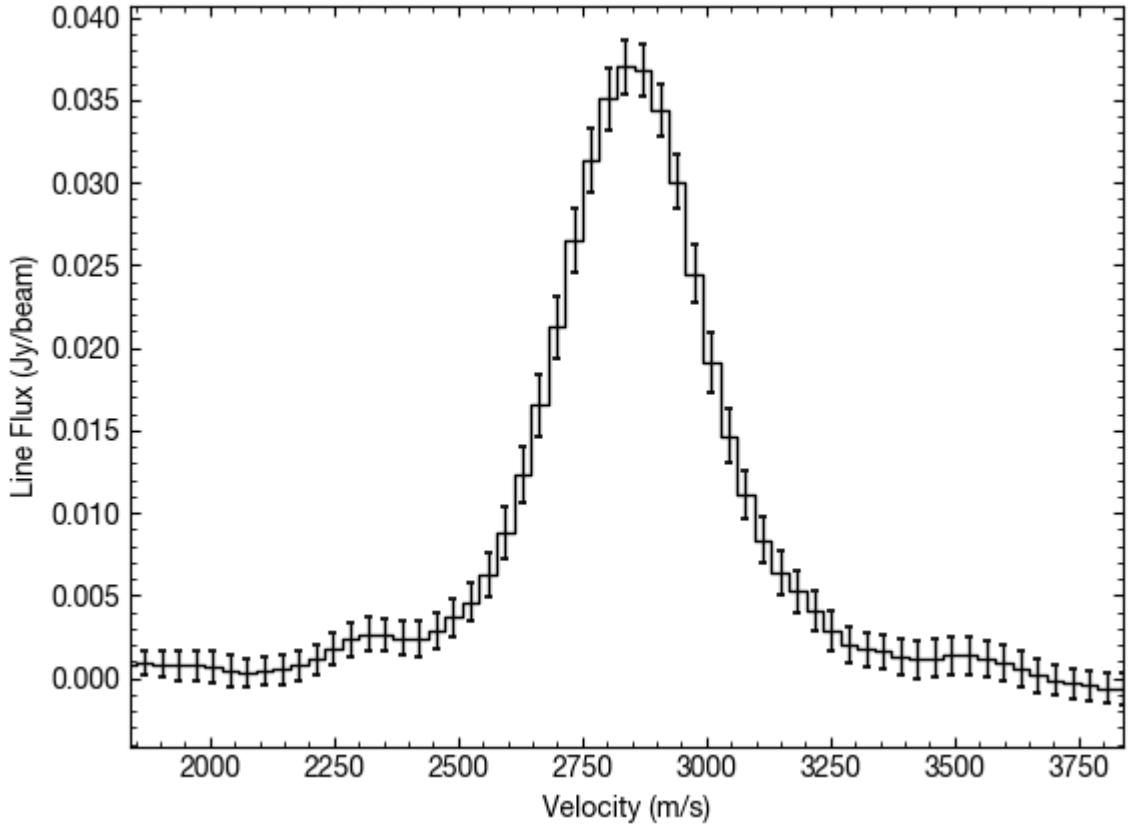


Note also that by default `dy` is the standard deviation of the samples in the velocity bin (calculated by calling for the standard deviation of the bin with `scipy.stats.binned_statistics`), rather than the uncertainty on the line flux. This is because the binned data is not strictly independent owing to both small spectral and spatial correlations in the data.

When we do the spectral shifting, we actually decorrelate pixels that are close together. This is discussed in Yen et al. (2016), where we essentially gain more independent samples. In `GoFish` this is implemented by default and uses an numerical approach, unlike the analytical discussed in Yen et al., to account for the beam shape and orientation. This can be ignored (slightly speeding up the calculation) by setting `include_spectral_decorrelation=False`, as below.

[15]:
```
x, y, dy = cube.average_spectrum(r_min=0.0, r_max=1.0, inc=5.0,
                                 PA=152., mstar=0.88, dist=59.5,
                                 include_spectral_decorrelation=False)

fig, ax = plt.subplots()
ax.errorbar(x, y, dy, fmt=' ', capsize=1.25, capthick=1.25, color='k', lw=1.0)
ax.step(x, y, where='mid', color='k', lw=1.0)
ax.set_xlabel('Velocity (m/s)')
ax.set_ylabel('Line Flux (Jy/beam)')
ax.set_xlim(1.84e3, 3.84e3)
```

[15]:
```
(1840.0, 3840.0)
```

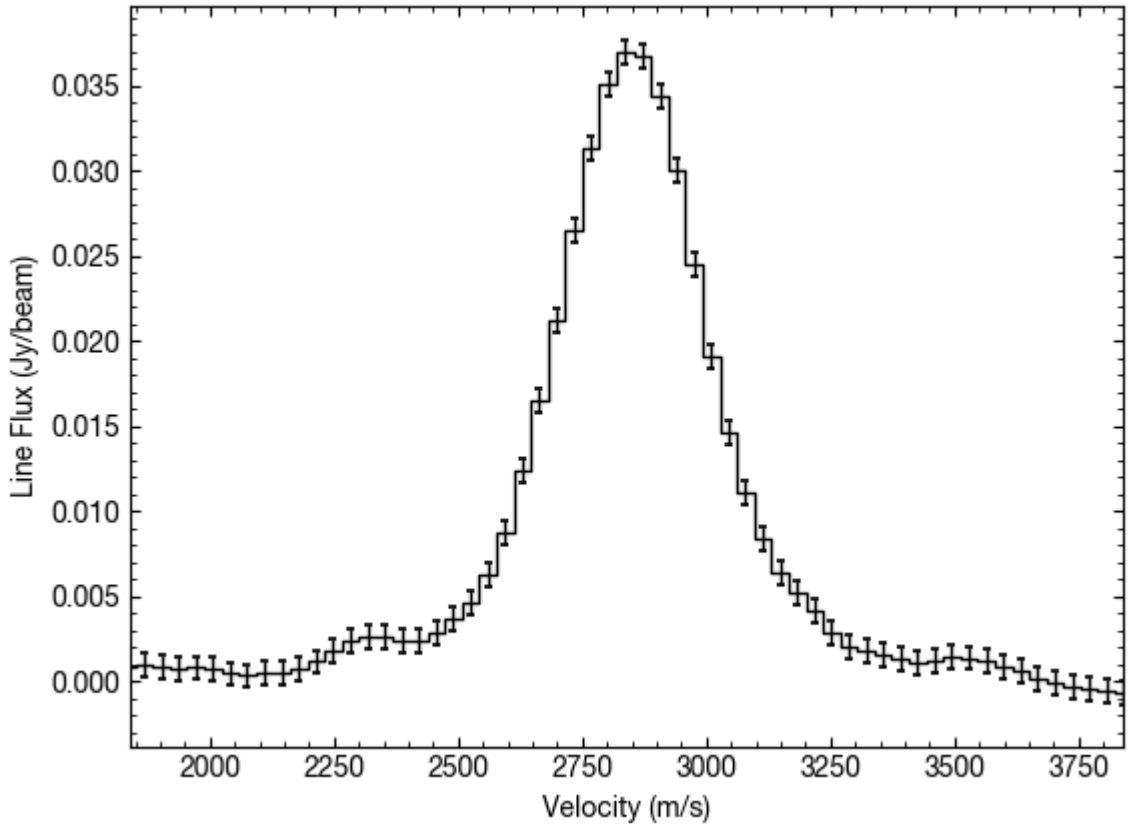


This results in a slightly larger uncertainty, but this is most noticeable when only considering area which have a large velocity gradient (most of TW Hya is a bad example for this as it is close to face on!).

We can also provide an empirical measure of the uncertainty by measuring the standard deviation in the spectrum wings. You can easily code up something which is more precise and can better define the line-free regions, but a quick work around would be to use the `empirical_uncertainty=True` option.

[16]:
```
x, y, dy = cube.average_spectrum(r_min=0.0, r_max=1.0, inc=5.0,
                                 PA=152., mstar=0.88, dist=59.5,
                                 empirical_uncertainty=True)

fig, ax = plt.subplots()
ax.errorbar(x, y, dy, fmt=' ', capsize=1.25, capthick=1.25, color='k', lw=1.0)
ax.step(x, y, where='mid', color='k', lw=1.0)
ax.set_xlabel('Velocity (m/s)')
ax.set_ylabel('Line Flux (Jy/beam)')
ax.set_xlim(1.84e3, 3.84e3)
```

[16]:
```
(1840.0, 3840.0)
```

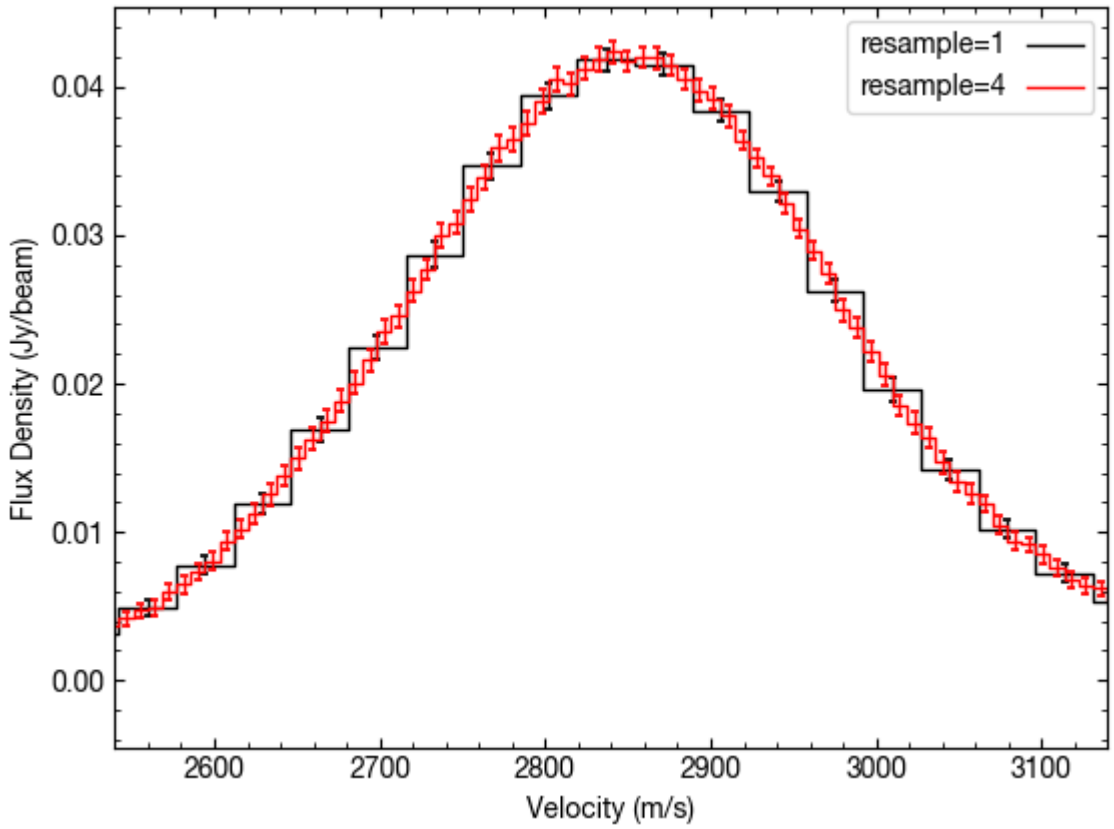


Because we end up sampling the line profile at a much higher rate thanks to the Doppler shift of the lines around the azimuth (see the discussion in Teague et al. 2018), we can resample at a higher rate. This is done through the `resample` parameter. By default this is `resample=1`, so returns the velocity axis attached to the cube.

If the argument is an `int`, then we super-sample by that factor, while if it is a `float`, then that is the channel spacing. For example:

[17]:
```
fig, ax = plt.subplots()

# Integeter
x, y, dy = cube.average_spectrum(r_min=0.5, r_max=1.0, inc=5.0, PA=152.,
                                 mstar=0.88, dist=59.5, dr=0.1, resample=1)
ax.errorbar(x, y, dy, fmt=' ', capsize=1.25, capthick=1.25, color='k', lw=1.0)
ax.step(x, y, where='mid', color='k', lw=1.0, label='resample=1')

# Float
x, y, dy = cube.average_spectrum(r_min=0.5, r_max=1.0, inc=5.0, PA=152.,
                                 mstar=0.88, dist=59.5, dr=0.1, resample=4)
ax.errorbar(x, y, dy, fmt=' ', capsize=1.25, capthick=1.25, color='r', lw=1.0)
ax.step(x, y, where='mid', color='r', lw=1.0, label='resample=4')

ax.legend(loc=1, markerfirst=False)
ax.set_xlabel('Velocity (m/s)')
ax.set_ylabel('Flux Density (Jy/beam)')
ax.set_xlim(2.54e3, 3.14e3)
```

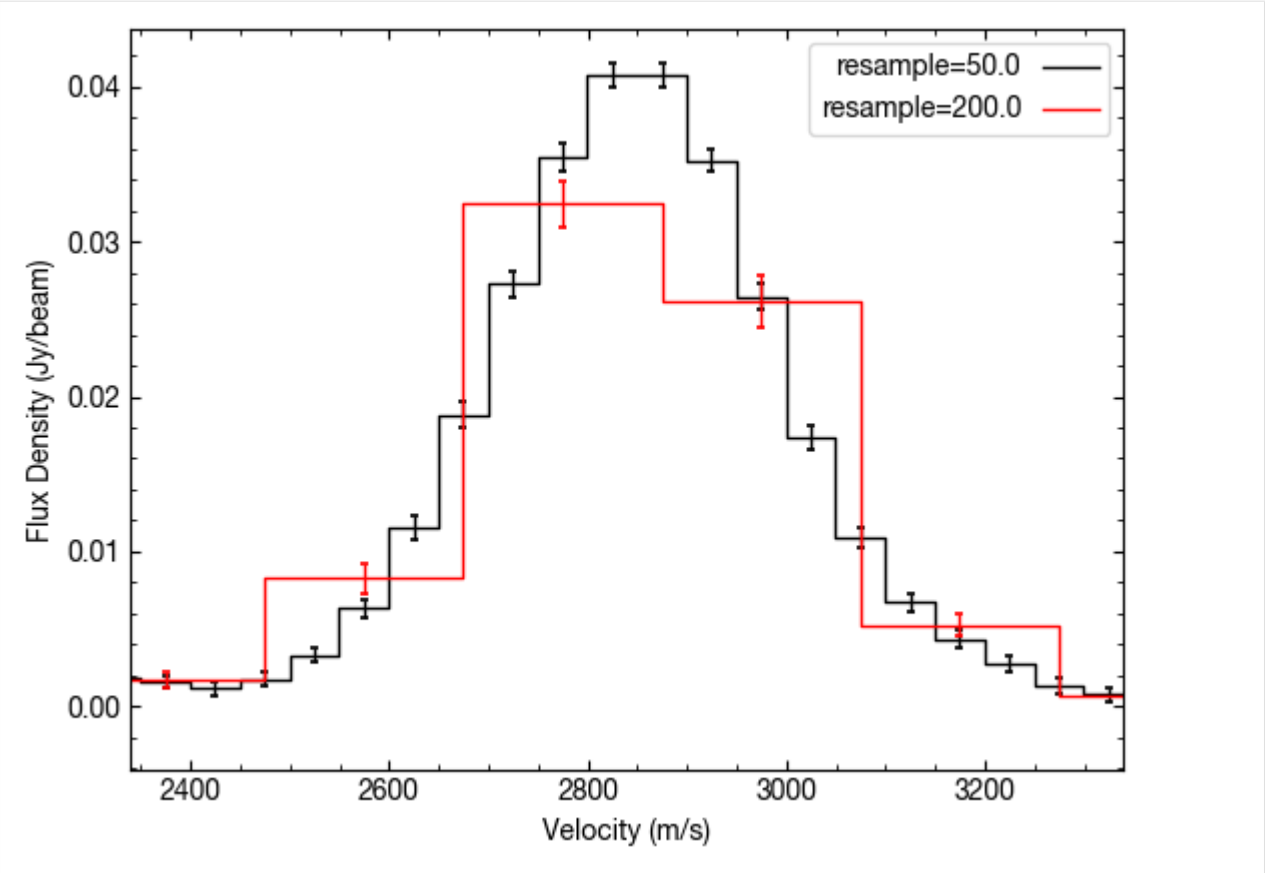
[17]:
```
(2540.0, 3140.0)
```



Similarlaly, we can use down-sample the velocity axis if we want to improve the SNR of the detection.

[18]:
```
fig, ax = plt.subplots()

# Integeter
x, y, dy = cube.average_spectrum(r_min=0.5, r_max=1.0, inc=5.0, PA=152.,
                                 mstar=0.88, dist=59.5, dr=0.1, resample=50.0)
ax.errorbar(x, y, dy, fmt=' ', capsize=1.25, capthick=1.25, color='k', lw=1.0)
ax.step(x, y, where='mid', color='k', lw=1.0, label='resample=50.0')

# Float
x, y, dy = cube.average_spectrum(r_min=0.5, r_max=1.0, inc=5.0, PA=152.,
                                 mstar=0.88, dist=59.5, dr=0.1, resample=200.0)
ax.errorbar(x, y, dy, fmt=' ', capsize=1.25, capthick=1.25, color='r', lw=1.0)
ax.step(x, y, where='mid', color='r', lw=1.0, label='resample=200.0')

ax.legend(loc=1, markerfirst=False)
ax.set_xlabel('Velocity (m/s)')
ax.set_ylabel('Flux Density (Jy/beam)')
ax.set_xlim(2.34e3, 3.34e3)
```

[18]:
```
(2340.0, 3340.0)
```



Quick Aside: By binning the data to lower spectral resolutions you are actually losing information, even though it may appear to improve things. If you know that your line profile is Gaussian, you'll get the best results when fitting a Gaussian line when you have the highest spectral sampling. See, for example Lenz & Ayres (1992).

Using the argument `unit='K'`, we can also return this in units of Kelvin, using the Rayleigh-Jeans approximation. And note it is exactly that, an approximation, and is a very poor conversion at sub-mm wavelengths, so use with caution! It does, however, offer a quick way to compare spectra *of the same frequency*, but imaged at different beam sizes.

5.3.6 Extracted a Disk Integrated Spectrum

In a very similar fashion we can extract a disk integrated spectrum, returning a spectrum in units of Jansky. Again we provide it the inner and outer radii to integrate over. Note here that this returns a single-peaked profile. This is because we've corrected for the Keplerian rotation of the disk prior to integrating (hence the need for specifying `mstar` and `dist`).

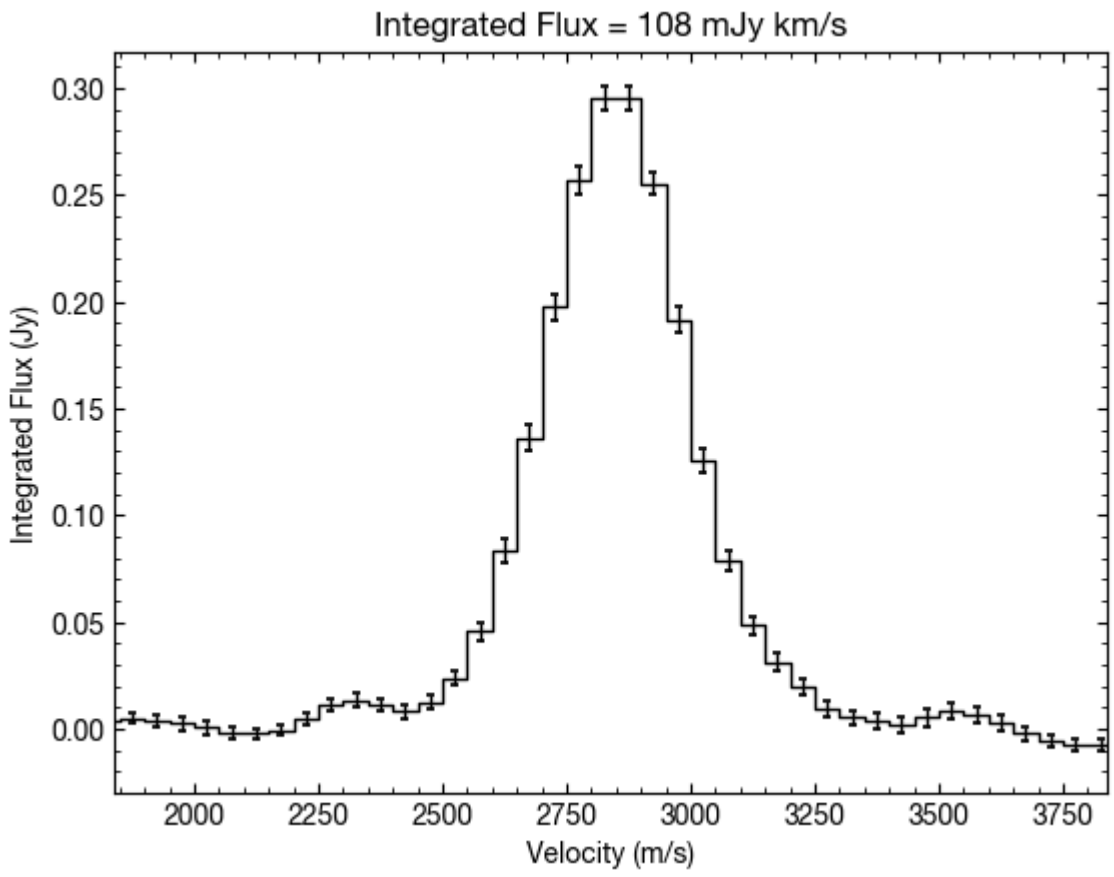
[19]:
```
fig, ax = plt.subplots()
x, y, dy = cube.integrated_spectrum(r_min=0.5, r_max=1.0, inc=5.0, PA=152.,
                                    mstar=0.88, dist=59.5, dr=0.1, resample=50.0)
ax.errorbar(x, y, dy, fmt=' ', capsize=1.25, capthick=1.25, color='k', lw=1.0)
ax.set_title('Integrated Flux = {:.0f} mJy km/s'.format(np.trapz(y, x)), fontsize=12)
ax.step(x, y, where='mid', color='k', lw=1.0)
```

(continues on next page)

(continued from previous page)

```
ax.set_xlabel('Velocity (m/s)')
ax.set_ylabel('Integrated Flux (Jy)')
ax.set_xlim(1.84e3, 3.84e3)
```

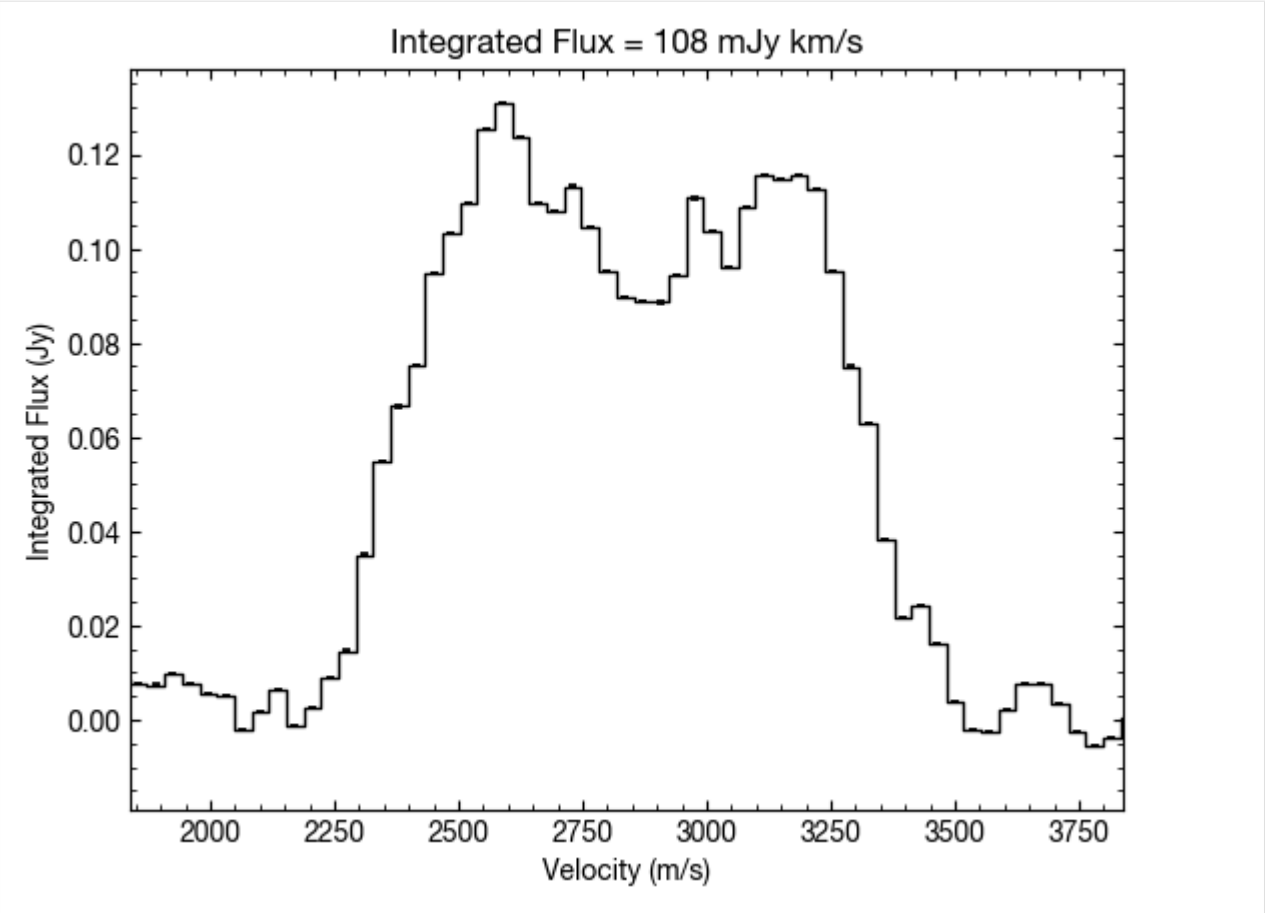
[19]: (1840.0, 3840.0)



If we were to remove these arguments then the function will default to a standard integration without correcting for the velocity structure of the disk. Here you can see the more typical double peaked profile of the disk, but note that the velocity integrated flux is the same.

[20]:
```
fig, ax = plt.subplots()
x, y, dy = cube.integrated_spectrum(r_min=0.5, r_max=1.0, inc=5.0, PA=152.)
ax.errorbar(x, y, dy, fmt=' ', capsize=1.25, capthick=1.25, color='k', lw=1.0)
ax.step(x, y, where='mid', color='k', lw=1.0)
ax.set_title('Integrated Flux = {:.0f} mJy km/s'.format(np.trapz(y, x)), fontsize=12)
ax.set_xlabel('Velocity (m/s)')
ax.set_ylabel('Integrated Flux (Jy)')
ax.set_xlim(1.84e3, 3.84e3)
```

[20]: (1840.0, 3840.0)



5.3.7 Radial Profiles

For well detected sources, you can also imaging wanting to plot radial profiles of their emission. Most typically people would plot the radial profile of the total integrated intensity, showing $\mathrm{Jy\,beam^{-1}\,km\,s^{-1}}$ as a function of radius, or the peak flux density, $\mathrm{Jy\,beam^{-1}}$, or the brightness temperature, K, as a function of radius.

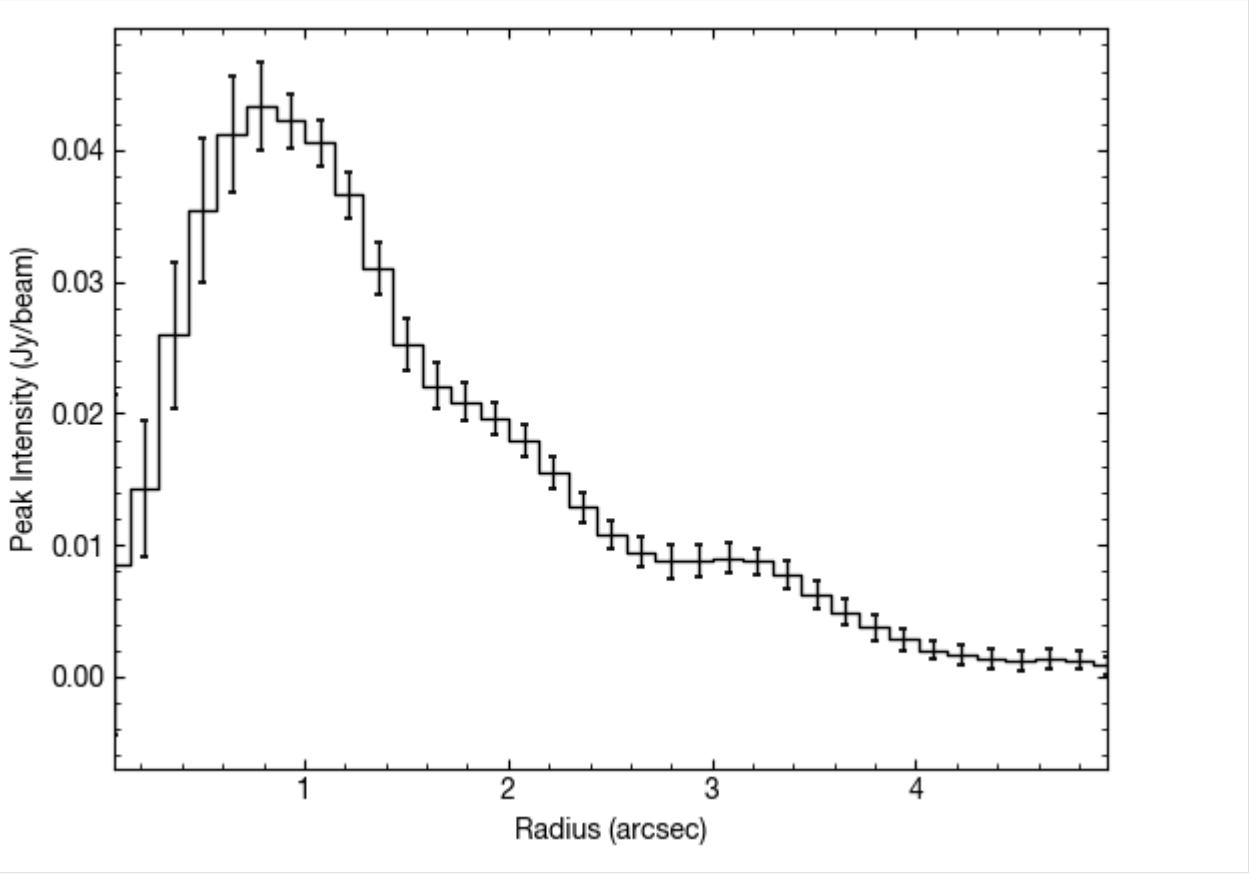
Most of the time this can be readily achieved by making moment maps (shameless plug for `bettermoments <https://github.com/richteague/bettermoments>`__ to make such moment maps) and creating the radial profiles. However, even for bright sources, in the outer disk this approach is limited to what can be detected on a *per pixel* basis. Shifting and stacking the spectra and give a significant improvement in the outer disk.

The easiest way to get a profile is to use the `radial_profile` function. As with the `averaged_spectrum` or `integrated_spectrum` you want to specify the source geometry. In addition, you can (should!) provide radial bin edges or centers (not both) and the unit you want. The allowed `unit` values are:

- `'Jy m/s'`: Integrated spectrum in Janskys.
- `'K m/s'`: Integrated spectrum in Kelvin.
- `'Jy'`: Peak of the integrated spectrum.
- `'Jy/beam'`: Peak of the averaged spectrum.
- `'K'`: Peak of the averaged spectrum in Kelvin.

[21]:
```
# Calculate the peak flux density as a function of radius.
x, y, dy = cube.radial_profile(inc=5., PA=152., mstar=0.88, dist=59.5, unit='Jy/beam')

# Plot
fig, ax = plt.subplots()
ax.errorbar(x, y, dy, fmt=' ', capsize=1.25, capthick=1.25, color='k', lw=1.0)
ax.step(x, y, where='mid', color='k', lw=1.0)
ax.set_xlim(x.min(), x.max())
ax.set_xlabel('Radius (arcsec)')
ax.set_ylabel('Peak Intensity (Jy/beam)')
```

[21]:
```
Text(0, 0.5, 'Peak Intensity (Jy/beam)')
```



For the integrated quantities, these are integrated over the *whole* spectrum. You might want to apply your own clipping thresholds or integration ranges. If this is the case you can return an array of all the spectra using the `radial_specrta` function which takes the same values. Note this time `unit` can only be `'Jy'`, `'Jy/beam'` or `'K'`. First we get the deprojected spectra.

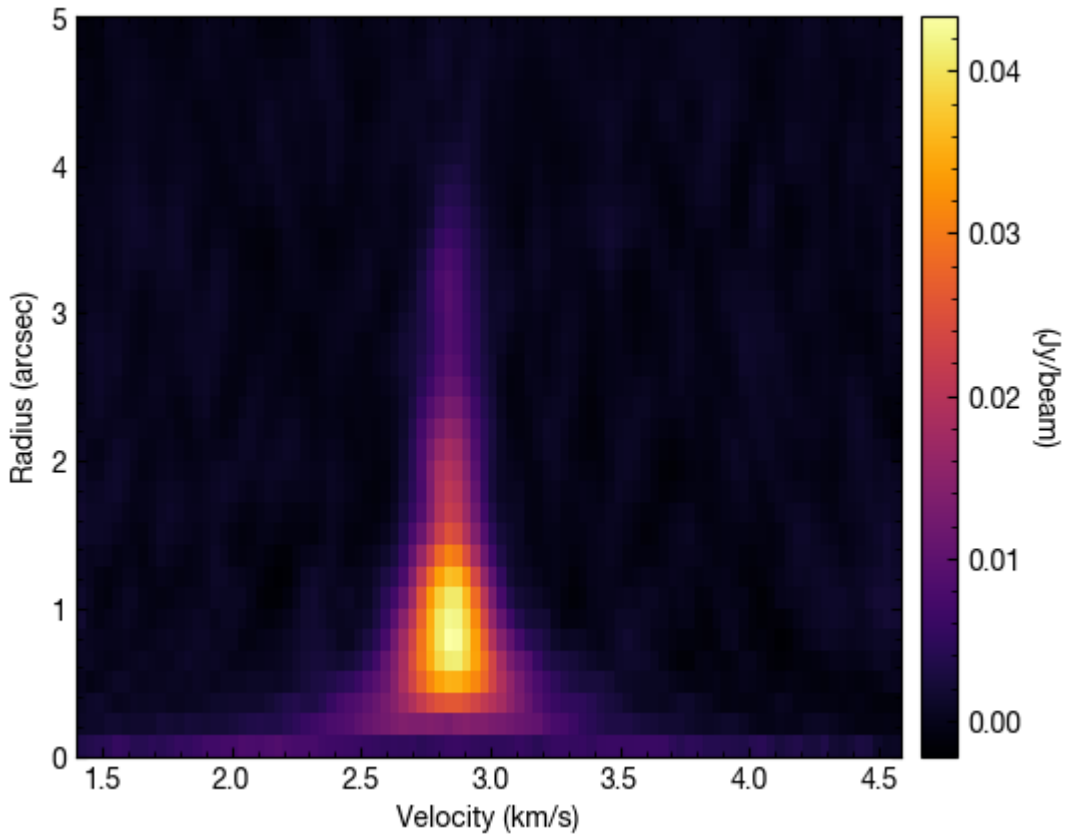
[22]:
```
rvals, velax, spectra, scatter = cube.radial_spectra(inc=5., PA=152., mstar=0.88,␣
↪dist=59.5, unit='Jy/beam')
```

Here `rvals` is the array of bin centers used for the profile while `spectra` is of shape `(M, 3, N)` where `M` is the number of radial samples and `N` is the number of velocity samples. The second (index 1) axis is split over velocity, flux density and uncertainty.

A nice way to plot all these spectra is as follows which nicely demonstrates how the lines get weaker in the outer disk but also narrower. This is called a 'teardrop plot'.

[23]:
```
cube.plot_teardrop(inc=5., PA=152., mstar=0.88, dist=59.5)
```

[23]:
```
<Axes: xlabel='Velocity (km/s)', ylabel='Radius (arcsec)'>
```



[ ]:

5.4 Advanced Fishing

This Notebook goes through some of the slightly more advanced tasks which can be performed with `GoFish`. As with the previous Notebook we will be using the CS (3-2) data from TW Hya. Make sure that cube is accessible when running this Notebook.

[1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from gofish import imagecube
```

[2]:
```
cube = imagecube('TWHya_CS_32.fits', FOV=6.0)
```

5.4.1 Checking Disk Center

If you don't have a strong line detection, it's hard to make sure that your image is centered. We can use the `find_center` function to apply a brute force approach to finding the optimum location. This will cycle through a

grid of `x0` and `y0` values and calculate the signal-to-noise ratio (SNR) of the resulting spectrum averaged over the provided parameters.

We can set the offset in the x- and y-direction considered, `dx` and `dy`, respectively, as well as the number of samples along each axis, `Nx` and `Ny`. By default we search up to a distance of the beam FWHM, sampling roughly every pixel.

We must also then consider how to calculate the noise, either on the averaged spectrum, `spectrum='avg'`, or on the integrated spectrum, `spectrum='int'`, and then whether to consider the signal the peak of that spectrum, `SNR='peak'`, or the integrated value, `SNR='int'`. By default we use the peak flux density of the averaged spectrum.

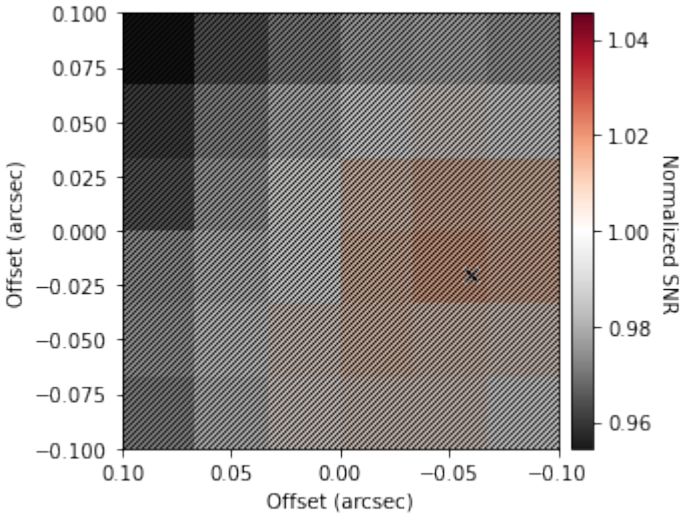
A mask is also needed, defining the regions where we calculate the signal, and the regions where to calculate the noise (we use the standard deviation of the masked pixels). By default the mask assumes the line emission is in the center 20% of the channels.

If you want to mask a specific range, you can use the `v_min` and `v_max` values. If your line profile is more complex, for example contains multiple transitions, you can include a user defined masked directly with the `mask` argument. Make sure that this is the same size as your expected velocity axis based on the `resample` parameter.

Note that this may take a while, particularly if you're searching a large area!

[3]:
```
x0s, y0s, SNR = cube.find_center(dx=0.1, dy=0.1, r_min=0.5, r_max=1.0,
                                 v_min=1.84e3, v_max=3.84e3,
                                 SNR='peak', inc=5.0, PA=152., mstar=0.88,
                                 dist=59.5, resample=50.0)
```

```
Peak SNR at (x0, y0) = (-0.06", -0.02").
```



What's plotted here is the SNR relative to the value at the image center, and in the bottom left corner, the synthesized beam as a size reference. This makes sense as the image was centered based on continuum.

Clearly the offset spectrum looks a lot better, as expected!

5.4.2 Masking Regions

As demonstrated in Yen et al. (2016), this method can be applied to specfic sections of the disk, i.e. you do not have to consider the disk as a whole.

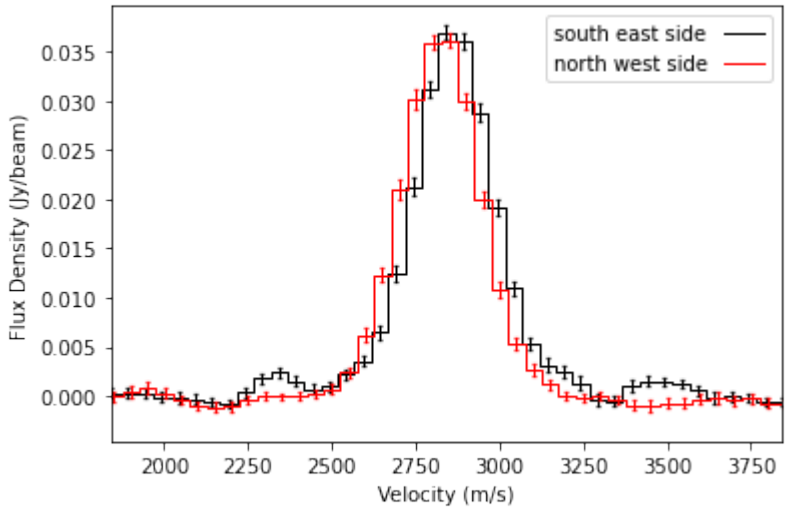
We've already seen the `r_min` and `r_max` values used in `average_spectrum` and `integrated_spectrum` which set the inner and outer radii of the annuli. But we also have `PA_min` and `PA_max`, which allow for similar

masks in the azimuthal direction. The are provided in degrees, where $\theta = 0$ aligns with the red-shifted major axis of the disk, and increasing in an eastwards direction, spanning $\pm 180°$. Thus, to only consider the red-shifted side of the disk, we use `PA_min=-90.0` and `PA_max=90.0`.

Because of the change in sign, we can access the other half of the disk through the `exclude_PA=True`, which by default is `False`.

[4]:
```
fig, ax = plt.subplots()

x, y, dy = cube.average_spectrum(r_min=0.5, r_max=1.5, inc=5.0, PA=152.,
                                 mstar=0.88, dist=59.5, resample=50.0,
                                 PA_min=-90.0, PA_max=90.0, exclude_PA=False)

ax.errorbar(x, y, dy, fmt=' ', lw=1.0, color='k', capsize=1.25, capthick=1.25)
ax.step(x, y, where='mid', color='k', lw=1.0, label='south east side')

x, y, dy = cube.average_spectrum(r_min=0.5, r_max=1.5, inc=5.0, PA=152.,
                                 mstar=0.88, dist=59.5, resample=50.0,
                                 PA_min=-90.0, PA_max=90.0, exclude_PA=True)

ax.errorbar(x, y, dy, fmt=' ', lw=1.0, color='r', capsize=1.25, capthick=1.25)
ax.step(x, y, where='mid', color='r', lw=1.0, label='north west side')

ax.legend(markerfirst=False)
ax.set_ylabel('Flux Density (Jy/beam)')
ax.set_xlabel('Velocity (m/s)')
ax.set_xlim(1.84e3, 3.84e3)
```

[4]:
```
(1840.0, 3840.0)
```



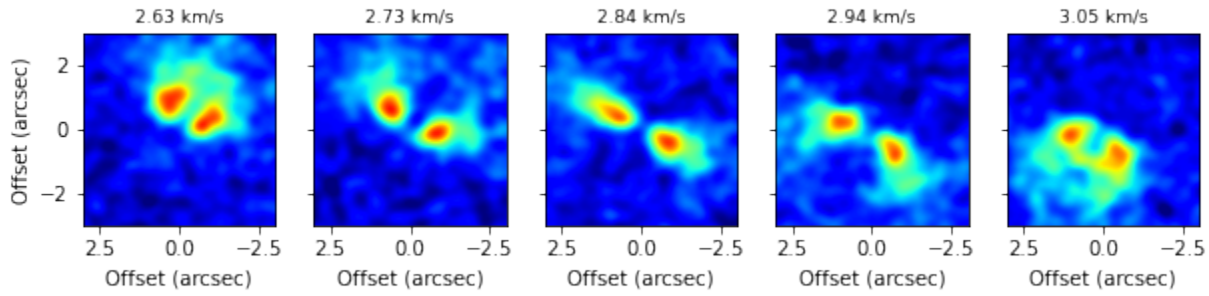
Note that there is some slight shift between the two lines because of beam convolution effects.

5.4.3 Shifted Cube

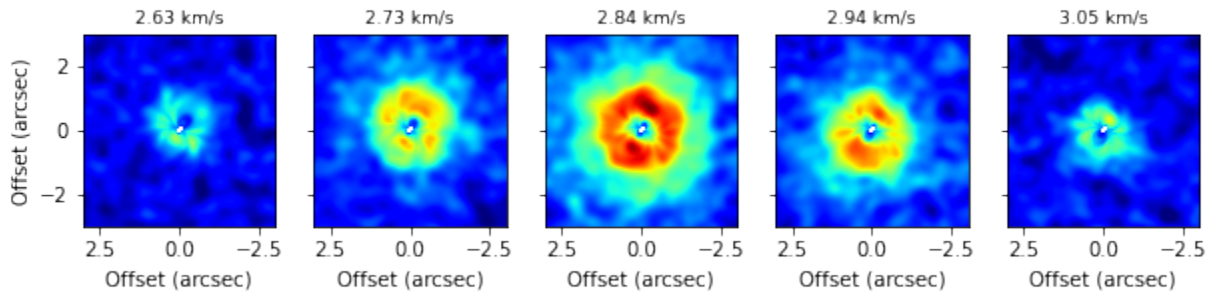
More details coming.

[5]:
```
shifted = cube.shifted_cube(inc=5.0, PA=152., mstar=0.88, dist=59.5)
```

[6]:
```
fig, axs = plt.subplots(ncols=5, figsize=(10, 2))

v0 = abs(cube.velax - 2.84e3).argmin()
for a, ax in enumerate(axs):
    idx = a * 3 + v0 - 6
    data = np.nanmean(cube.data[idx-1:idx+2], axis=0)
    ax.imshow(data, origin='lower', extent=cube.extent, vmin=-0.005, vmax=0.05, cmap=
↪'jet')
    if a > 0:
        ax.set_yticklabels([])
    else:
        ax.set_ylabel('Offset (arcsec)')
    ax.set_xlabel('Offset (arcsec)')
    ax.set_title('{:.2f} km/s'.format(cube.velax[idx] / 1e3), fontsize=9)
```



[7]:
```
fig, axs = plt.subplots(ncols=5, figsize=(10, 2))

v0 = abs(cube.velax - 2.84e3).argmin()
for a, ax in enumerate(axs):
    idx = a * 3 + v0 - 6
    data = np.nanmean(shifted[idx-1:idx+2], axis=0)
    ax.imshow(data, origin='lower', extent=cube.extent, vmin=-0.005, vmax=0.05, cmap=
↪'jet')
    if a > 0:
        ax.set_yticklabels([])
    else:
        ax.set_ylabel('Offset (arcsec)')
    ax.set_xlabel('Offset (arcsec)')
    ax.set_title('{:.2f} km/s'.format(cube.velax[idx] / 1e3), fontsize=9)
```



5.5 Masking with GoFish

This notebook will discuss the various ways a user can build a mask in GoFish and use it for the various pixel stacking functions.

[1]:
```
import matplotlib.pyplot as plt
from gofish import imagecube
import numpy as np
```

[2]:
```
import os
if not os.path.exists('TWHya_CS_32_M0.fits'):
    !wget -O TWHya_CS_32_M0.fits -q https://dataverse.harvard.edu/api/access/datafile/
↪:persistentId?persistentId=doi:10.7910/DVN/LO2QZM/NYVZKS
```

[3]:
```
cube = imagecube('TWHya_CS_32_M0.fits', FOV=10.0)
```

```
WARNING: Provided cube is only 2D. Shifting not available.
```

5.5.1 Defining a Mask

Most of the functions in GoFish allow the user to define a mask with the get_mask function. In general, you can check what the mask looks like using the plot_mask function to check it is doing what you want it to do.

Below we'll discuss some of the various masks one can use.

Simple Mask

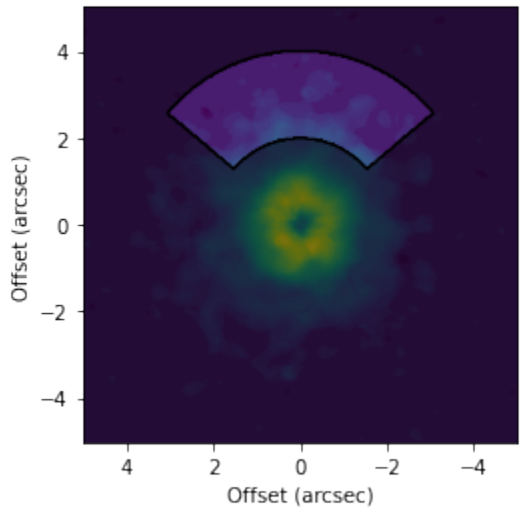
The most simple mask is just bounded in both radius and azimuth. This has the following variables:

- r_min - The inner radius of the mask in arcseconds.
- r_max - The outer radius of the mask in arcseconds.
- PA_min - The minimum position / polar angle in degrees.
- PA_max - The maximum position / polar angle in degrees.
- mask_frame - Whether the radial and azimuthal coordinates represent the disk-frame, mask_frame='disk', the default value, or the sky-frame, mask_frame='sky'.

When using mask_frame='disk', you must also specify the geometrical properties of the disk, i.e. at least the inclination and position angle of the disk. In this coordinate system, all polar angles are measured from the red-shifted major axis of the disk, while for mask_frame='sky', all position angles are measured from North.

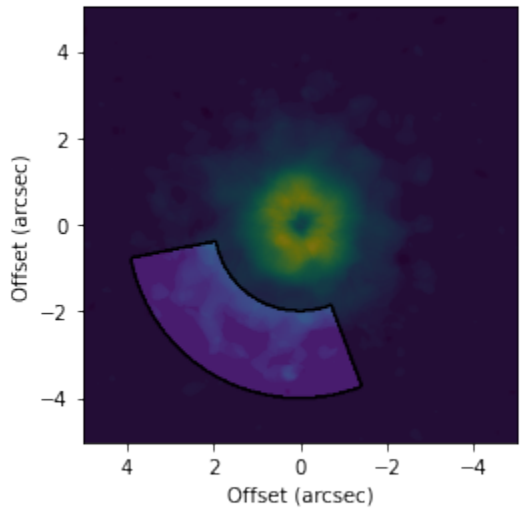
[4]:
```
# Make a mask specified in the sky-plane.
fig, ax = plt.subplots()
ax.imshow(cube.data, origin='lower', extent=cube.extent)
cube.plot_mask(ax=ax, r_min=2.0, r_max=4.0, PA_min=-50.0, PA_max=50.0, mask_frame='sky
↪')
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
```

[4]:
```
Text(0, 0.5, 'Offset (arcsec)')
```



[5]:
```
# Make a mask specified in the disk-plane.
fig, ax = plt.subplots()
ax.imshow(cube.data, origin='lower', extent=cube.extent)
cube.plot_mask(ax=ax, r_min=2.0, r_max=4.0, PA_min=-50.0, PA_max=50.0, inc=6.5,␣
↪PA=151.0, mask_frame='disk')
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
```

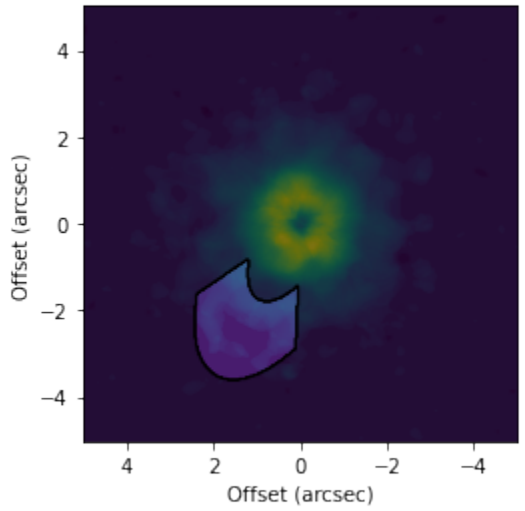
[5]:
```
Text(0, 0.5, 'Offset (arcsec)')
```



It is also important to note that with highly inclined disks, the perspective means that masks specified in the disk-plane wil not have the same projected azimuthal extent. Consider the above mask, but now for a highly inclined disk.

[6]:
```
# Make a mask specified in the disk-plane.
fig, ax = plt.subplots()
ax.imshow(cube.data, origin='lower', extent=cube.extent)
cube.plot_mask(ax=ax, r_min=2.0, r_max=4.0, PA_min=-50.0, PA_max=50.0, inc=65.0,␣
↪PA=151.0, mask_frame='disk')
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
```

[6]:
```
Text(0, 0.5, 'Offset (arcsec)')
```



This gives a very different mask than the one above.

5.5.2 Symmetric Masks

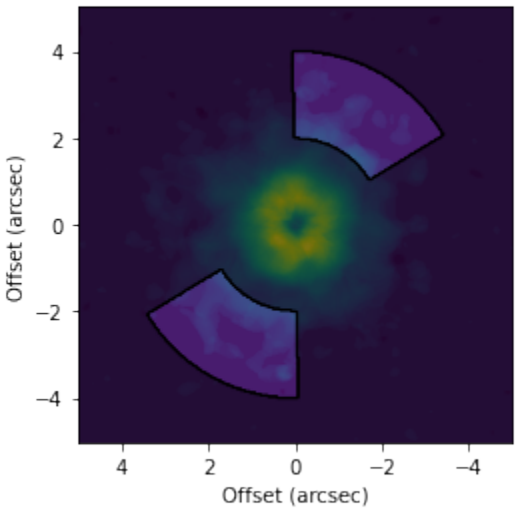
You may want to define a mask which is symmetric, for example encompasses a small range about the major or minor axes. This can be achieved with the following variables:

- `exclude_r` - Whether to exclude the radial range given, default is `False`.
- `exclude_PA` - Whether to exclude the azimuthal range given, default is `False`.
- `abs_PA` - Calculate the mask taking the absolute values of the polar / position angles.

For example, to create a mask that is +/- 30 degrees about the major or minor axis:

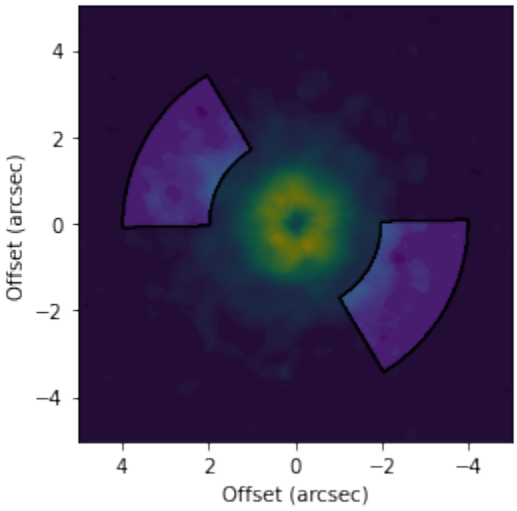
[7]:
```
# Make a mask symmetric about the major axis.
fig, ax = plt.subplots()
ax.imshow(cube.data, origin='lower', extent=cube.extent)
cube.plot_mask(ax=ax, r_min=2.0, r_max=4.0, PA_min=30.0, PA_max=150.0, abs_PA=True,␣
↪exclude_PA=True,
               inc=6.5, PA=151.0, mask_frame='disk')
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
```

[7]:
```
Text(0, 0.5, 'Offset (arcsec)')
```



[8]:
```
# Make a mask symmetric about the minor axis.
fig, ax = plt.subplots()
ax.imshow(cube.data, origin='lower', extent=cube.extent)
cube.plot_mask(ax=ax, r_min=2.0, r_max=4.0, PA_min=60.0, PA_max=120.0, abs_PA=True,␣
↪exclude_PA=False,
               inc=6.5, PA=151.0, mask_frame='disk')
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
```

[8]:
```
Text(0, 0.5, 'Offset (arcsec)')
```

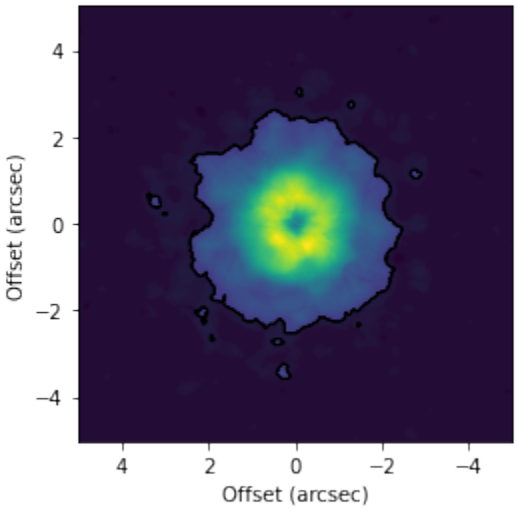


5.5.3 User-Defined Masks

If you want something more complex, you can input the mask yourself. For example if you want to generate a mask based on the emission morphology or some more complex function. Most functions will take directly a `mask` value which should have the same shape as the data, either a channel for an imagecube or the moment map size.

[9]:
```
# Make a mask based on the emission morphology.
fig, ax = plt.subplots()
ax.imshow(cube.data, origin='lower', extent=cube.extent)

# Make a mask based on the maximum value of the imagecube.
mask = cube.data > 0.1 * np.max(cube.data)

cube.plot_mask(ax=ax, mask=mask)
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
```

[9]:
```
Text(0, 0.5, 'Offset (arcsec)')
```



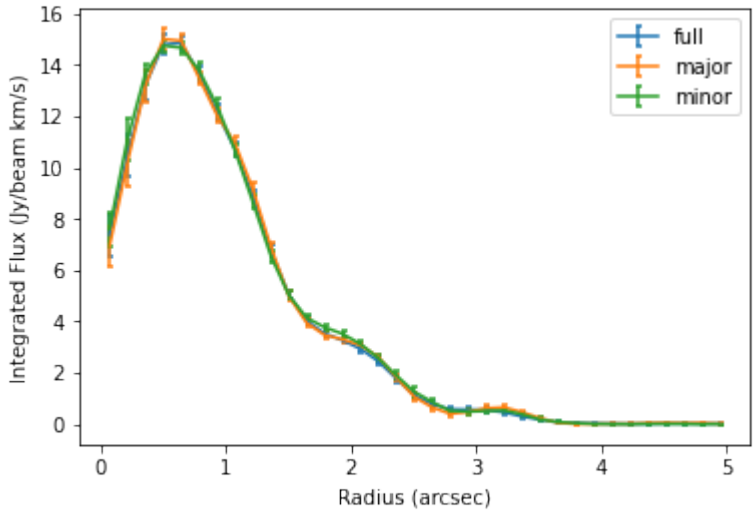
5.5.4 Usage with Other Functions

Once we have a mask you're happy with, you can use it with most `GoFish` functions. Below we demonstrate how this affects the radial profile function.

[10]:
```
fig, ax = plt.subplots()

# radial profile from the full azimuth.
x, y, dy = cube.radial_profile(inc=6.5, PA=151.0)
ax.errorbar(x, y, dy, capsize=1.5, capthick=1.5, label='full')

# radial profile from the major axis.
x, y, dy = cube.radial_profile(inc=6.5, PA=151.0, PA_min=20.0,
                               PA_max=160.0, abs_PA=True, exclude_PA=True)
ax.errorbar(x, y, dy, capsize=1.5, capthick=1.5, label='major')

# radial profile from the minor axis.
x, y, dy = cube.radial_profile(inc=6.5, PA=151.0, PA_min=70.0,
                               PA_max=110.0, abs_PA=True, exclude_PA=False)
ax.errorbar(x, y, dy, capsize=1.5, capthick=1.5, label='minor')

ax.legend()
ax.set_xlabel('Radius (arcsec)')
ax.set_ylabel('Integrated Flux (Jy/beam km/s)')
```

```
WARNING: Attached data is not 3D, so shifting cannot be applied.
         Reverting to standard azimuthal averaging; will ignore `unit` argument.
WARNING: Attached data is not 3D, so shifting cannot be applied.
         Reverting to standard azimuthal averaging; will ignore `unit` argument.
WARNING: Attached data is not 3D, so shifting cannot be applied.
         Reverting to standard azimuthal averaging; will ignore `unit` argument.
```

[10]:
```
Text(0, 0.5, 'Integrated Flux (Jy/beam km/s)')
```



Note: If there is a function which does not accept these mask parameters and you think it should, please raise an issue on GitHub.

5.6 Searching For (Sub-)Structure in Disks

There is a wealth of information that is hidden in the line emission. While `GoFish` is useful for teasing our low level emission from noisy data, we can also harness the velocity structure to tease out subtle deivations in the background physical or dynamical structure.

This notebook will guide you through a few ways you can explore the data a little futher. It'll use the zeroth moment map of the CS data we've worked with in the Masking Tutorial.

[1]:
```
import os
if not os.path.exists('TWHya_CS_32_M0.fits'):
    !wget -O TWHya_CS_32_M0.fits -q https://dataverse.harvard.edu/api/access/datafile/
↪:persistentId?persistentId=doi:10.7910/DVN/LO2QZM/NYVZKS
```

5.6.1 Load the Data

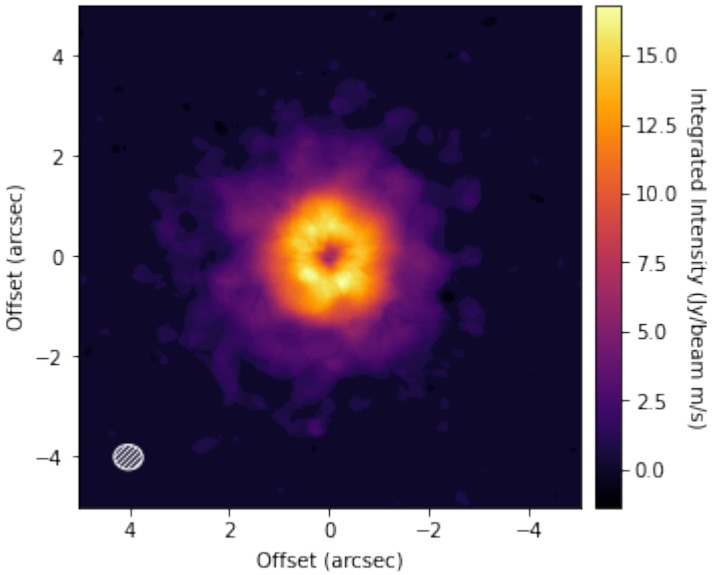
The first port of call is to inspect the data, so let's load it up.

[2]:
```
import matplotlib.pyplot as plt
from gofish import imagecube
import numpy as np
```

[3]:
```
cube = imagecube('TWHya_CS_32_M0.fits', FOV=10.0)
```

```
WARNING: Provided cube is only 2D. Shifting not available.
```

[4]:
```
fig, ax = plt.subplots(constrained_layout=True)
im = ax.imshow(cube.data, origin='lower', extent=cube.extent, cmap='inferno')
cb = plt.colorbar(im, pad=0.02)
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
cb.set_label('Integrated Intensity (Jy/beam m/s)', rotation=270, labelpad=13)
cube.plot_beam(ax=ax, color='w')
```

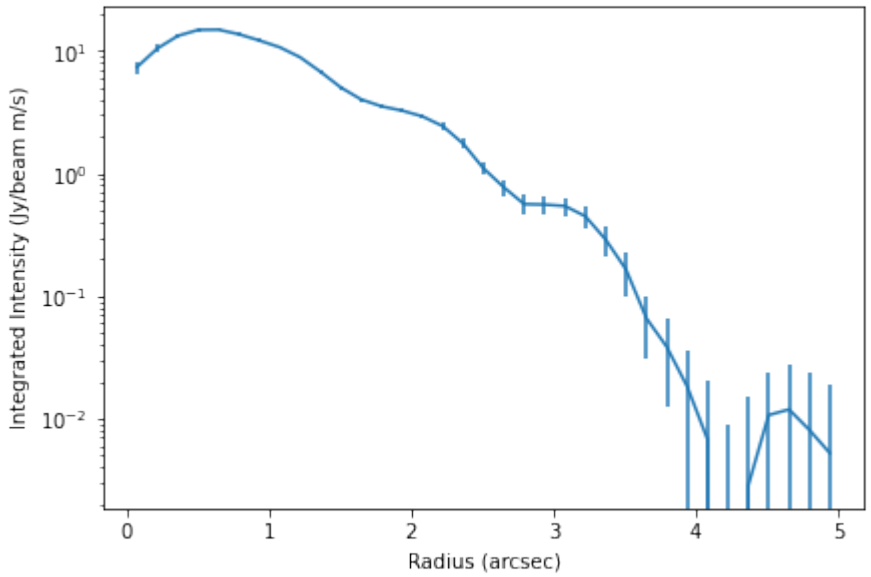


[5]:
```
x, y, dy = cube.radial_profile(inc=5.0, PA=151.0)
```

```
WARNING: Attached data is not 3D, so shifting cannot be applied.
         Reverting to standard azimuthal averaging; will ignore `unit` argument.
```

[6]:
```
fig, ax = plt.subplots(constrained_layout=True)
ax.errorbar(x, y, dy)
ax.set_yscale('log')
ax.set_xlabel('Radius (arcsec)')
ax.set_ylabel('Integrated Intensity (Jy/beam m/s)')
```

[6]:
```
Text(0, 0.5, 'Integrated Intensity (Jy/beam m/s)')
```



You can clearly see bumps and wiggles associated with the gaps and rings in the system.

5.6.2 Localized Deviations

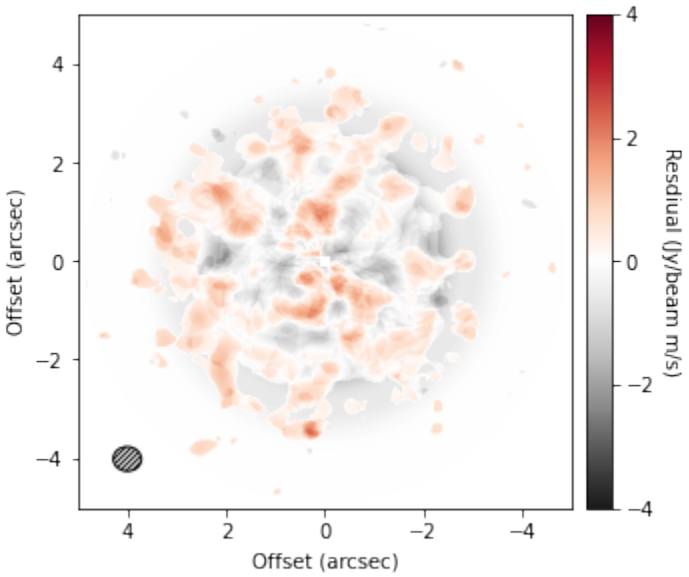
One of the most exciting ideas for causing gaps and rings are embedded planets. Cleeves et al. (2015) suggested that a way to find these features is to subtract a mean background to reveal localized enhancements in emission which can be associated with embedded planets.

To do this, we can use the `background_residual` function. In brief, this function makes an azimuthally averaged radial profile of the data (as we have done above), then projects it onto the sky to make a mean background model which is subtracted from the data.

[7]:
```
residual = cube.background_residual(inc=5.0, PA=151.0)
```

```
WARNING: Attached data is not 3D, so shifting cannot be applied.
         Reverting to standard azimuthal averaging; will ignore `unit` argument.
```

[8]:
```
fig, ax = plt.subplots(constrained_layout=True)
im = ax.imshow(residual, origin='lower', extent=cube.extent, cmap='RdGy_r', vmin=-4,␣
↪vmax=4)
cb = plt.colorbar(im, pad=0.02, ticks=np.arange(-4, 5, 2))
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
cb.set_label('Resdiual (Jy/beam m/s)', rotation=270, labelpad=13)
cube.plot_beam(ax=ax)
```



Clearly this looks just looks like noise, but we'll need to try and find better data.

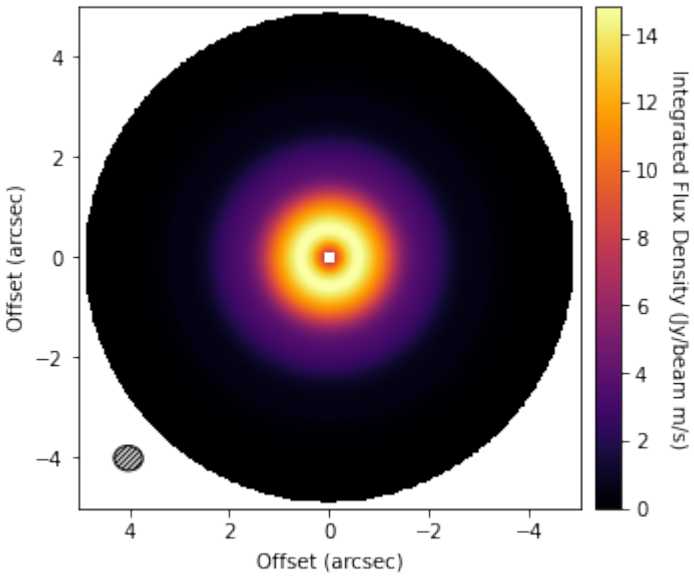
Regions outside the interpolation range are set to `np.nan`, however this can be changed with the inclusion of `interp1d_kw=dict(fill_value=0.0)` which will set all filled values to 0 (or any value you wish).

Sometimes it's also useful to visualize the background model too. This can be done by setting `background_only=True` in the call to `background_residual`, as so:

[9]:
```
background = cube.background_residual(inc=5.0, PA=151.0, background_only=True)
```

```
WARNING: Attached data is not 3D, so shifting cannot be applied.
         Reverting to standard azimuthal averaging; will ignore `unit` argument.
```

[10]:
```
fig, ax = plt.subplots(constrained_layout=True)
im = ax.imshow(background, origin='lower', extent=cube.extent, cmap='inferno')
cb = plt.colorbar(im, pad=0.02,)
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
cb.set_label('Integrated Flux Density (Jy/beam m/s)', rotation=270, labelpad=13)
cube.plot_beam(ax=ax)
```

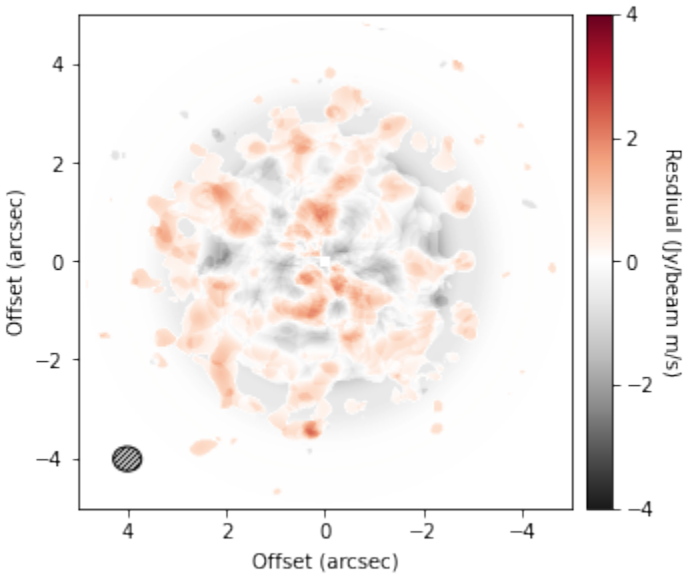


Note that you can also include the standard `GoFish` masking properties for defining the region you wish to use to make your background model. For example, only the red-shifted side of the disk:

[11]:
```
residual_mask = cube.background_residual(inc=5.0, PA=151.0, PA_min=-90.0, PA_max=90.0)
```

```
WARNING: Attached data is not 3D, so shifting cannot be applied.
         Reverting to standard azimuthal averaging; will ignore `unit` argument.
```

[12]:
```
fig, ax = plt.subplots(constrained_layout=True)
im = ax.imshow(residual, origin='lower', extent=cube.extent, cmap='RdGy_r', vmin=-4,␣
↪vmax=4)
cb = plt.colorbar(im, pad=0.02, ticks=np.arange(-4, 5, 2))
ax.set_xlabel('Offset (arcsec)')
ax.set_ylabel('Offset (arcsec)')
cb.set_label('Resdiual (Jy/beam m/s)', rotation=270, labelpad=13)
cube.plot_beam(ax=ax)
```



However, for the broadly azimuthally symmetric TW Hya, this doesn't make a noticable difference.

Python Module Index

g

gofish.gofish.imagecube, 11

Index

A

average_spectrum() (*gofish.imagecube method*), 11

B

background_residual() (*gofish.imagecube method*), 24
beams_per_pix (*gofish.imagecube attribute*), 28

C

convolve_with_beam() (*gofish.imagecube method*), 31
correct_PB() (*gofish.imagecube method*), 29
cross_section() (*gofish.imagecube method*), 31

D

disk_coords() (*gofish.imagecube method*), 25
disk_to_sky() (*gofish.imagecube method*), 27

E

estimate_RMS() (*gofish.imagecube method*), 29
estimate_uncertainty() (*gofish.imagecube static method*), 13
extent (*gofish.imagecube attribute*), 30

F

find_center() (*gofish.imagecube method*), 20
FOV (*gofish.imagecube attribute*), 28
frequency() (*gofish.imagecube method*), 28
frequency_offset() (*gofish.imagecube method*), 29

G

get_annulus() (*gofish.imagecube method*), 22
get_mask() (*gofish.imagecube method*), 23
get_spectrum() (*gofish.imagecube method*), 30
get_vlos() (*gofish.imagecube method*), 21
gofish.gofish.imagecube (*module*), 11

I

imagecube (*class in gofish*), 11
integrated_spectrum() (*gofish.imagecube method*), 14

J

jybeam_to_Tb() (*gofish.imagecube method*), 29
jybeam_to_Tb_RJ() (*gofish.imagecube method*), 29

K

keplerian() (*gofish.imagecube method*), 19
keplerian_mask() (*gofish.imagecube method*), 27

P

pix_per_beam (*gofish.imagecube attribute*), 28
plot_beam() (*gofish.imagecube method*), 33
plot_center() (*gofish.imagecube method*), 32
plot_mask() (*gofish.imagecube method*), 34
plot_maximum() (*gofish.imagecube method*), 34
plot_surface() (*gofish.imagecube method*), 33
plot_teardrop() (*gofish.imagecube method*), 33
print_beam() (*gofish.imagecube method*), 28
print_RMS() (*gofish.imagecube method*), 29

R

radial_profile() (*gofish.imagecube method*), 17
radial_sampling() (*gofish.imagecube method*), 24
radial_spectra() (*gofish.imagecube method*), 15
restframe_frequency_to_velocity() (*gofish.imagecube method*), 27
rms (*gofish.imagecube attribute*), 30
rotate_image() (*gofish.imagecube method*), 30

S

shift_image() (*gofish.imagecube method*), 29
shifted_cube() (*gofish.imagecube method*), 19
sky_to_disk() (*gofish.imagecube method*), 26
spectral_resolution() (*gofish.imagecube method*), 27

spiral_coords() (*gofish.imagecube method*), 32

T

Tb_to_jybeam() (*gofish.imagecube method*), 29
Tb_to_jybeam_RJ() (*gofish.imagecube method*), 29

V

velocity_resolution() (*gofish.imagecube method*), 27
velocity_to_restframe_frequency() (*gofish.imagecube method*), 27