disksurf

Richard Teague, Jane Huang, Charles Law

Nov 10, 2021

CONTENTS

`disksurf` is a package that implements the method described in Pinte et al. (2018) to extract the emission surface from spatially and spectrally resolved observations of molecular emission from a protoplanetary disk. The package provides a suite of convenience functions to not only extract an emission surface, but also fit commonly used analytical forms, and over plot isovelocity contours on channel maps to verify the correct surface was extracted.

CHAPTER

ONE

INSTALLATION

To install `disksurf` we'd recommend using PyPI:

```
pip install disksurf
```

Alternatively you can clone the repository and install that version.

```
git clone https://github.com/richteague/disksurf.git
cd disksurf
pip install .
```

To guide you through how to use `disksurf` we've created a tutorial using data from the DSHARP Large Program. This tutorial also serves as a test that the software has been installed correctly.

CHAPTER

TWO

SUPPORT

Information for all the functions can be found in the API documentation. If you are having issues, please open a issue on the GitHub page.

2.1 Extracting an Emission Surface

In this notebook, we'll walk through how you can extract an emission surface from an image cube following the method presented in Pinte et al. (2018). For this example, we'll use the DSHARP ^{12}CO J=2-1 data for the disk around HD 163296, presented in Isella et al. (2018) and available from the DSHARP website, or using the following bit of code. Note that this is a bit of a chunky file (~1.6GB), so make sure you have space for it!

[1]:
```
import os
if not os.path.exists('HD163296_CO.fits'):
    from wget import download
    download('https://almascience.eso.org/almadata/lp/DSHARP/images/HD163296_CO.fits')
```

First, we start off with the standard imports.

[2]:
```
from disksurf import observation
import matplotlib.pyplot as plt
import numpy as np
```

Now we want to load up the data: simply point `observation` to the FITS cube we want to use. If the cube is large (as is the case for the DSHARP dataset), you can provide a field of view, `FOV` in arcsecs, and a velocity range, `velocity_range` in $\mathrm{m\,s^{-1}}$, which will cut down the data array to only the region you're interested in. This can save a lot of computational time later on.

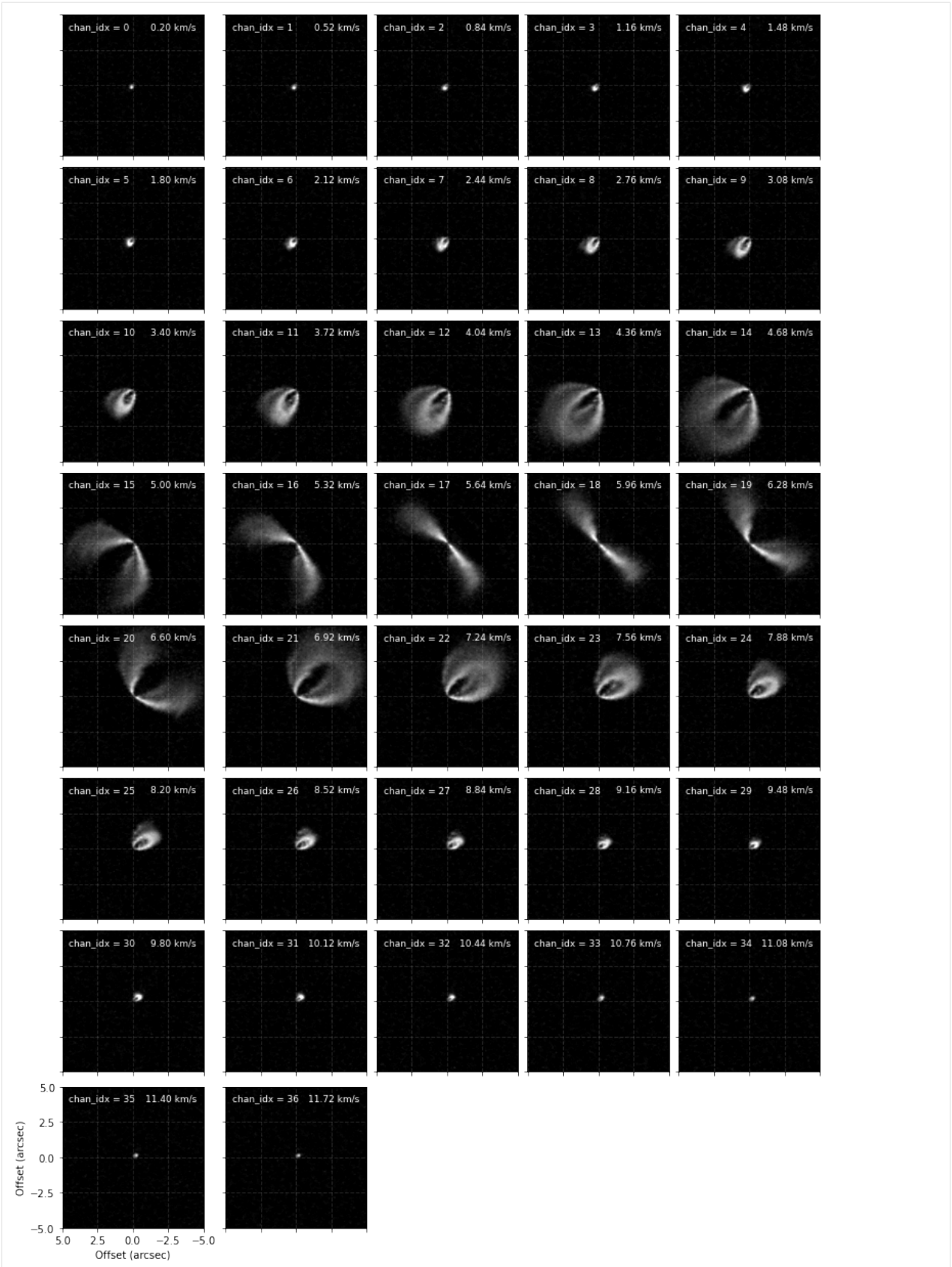
[3]:
```
cube = observation('HD163296_CO.fits', FOV=10.0, velocity_range=[0.0, 12e3])
```

With these commands, we cut out a region that is $\pm 5''$ around the image center, and taking channels between $0\ \mathrm{km\,s^{-1}}$ and $+12\ \mathrm{km\,s^{-1}}$.

Although we've already cut down the velocity range through the `velocity_range` argument when we loaded up the data, it's better to select only a channel range where we can easily see some difference in the emission height. To get a quick idea of emission structure, we can plot all channel maps.

WARNING - If you have a cube with many channels (> 100), this can take a long time!

[4]:
```
cube.plot_channels()
```


chan_idx = 0 0.20 km/s
chan_idx = 1 0.52 km/s
chan_idx = 2 0.84 km/s
chan_idx = 3 1.16 km/s
chan_idx = 4 1.48 km/s
chan_idx = 5 1.80 km/s
chan_idx = 6 2.12 km/s
chan_idx = 7 2.44 km/s
chan_idx = 8 2.76 km/s
chan_idx = 9 3.08 km/s
chan_idx = 10 3.40 km/s
chan_idx = 11 3.72 km/s
chan_idx = 12 4.04 km/s
chan_idx = 13 4.36 km/s
chan_idx = 14 4.68 km/s
chan_idx = 15 5.00 km/s
chan_idx = 16 5.32 km/s
chan_idx = 17 5.64 km/s
chan_idx = 18 5.96 km/s
chan_idx = 19 6.28 km/s
chan_idx = 20 6.60 km/s
chan_idx = 21 6.92 km/s
chan_idx = 22 7.24 km/s
chan_idx = 23 7.56 km/s
chan_idx = 24 7.88 km/s
chan_idx = 25 8.20 km/s
chan_idx = 26 8.52 km/s
chan_idx = 27 8.84 km/s
chan_idx = 28 9.16 km/s
chan_idx = 29 9.48 km/s
chan_idx = 30 9.80 km/s
chan_idx = 31 10.12 km/s
chan_idx = 32 10.44 km/s
chan_idx = 33 10.76 km/s
chan_idx = 34 11.08 km/s
chan_idx = 35 11.40 km/s
chan_idx = 36 11.72 km/s
5.0
2.5
0.0
−2.5
−5.0
Offset (arcsec)
5.0 2.5 0.0 −2.5 −5.0
Offset (arcsec)

From these we can see that between channels 9 and 25 we can really nicely distinguish the front and back sides of the disk and would be the best channels to include for the fitting. However, in the central channels around the systemic velocity of $5.7\,\mathrm{km\,s^{-1}}$ the back side is hidden behind the front side so we'd like to avoid those. Instead of just provide a minimum and maximum channel range, e.g., `chans=(9, 25)`, we can provide a list of min/max channel tuples:

[5]:
```
chans = [(9, 15), (19, 25)]
```

This will include channels between 9 and 15 and then again from 19 to 25.

We also want some idea of the geometry of the system, namely the inclination, `inc` in degrees, and position angle, `PA` in degrees of the disk. Adopting those from fits of the continuum emission are usually just fine here, but there are two things to take note of.

- In both `disksurf` and the `GoFish` package on which it is built, the inclination is defined from -90^o to $+90^o$, with positive inclinations representing a disk that is rotating in a clockwise direction, while a negative inclination describes a disk rotating in a counter-clockwise direction. The `GoFish` documentation shows a nice example of how these disks may look.
- Again, in both `disksurf` and `GoFish`, the position angle is measured to the red-shifted major axis of the disk in an anticlockwise direction from North. This may result in a $\pm 180^o$ difference from those derived from fits to the continuum.

We also want to specify any offset of the source center relative to the image center with `(x0, y0)` in units of arcsecs. If the image has been centered, then these should be zero. We take these offsets from the continuum fitting described in Huang et al. (2018; DSHARP II).

[6]:
```
x0 = -2.8e-3 # arcsec
y0 = 7.7e-3  # arcsec
inc = 46.7   # deg
PA = 312.0   # deg
```

With these values to hand, we can apply the emission surface extraction method to the entire data cube with the `get_emission_surface` function. This will return a `surface` class which contains the extracted emission surface. It will have contain arrays of the radial and vertical coordinates, `r` and `z` in arcsecs, the emission intensity at that point, `I`, in $\mathrm{Jy\,beam^{-1}}$ (also converted to a brightness temperature, `T`, using the full Planck law) and the velocity of the channel that this point was extracted from, `v` in $\mathrm{m\,s^{-1}}$.

When calling this we also include a `smooth=1.0` argument. This will convolve each vertical column with a Gaussian kernel with a FWHM that is `smooth` times the beam major FWHM. This is useful to remove some of the noise in the data. Some data may require more smoothing than others and will probably require some playing around to find the best value.

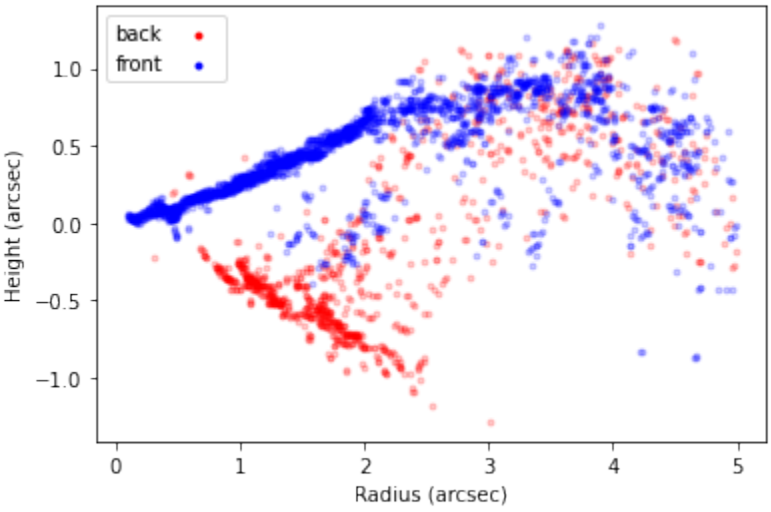
[7]:
```
surface = cube.get_emission_surface(x0=x0, y0=y0, inc=inc,
                                    PA=PA, chans=chans,
                                    smooth=1.0)
```

```
Centering data cube...
Rotating data cube...
Detecting peaks...
Done!
```

[ ]:
```

```

A quick way to see your extracted surface is with the `plot_surface` function.

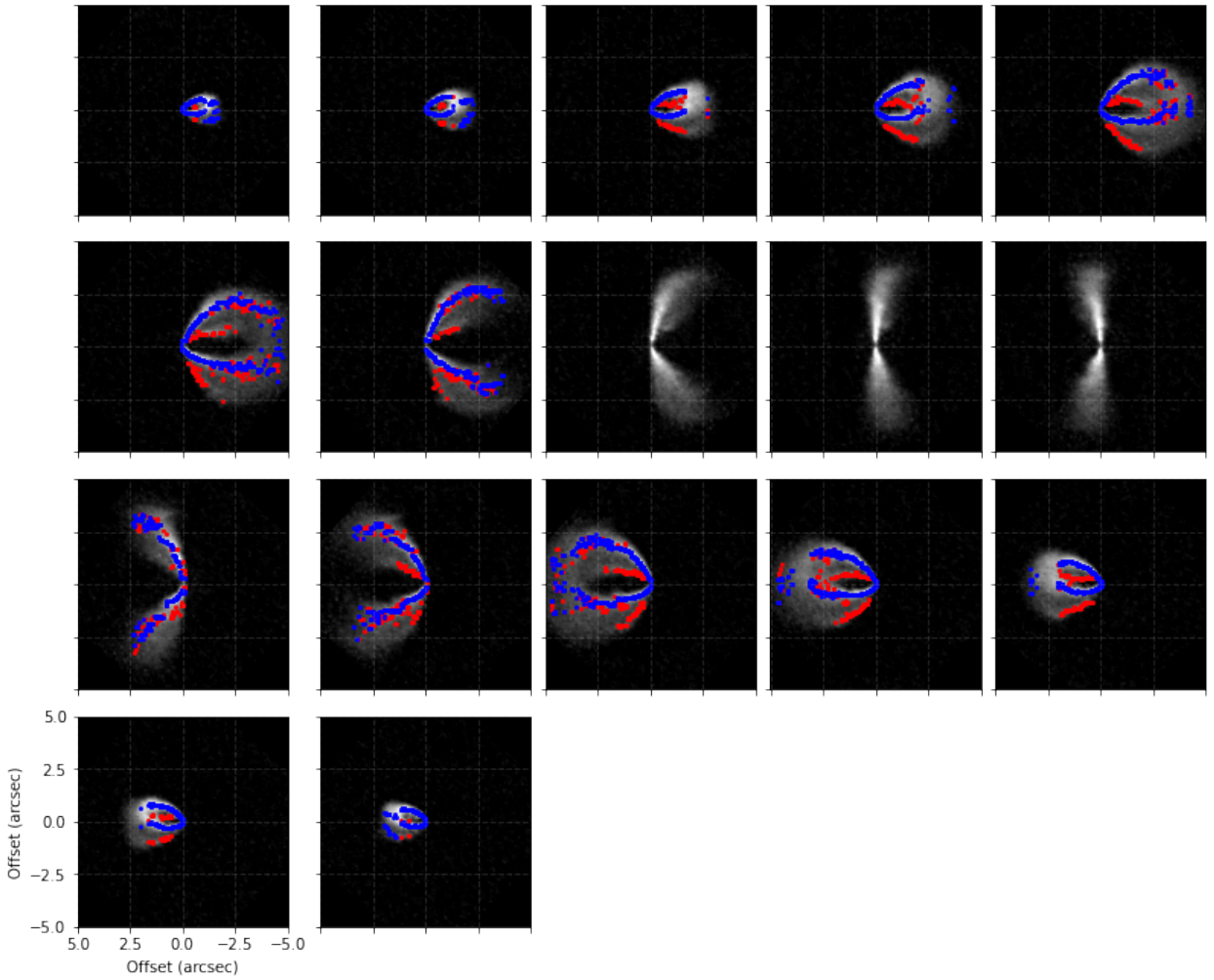
[8]:
```
surface.plot_surface()
```



This will plot all the extracted points, with the 'front' side of the disk shown in blue and the 'back' side of the disk in red. In the inner region of the disk, within 2″ we see we are able to well recover both sides of the disk. However, in the outer disk we see that the 'front' and 'back' side are overlapping. This is because we no longer have four distinct peaks (two for each surface), but just two, as the two sides are no longer spatially resolved.

To better see this, we can use this extracted `surface` to plot where the peaks were found.

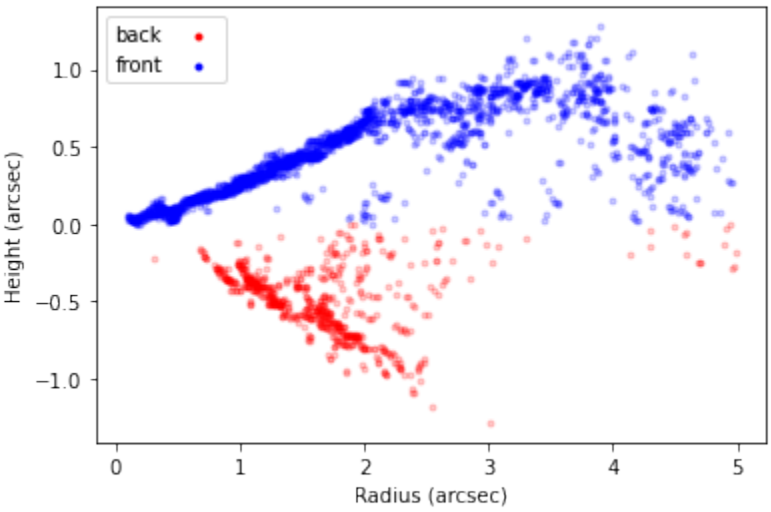
[9]:
```
cube.plot_peaks(surface=surface)
```



Note here that the channels around the systemic velocity are skipped because of how we specified `chans`.

We can now use our physical intuition to remove some of these noisy points. The most straightforward and easy-to-defend cut is simply making cuts in z / r: we know that the aspect ratio of the disk can't be greater than one (at least in most cases), and it shouldn't be negative. We can use the `mask_surface` function to apply these cuts to the `zr` attribute.

[10]:
```
surface.mask_surface(side='both', min_zr=0.0, max_zr=1.0, reflect=True)
surface.plot_surface()
```

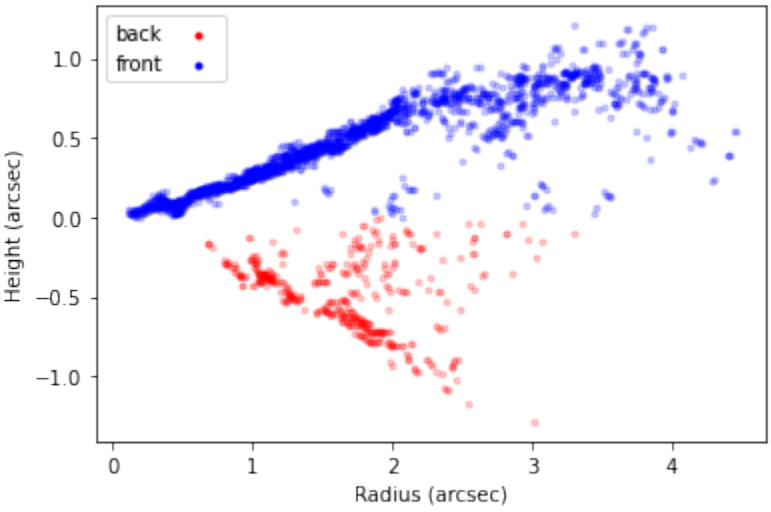


In the above example we've also specified `side='both'` to apply the mask to both sides of the disk (alternatively, we can set `side='front'` or `side='back'` to mask each side separately). We've also included `reflect=True` which will reflect the back side of the disk about the disk midplane for the case of masking, meaning that you can more easily apply the same mask. If we wanted to do this to each separately we could write:

```
surface.mask_surface(side='front', min_zr=0.0, max_zr=1.0)
surface.mask_surface(side='back', min_zr=-1.0, max_zr=0.0)
```

In addition to geometrical cuts, we make a signal-to-noise cut. Let's remove all points where the SNR is less than 10. By default the RMS is calculated as the RMS of the pixels in the first 5 and last 5 channels using the `estimate_RMS` function (this result is stored in the `surface.rms` attribute). If you would like to provide a different RMS value you can use the `RMS` value in mask_surface.

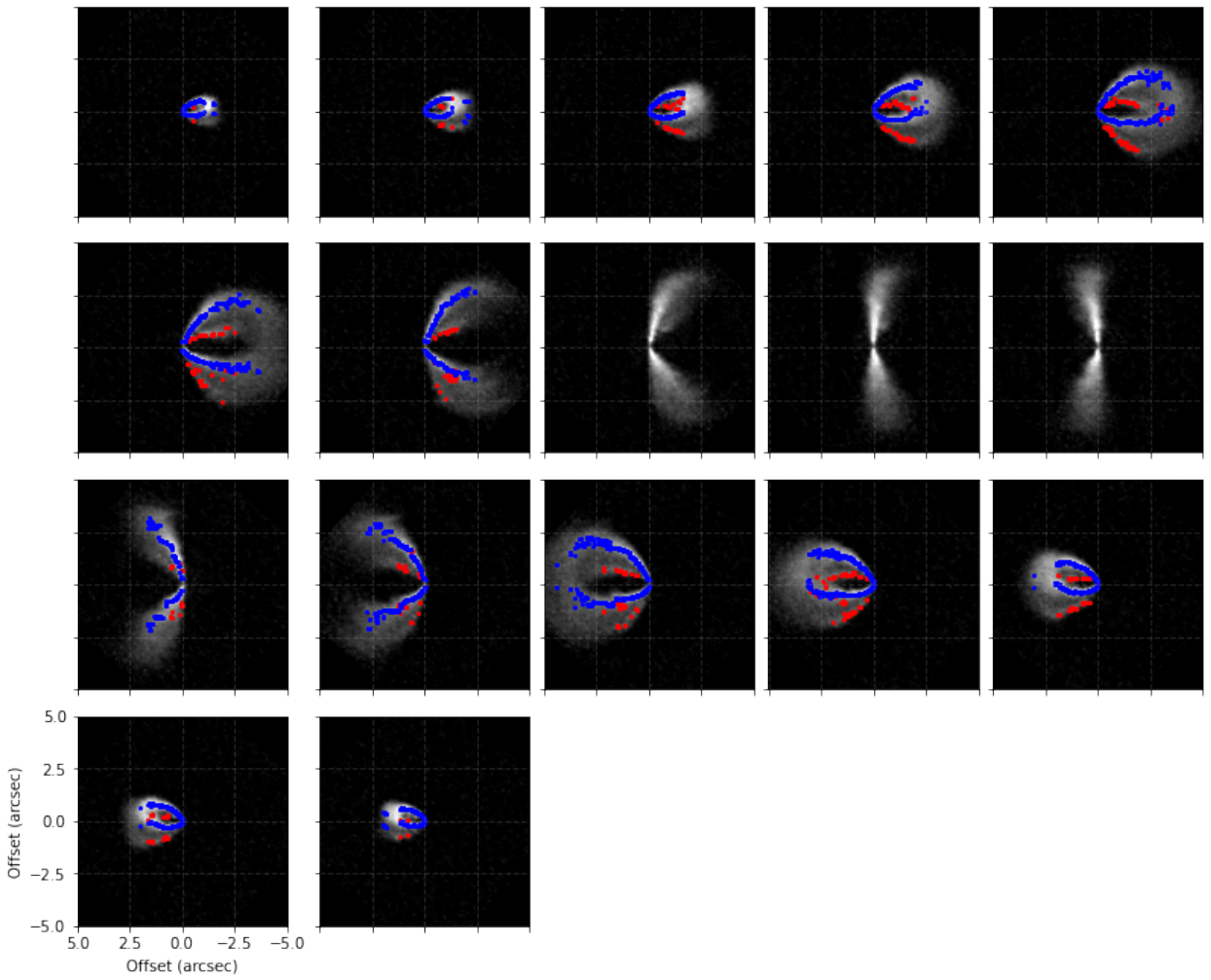
[11]:
```
surface.mask_surface(side='both', min_SNR=10.0)
surface.plot_surface()
```



NOTE - At any point you can use `surface.reset_mask()` to rest the masking you've applied. Most functions also accept the `masked` argument which, if set to `False` will use the unmasked points instead.

We can now see that this masking has removed some of those pesky points when plotting the peaks.

[12]:
```
cube.plot_peaks(surface=surface)
```



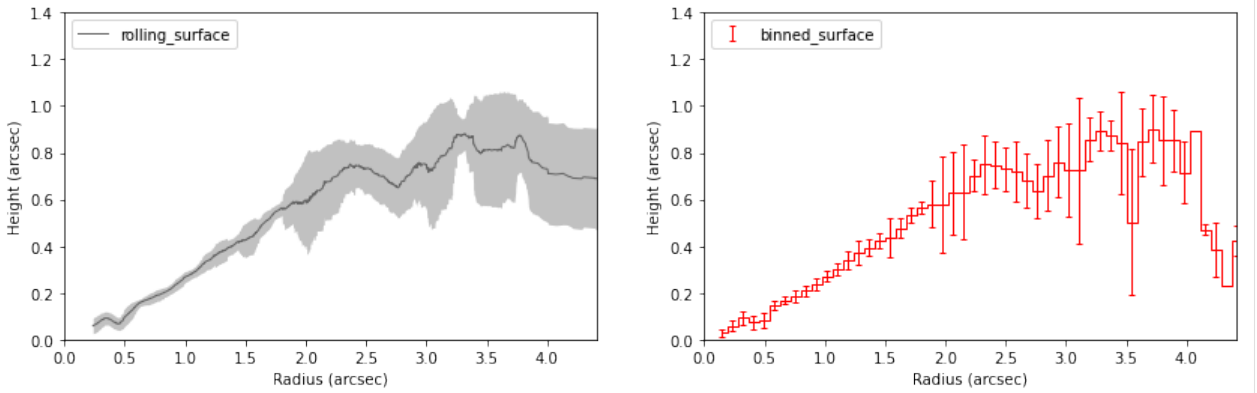
Now that we're happy with the surface, we can extract it to use it in other functions. Two functions provide this sort of functionality: `surface.binned_surface`, which will return the emission surface binned onto a radial grid, and `surface.rolling_surface` which will return the rolling average.

[13]:
```
fig, [ax1, ax2] = plt.subplots(figsize=(14, 4), ncols=2)

r, z, dz = surface.rolling_surface()
ax1.fill_between(r, z - dz, z + dz, lw=0.0, color='0.4', alpha=0.4)
ax1.plot(r, z, lw=1.0, color='0.4', label='rolling_surface')

r, z, dz = surface.binned_surface()
ax2.step(r, z, where='mid', c='r', lw=1.0)
ax2.errorbar(r, z, yerr=dz, c='r', lw=1.0, capsize=2.0,
             capthick=1.0, fmt=' ', label='binned_surface')

for ax in fig.axes:
    ax.legend(loc=2)
    ax.set_xlim(0, r.max())
```

(continues on next page)

(continued from previous page)

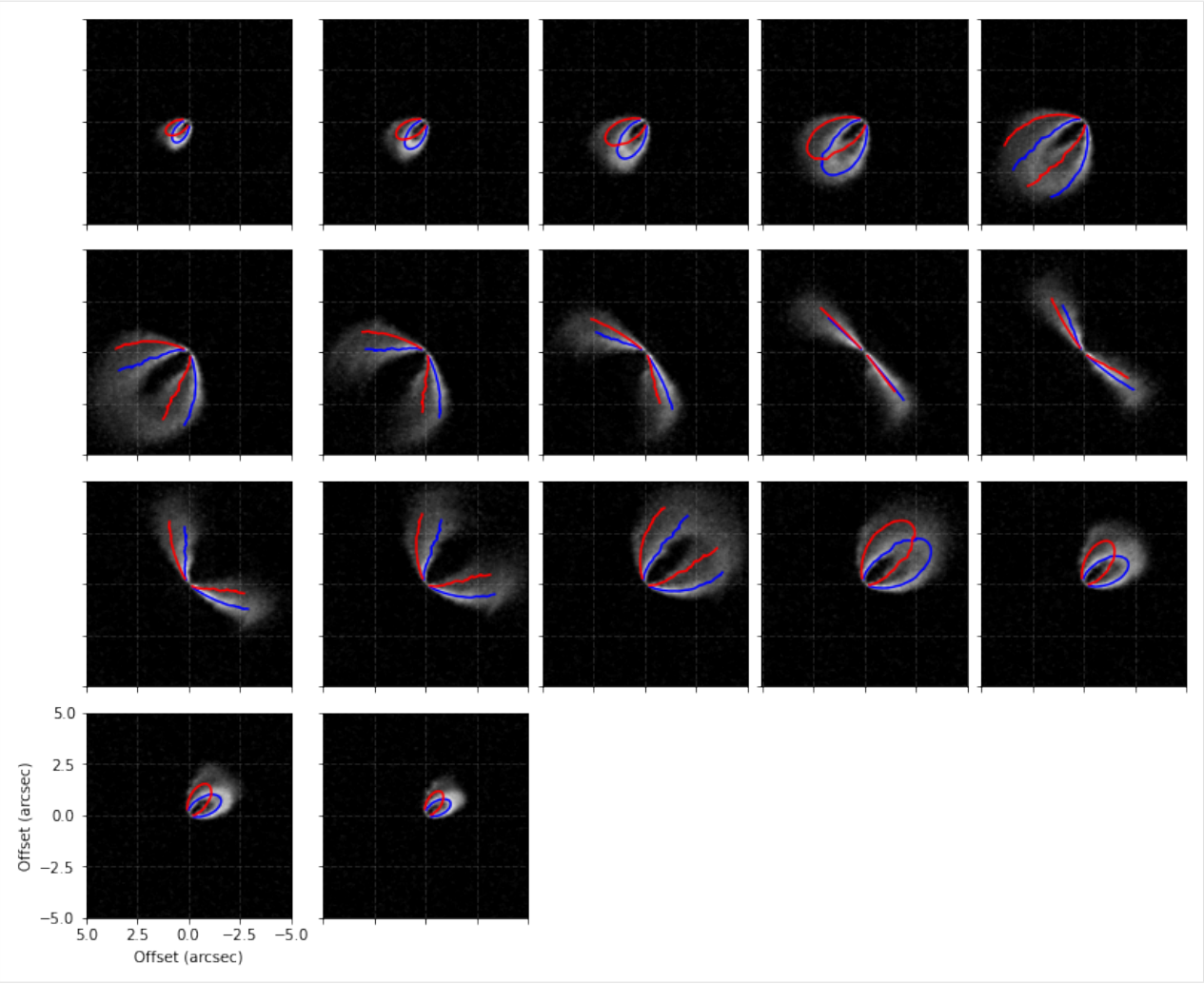
```
    ax.set_ylim(0, 1.4)
    ax.set_xlabel('Radius (arcsec)')
    ax.set_ylabel('Height (arcsec)')
```



Note that these are both wrappers for the functions `surface.binned_parameter` and `surface.rolling_parameter` which will radially bin or calculate the rolling average for a specific parameter, such as the height, intensity or brightness temperature.

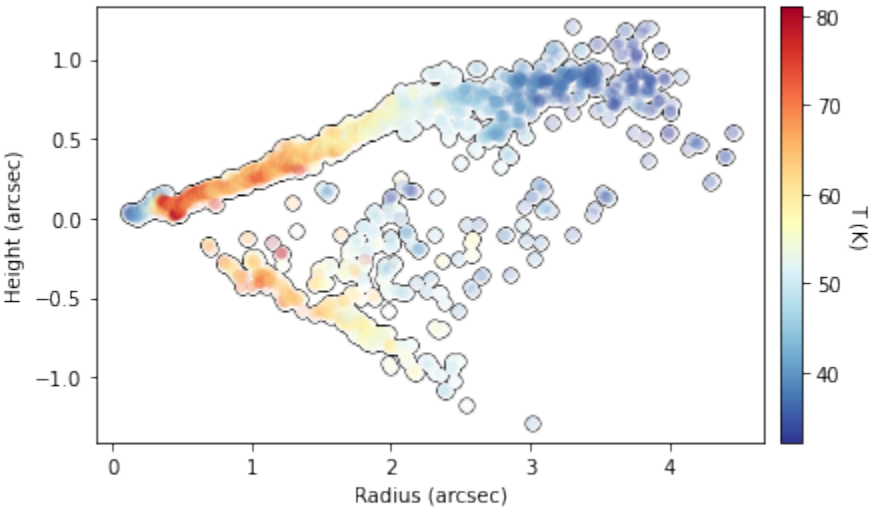
We can also use this surface to plot isovelocity contours on our channel maps and see how they fare. For this, we need to provide information about the velocity field, namely the stellar mass in units of $M_{\odot}$, `mstar`, the distance to the source in parsecs, `dist`, and the systemic velocity in $\mathrm{m\,s^{-1}}$, `vlsr`.

[14]:
```
cube.plot_isovelocities(surface=surface, mstar=2.0, dist=101.0, vlsr=5.7e3, side='both')
```



Finally, we can also use the peak intensity, converted to a brightness temperature in Kelvin using the full Planck function (which depends on the beam size, hence the reason this function requires `observation` to run), to estimate the gas temperature structure. Note that for the brightness temperatures to truely represent the gas temperatures the lines must be optically thick and the continuum emission must also be included. See Weaver et al. (2018) for more details.

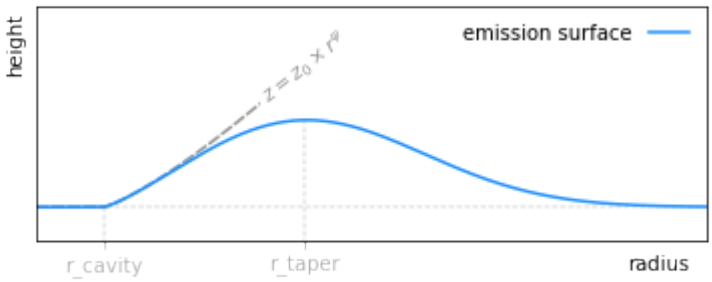
[15]:
```
fig = cube.plot_temperature(surface=surface)
```



We can also fit an exponentially tapered powerlaw surface of the form,

$$z(r) = z_0 \left(\frac{r - r_{\rm cavity}}{r_0}\right)^{\psi} \times \exp\left(-\left[\frac{r - r_{\rm cavity}}{r_{\rm taper}}\right]^{q_{\rm taper}}\right),$$

where r_0, z_0 and ψ describe the powerlaw function, $r_{\rm taper}$ and $q_{\rm taper}$ describe the exponential taper, and $r_{\rm cavity}$ allows for any inner cavity. This is the analytical form that is used by default within `GoFish` and demonstrated in the figure below.



There are two methods to fit this surface: `fit_mission_surface` which uses `scipy.optimize.curve_fit`, or `fit_emission_surface_MCMC` which wraps emcee, the Affine Invariant Markov chain Monte Carlo (MCMC) Ensemble sampler. Both work in a very similar way. By default, these functions fit the emission surface in units of arcseconds, unless the source distance, `dist`, is specified in parsecs, which converts all distances to au. The reference radius, r_0, defaults to $1''$ or 100 au depending on which units are being used, but this can be changed through the `r0` argument.

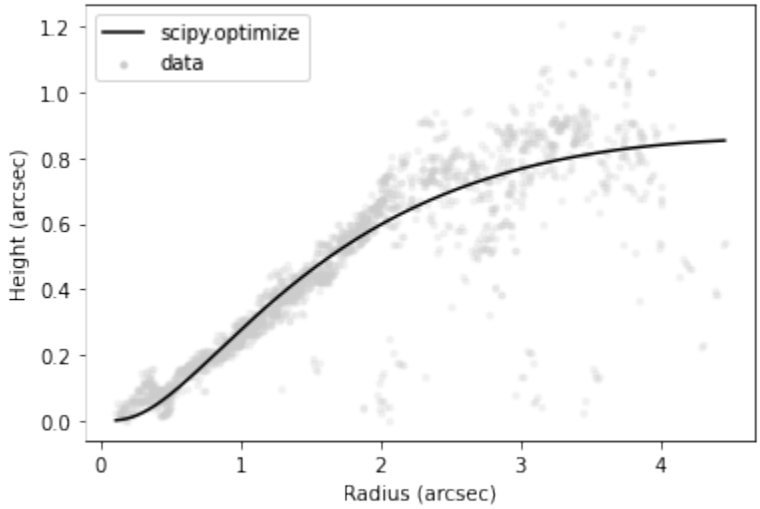
[16]:
```
# plot the surface
fig = surface.plot_surface(side='front', return_fig=True)

# set the blue points to gray
for i in range(3):
    fig.axes[0].get_children()[i].set_facecolor('0.8')
    fig.axes[0].get_children()[i].set_edgecolor('0.8')
fig.axes[0].get_children()[1].set_label('data')

# fit an exponentitally tapered power law model and plot
```

(continues on next page)

(continued from previous page)

```
r_mod, z_mod = surface.fit_emission_surface(side='front', return_model=True,
                                            tapered_powerlaw=True,
                                            include_cavity=False)
fig.axes[0].plot(r_mod, z_mod, color='k', lw=1.5, label='scipy.optimize')

# update legend
fig.axes[0].legend()
```

[16]:
```
<matplotlib.legend.Legend at 0x7fe1f9293310>
```



2.2 API

2.2.1 observation

This class is built upon the *imagecube* class from GoFish, so contains all functionality described there.

class disksurf.**observation**(**args: Any*, ***kwargs: Any*)

Wrapper of a GoFish imagecube class containing the emission surface extraction methods.

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 the minimum and maximum velocity in [m/s] to cut the cube down to.

get_emission_surface(*inc*, *PA*, *x0=0.0*, *y0=0.0*, *chans=None*, *r_min=None*, *r_max=None*, *smooth=None*, *nsigma=None*, *min_SNR=5*, *detect_peaks_kwargs=None*, *keplerian_mask_kwargs=None*, *force_opposite_sides=True*)

Implementation of the method described in Pinte et al. (2018). There are several pre-processing options to help with the peak detection.

Parameters

- **inc** (*float*) – Disk inclination in [degrees].

- **PA** (*float*) – Disk position angle in [degrees].
- **x0** (*optional[float]*) – Disk offset along the x-axis in [arcsec].
- **y0** (*optional[float]*) – Disk offset along the y-axis in [arcsec].
- **chans** (*optional[list]*) – First and last channels to include in the inference.
- **r_min** (*optional[float]*) – Minimuim radius in [arcsec] of values to return. Default is all possible values.
- **r_max** (*optional[float]*) – Maximum radius in [arcsec] of values to return. Default is all possible values.
- **smooth** (*optional[float]*) – Prior to detecting peaks, smooth the pixel column with a Gaussian kernel with a FWHM equal to `smooth * cube.bmaj`. If `smooth == 0` then no smoothing is applied.
- **return_sorted** (*optional[bool]*) – If `True`, return the points ordered in increasing radius.
- **smooth_threshold_kwargs** (*optional[dict]*) – Keyword arguments passed to `smooth_threshold`.
- **detect_peaks_kwargs** (*optional[dict]*) – Keyword arguments passed to `detect_peaks`. If any values are duplicated from those required for `get_emission_surface`, they will be overwritten.
- **force_opposite_sides** (*optional[bool]*) – Whether to assert that all pairs of peaks have one on either side of the major axis. By default this is `True` which is a more conservative approach but results in a lower sensitivity in the outer disk.

Returns A `disksurf.surface` instance containing the extracted emission surface.

keplerian_mask(*x0*, *y0*, *inc*, *PA*, *mstar*, *vlsr*, *dist*, *r_min=0.0*, *r_max=None*, *width=2.0*, *smooth=None*, *tolerance=0.0001*)

Produce a Keplerian mask for the data.

Parameters

- **x0** (*float*) – Disk offset along the x-axis in [arcsec].
- **y0** (*float*) – Disk offset along the y-axis in [arcsec].
- **inc** (*float*) – Disk inclination in [degrees].
- **PA** (*float*) – Disk position angle in [degrees].
- **mstar** (*float*) – Stellar mass in [Msun].
- **vlsr** (*float*) – Systemic velocity in [m/s].
- **dist** (*float*) – Source distance in [pc].
- **r_min** (*optional[float]*) – Inner radius in [arcsec].
- **r_max** (*optional[float]*) – Outer radius in [arcsec].
- **width** (*optional[float]*) – The spectral 'width' of the mask as a fraction of the channel spacing.
- **smooth** (*optional[float]*) – Apply a convolution with a 2D Gaussian with a FWHM of `smooth` to broaden the mask. By default this is four times the beam FWHM. If no smoothing is desired, set this to `0.0`.

- **tolerance** (*optional[float]*) – The minimum value (between 0 and 1) to consider part of the mask after convolution.

Returns A 3D array describing the mask with either 1 or 0.

get_integrated_spectrum(*x0=0.0*, *y0=0.0*, *inc=0.0*, *PA=0.0*, *r_max=None*)

Calculate the integrated spectrum over a specified spatial region. The uncertainty is calculated assuming the spatially correlation is given by elliptical beams.

Parameters

- **x0** (*optional[float]*) – Right Ascension offset in [arcsec].
- **y0** (*optional[float]*) – Declination offset in [arcsec].
- **inc** (*optional[float]*) – Disk inclination in [deg].
- **PA** (*optional[float]*) – Disk position angle in [deg].
- **r_max** (*optional[float]*) – Radius to integrate out to in [arcsec].

Returns The integrated intensity, `spectrum`, and associated uncertainty, `uncertainty`, in [Jy].

plot_channels(*chans=None*, *velocities=None*, *return_fig=False*, *keplerian_mask_kwargs=None*)

Plot the channels within the channel range or velocity range. Only one of `chans` or `velocities` can be specified. If neither is specified, all channels are plotted which may take some time for large data cubes.

Parameters

- **chans** (*optional[tuple]*) – A tuple containing the index of the first and last channel to plot. Cannot be specified if `velocities` is also specified.
- **velocities** (*optional[tuple]*) – A tuple containing the velocity of the first and last channel to plot in [m/s]. Cannot be specified if `chans` is also specified.
- **return_fig** (*optional[bool]*) – Whether to return the Matplotlib figure.
- **keplerian_mask_kwargs** (*optional[dict]*) – A dictionary of arguments to pass to `keplerian_mask` such that the mask outline can be overlaid.

Returns If `return_fig=True`, the Matplotlib figure used for plotting.

plot_integrated_spectrum(*x0=0.0*, *y0=0.0*, *inc=0.0*, *PA=0.0*, *r_max=None*, *return_fig=False*)

Plot the integrated spectrum integrated over a spatial region.

Parameters

- **x0** (*optional[float]*) – Right Ascension offset in [arcsec].
- **y0** (*optional[float]*) – Declination offset in [arcsec].
- **inc** (*optional[float]*) – Disk inclination in [deg].
- **PA** (*optional[float]*) – Disk position angle in [deg].
- **r_max** (*optional[float]*) – Radius to integrate out to in [arcsec].

Returns If `return_fig=True`, the Matplotlib figure used for plotting.

plot_isovelocities(*surface*, *mstar*, *vlsr*, *dist*, *side='both'*, *reflect=True*, *smooth=None*, *return_fig=False*)

Plot the isovelocity contours for the given emission surface. This will use the channels used for the extraction of the emission surface.

Parameters

- **surface** (*surface instance*) – The extracted emission surface.
- **mstar** (*float*) – The stellar mass in [Msun].

- **vlsr** (*float*) – The systemic velocity in [m/s].
- **dist** (*float*) – The source distance in [pc].
- **side** (*optional[str]*) – The emission side to plot, must be either `'both'`, `'front'` or `'back'`.
- **reflect** (*optional[bool]*) – Whether to reflect the back side of the disk about the midplane. Default is `False`.
- **smooth** (*optional[int]*) – If provided, smooth the emission surface with a Hanning kernel with a width of `smooth`. Typically values of 3 or 5 are sufficient for plotting purposes.
- **return_fig** (*optional[bool]*) – If no axis is provided, whether to return the Matplotlib figure. The axis can then be accessed through `fig.axes[0]`.

Returns If `return_fig=True`, the Matplotlib figure used for plotting.

plot_peaks(*surface*, *side='both'*, *return_fig=False*)

Plot the peak locations used to calculate the emission surface on channel maps. This will use the channels used for the extraction of the emission surface.

Parameters

- **surface** (*surface instance*) – The extracted surface returned from `get_emission_surface`.
- **side** (*Optional[str]*) – Side to plot. Must be `'front'`, `'back'` or `'both'`. Defaults to `'both'`.
- **return_fig** (*Optional[bool]*) – Whether to return the Matplotlib figure. Defaults to `True`.

Returns If `return_fig=True`, the Matplotlib figure used for plotting.

plot_temperature(*surface*, *side='both'*, *reflect=False*, *masked=True*, *ax=None*, *return_fig=False*)

Plot the temperature structure using the provided surface instance. Note that the brightness temperature only provides a good measure of the true gas temperature when the lines are optically thick such that $\tau \gtrsim 5$.

Parameters

- **surface** (*surface instance*) – The extracted emission surface.
- **side** (*optional[str]*) – The emission side to plot, must be either `'both'`, `'front'` or `'back'`.
- **reflect** (*optional[bool]*) – Whether to reflect the back side of the disk about the midplane. Default is `False`.
- **masked** (*optional[bool]*) – Whether to plot the masked points, the default, or all extracted points.
- **ax** (*optional[axes instance]*) – The Matplolib axis to use for plotting. If none is provided, one will be generated. If an axis is provided, the same color scaling will be used.
- **return_fig** (*optional[bool]*) – If no axis is provided, whether to return the Matplotlib figure. The axis can then be accessed through `fig.axes[0]`.

Returns If `return_fig=True`, the Matplotlib figure used for plotting.

2.2.2 surface

The `surface` class is returned from the *get_emission_surface()* function and was not designed to be created by a user (hence the rather long list of variables required to instantiate the class).

class disksurf.**surface**(*r_f*, *z_f*, *I_f*, *T_f*, *v*, *x*, *y_n*, *y_f*, *r_b*, *z_b*, *I_b*, *T_b*, *y_n_b*, *y_f_b*, *chans*, *rms*, *x0*, *y0*, *inc*, *PA*, *r_min*, *r_max*, *data*)

A container for the emission surface returned by `detect_peaks`. This class has been designed to be created by the `get_emission_surface` function and not by the user.

Parameters

- **r_f** (*array*) – Radial position of the front surface in [arcsec].
- **z_f** (*array*) – Vertical position of the front surface in [arcsec].
- **I_f** (*array*) – Intensity along the front surface in [Jy/beam].
- **T_f** (*array*) – Brightness temperature along the front surface in [K].
- **v** (*array*) – Velocity in [km/s].
- **x** (*array*) – Distance along the major axis the point was extracted in [arcsec].
- **y_n** (*array*) – Distance along the minor axis of the near peak for the front surface in [arcsec].
- **y_f** (*array*) – Distance along the minor axis of the far peak for the front surface in [arcsec].
- **r_b** (*array*) – Radial position of the back surface in [arcsec].
- **z_b** (*array*) – Vertical position of the back surface in [arcsec].
- **I_b** (*array*) – Intensity along the back surface in [Jy/beam].
- **T_b** (*array*) – Brightness temperature along the back surface in [K].
- **y_n_b** (*array*) – Distance along the minor axis of the near peak for the back surface in [arcsec].
- **y_f_b** (*array*) – Distance along the minor axis of the far peak for the back surface in [arcsec].
- **chans** (*tuple*) – A tuple of the first and last channels used for the emission surface extraction.
- **rms** (*float*) – Noise in the cube in [Jy/beam].
- **x0** (*float*) – Right ascencion offset used in the emission surface extraction in [arcsec].
- **y0** (*float*) – Declination offset used in the emission surface extraction in [arcsec].
- **inc** (*float*) – Inclination of the disk used in the emission surface extraction in [deg].
- **PA** (*float*) – Position angle of the disk used in the emission surface extraction in [deg].
- **r_min** (*float*) – Minimum disk-centric radius used in the emission surface extraction in [arcsec].
- **r_max** (*array*) – Maximum disk-centric radius used in the emission surface extraction in [arcsec].
- **data** (*array*) – The data used to extract the emission surface in [Jy/beam].

r(*side='front'*, *masked=True*)

Radial cylindrical coordinate in [arcsec].

Parameters

- **side** (*optional[str]*) – Side of the disk. Must be `'front'`, `'back'` or `'both'`. Defaults to `'both'`.
- **masked** (*optional[bool]*) – Whether to return only the masked points, the default, or all points.

Returns Radial cylindrical coordinates in [arcsec].

z(*side='front'*, *reflect=False*, *masked=True*)
Vertical cylindrical coordinate in [arcsec].

Parameters

- **side** (*optional[str]*) – Side of the disk. Must be `'front'`, `'back'` or `'both'`. Defaults to `'both'`.
- **reflect** (*optional[bool]*) – Whether to reflect the backside points about the midplane. Defaults to `False`.
- **masked** (*optional[bool]*) – Whether to return only the masked points, the default, or all points.

Returns Vertical cylindrical coordinate in [arcsec].

I(*side='front'*, *masked=True*)
Intensity at the (r, z) coordinate in [Jy/beam].

Parameters

- **side** (*optional[str]*) – Side of the disk. Must be `'front'`, `'back'` or `'both'`. Defaults to `'both'`.
- **masked** (*optional[bool]*) – Whether to return only the masked points, the default, or all points.

Returns Intensity at the (r, z) coordinate in [Jy/beam].

T(*side='front'*, *masked=True*)
Brightness temperature at the (r, z) coordinate in [K].

Parameters

- **side** (*optional[str]*) – Side of the disk. Must be `'front'`, `'back'` or `'both'`. Defaults to `'both'`.
- **masked** (*optional[bool]*) – Whether to return only the masked points, the default, or all points.

Returns Brightness temperature at the (r, z) coordinate in [K].

v(*side='front'*, *masked=True*)
Velocity that the (r, z) coordinate was extracted at in [m/s].

Parameters

- **side** (*optional[str]*) – Side of the disk. Must be `'front'`, `'back'` or `'both'`. Defaults to `'both'`.
- **masked** (*optional[bool]*) – Whether to return only the masked points, the default, or all points.

Returns Velocity that the (r, z) coordinate was extracted at in [m/s].

x(*side='front'*, *masked=True*)
RA offset that the (r, z) coordinate was extracted in [arcsec].

Parameters

- **side** (*optional[str]*) – Side of the disk. Must be `'front'`, `'back'` or `'both'`. Defaults to `'both'`.
- **masked** (*optional[bool]*) – Whether to return only the masked points, the default, or all points.

Returns RA offset that the (r, z) coordinate was extracted in [arcsec].

y(*side='front'*, *edge='near'*, *masked=True*)

Dec offset that the (r, z) coordinate was extracted in [arcsec].

Parameters

- **side** (*optional[str]*) – Side of the disk. Must be `'front'`, `'back'` or `'both'`. Defaults to `'both'`.
- **edge** (*optional[str]*) – Which of the edges to return, either the `'near'` or `'far'` edge.
- **masked** (*optional[bool]*) – Whether to return only the masked points, the default, or all points.

Returns Dec offset that the (r, z) coordinate was extracted in [arcsec].

zr(*side='front'*, *reflect=True*, *masked=True*)

Inverse aspect ratio (height divided by radius) of the emission surface.

Parameters

- **side** (*optional[str]*) – Side of the disk. Must be `'front'`, `'back'` or `'both'`. Defaults to `'both'`.
- **reflect** (*optional[bool]*) – Whether to reflect the backside points about the midplane. Defaults to `False`.
- **masked** (*optional[bool]*) – Whether to return only the masked points, the default, or all points.

Returns Inverse aspect ratio of the emission surface.

SNR(*side='front'*, *masked=True*)

Signal-to-noise ratio for each coordinate.

Parameters

- **side** (*optional[str]*) – Side of the disk. Must be `'front'`, `'back'` or `'both'`. Defaults to `'both'`.
- **masked** (*optional[bool]*) – Whether to return only the masked points, the default, or all points.

Returns Signal-to-noise ratio for each coordinate.

reset_mask(*side='both'*)

Reset the mask.

Parameters **side** (*optional[str]*) – Side of the disk. Must be `'front'`, `'back'` or `'both'`. Defaults to `'both'`.

mask_surface(*side='front'*, *reflect=False*, *min_r=None*, *max_r=None*, *min_z=None*, *max_z=None*, *min_zr=None*, *max_zr=None*, *min_I=None*, *max_I=None*, *min_v=None*, *max_v=None*, *min_SNR=None*, *max_SNR=None*, *RMS=None*)

Mask the surface based on simple cuts to the parameters.

Parameters

- **min_r** (*optional[float]*) – Minimum radius in [arcsec].
- **max_r** (*optional[float]*) – Maximum radius in [arcsec].
- **min_z** (*optional[float]*) – Minimum emission height in [arcsec].
- **max_z** (*optional[float]*) – Maximum emission height in [arcsec].
- **min_zr** (*optional[float]*) – Minimum z/r ratio.
- **max_zr** (*optional[float]*) – Maximum z/r ratio.
- **min_Inu** (*optional[float]*) – Minumum intensity in [Jy/beam].
- **max_Inu** (*optional[float]*) – Maximum intensity in [Jy/beam].
- **min_v** (*optional[float]*) – Minimum velocity in [m/s].
- **max_v** (*optional[float]*) – Maximum velocity in [m/s].
- **min_snr** (*optional[float]*) – Minimum SNR ratio.
- **max_snr** (*optional[float]*) – Maximum SNR ratio.
- **RMS** (*optional[float]*) – Use this RMS value in place of the `self.rms` value for calculating the SNR masks.

binned_surface(*rvals=None*, *rbins=None*, *side='front'*, *reflect=True*, *masked=True*)

Bin the emisison surface onto a regular grid. This is a simple wrapper to the `binned_parameter` function.

Parameters

- **rvals** (*optional[array]*) – Desired bin centers.
- **rbins** (*optional[array]*) – Desired bin edges.
- **side** (*optional[str]*) – Which 'side' of the disk to bin, must be one of `'both'`', `'front'` or `'back'`.
- **reflect** (*Optional[bool]*) – Whether to reflect the emission height of the back side of the disk about the midplane.
- **masked** (*Optional[bool]*) – Whether to use the masked data points. Default is `True`.

Returns The bin centers, `r`, and the average emission surface, `z`, with the uncertainty, `dz`, given as the bin standard deviation.

binned_parameter(*p*, *rvals=None*, *rbins=None*, *side='front'*, *reflect=True*, *masked=True*)

Bin the provided parameter onto a regular grid. If neither `rvals` nor `rbins` is specified, will default to 50 bins across the radial range of the bins.

Parameters

- **p** (*str*) – Parameter to bin. For example, to bin the emission height, `p='z'`.
- **rvals** (*optional[array]*) – Desired bin centers.
- **rbins** (*optional[array]*) – Desired bin edges.
- **side** (*optional[str]*) – Which 'side' of the disk to bin, must be one of `'both'`', `'front'` or `'back'`.
- **reflect** (*optional[bool]*) – Whether to reflect the emission height of the back side of the disk about the midplane.
- **masked** (*optional[bool]*) – Whether to use the masked data points. Default is `True`.

Returns The bin centers, `r`, and the binned mean, `mu`, and standard deviation, `std`, of the desired parameter.

rolling_surface(*window=0.1*, *side='front'*, *reflect=True*, *masked=True*)

Return the rolling average of the emission surface. As the radial sampling is unevenly spaced the kernel size, which is a fixed number of samples, can vary in the radial range it represents. The uncertainty is taken as the rolling standard deviation.

Parameters

- **window** (*optional[float]*) – Window size in [arcsec].
- **side** (*optional[str]*) – Which 'side' of the disk to bin, must be one of `'both''`, `'front'` or `'back'`.
- **reflect** (*optional[bool]*) – Whether to reflect the emission height of the back side of the disk about the midplane.
- **masked** (*optional[bool]*) – Whether to use the masked data points. Default is `True`.

Returns The radius, `r`, emission height, `z`, and uncertainty, `dz`.

rolling_statistic(*p*, *func=numpy.nanmean*, *window=0.1*, *side='front'*, *reflect=True*, *masked=True*, *remove_NaN=True*)

Return the rolling statistic of the provided parameter. As the radial sampling is unevenly spaced the kernel size, which is a fixed number of samples, can vary in the radial range it represents.

Parameters

- **p** (*str*) – Parameter to appl the rolling statistic to. For example, to use the emission height, `p='z'`.
- **func** (*Optional[callable]*) – The function to apply to the data.
- **window** (*Optional[float]*) – Window size in [arcsec].
- **side** (*Optional[str]*) – Which 'side' of the disk to bin, must be one of `'both''`, `'front'` or `'back'`.
- **reflect** (*Optional[bool]*) – Whether to reflect the emission height of the back side of the disk about the midplane.
- **masked** (*Optional[bool]*) – Whether to use the masked data points. Default is `True`.
- **remove_NaN** (*Optional[bool]*) – Whether to remove the NaNs.

Returns The radius, `r` and the rolling statistic, `s`. All NaNs will have been removed.

fit_emission_surface(*tapered_powerlaw=True*, *include_cavity=False*, *r0=None*, *dist=None*, *side='front'*, *masked=True*, *return_model=False*, *curve_fit_kwargs=None*)

Fit the extracted emission surface with a tapered power law of the form

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

where a single power law profile is recovered when $r_{\rm taper} \rightarrow \infty$, and can be forced using the `tapered_powerlaw=False` argument.

We additionally allow for an inner cavity, $r_{\rm cavity}$, inside which all emission heights are set to zero, and the radial range is shifted such that $r' = r - r_{\rm cavity}$. This can be toggled with the `include_cavity` argument.

The fitting is performed with `scipy.optimize.curve_fit` where the returned uncertainties are the square root of the diagnal components of the covariance maxtrix returned by `curve_fit`. We use the SNR of each point as a weighting in the fit.

Parameters

- **tapered_powerlaw** (*optional[bool]*) – If True, fit the tapered power law profile rather than a single power law function.
- **include_cavity** (*optional[bool]*) – If True, include a cavity in the functional form, inside of which all heights are set to 0.
- **r0** (*optional[float]*) – The reference radius for z_0. Defaults to 1 arcsec, unless dist is provided, then defaults to 100 au.
- **dist** (*optional[float]*) – Convert all distances from [arcsec] to [au] for the fitting. If this is provided, r_ref will change to 100 au unless specified by the user.
- **side** (*optional[str]*) – Which 'side' of the disk to bin, must be one of 'both'', 'front' or 'back'.
- **masked** (*optional[bool]*) – Whether to use the masked data points. Default is True.
- **curve_fit_kwargs** (*optional[dict]*) – Keyword arguments to pass to scipy.optimize.curve_fit.

Returns Best-fit values, popt, and associated uncertainties, copt, for the fits if return_fit=False, else the best-fit model evaluated at the radial points.

fit_emission_surface_MCMC(*r0=None*, *dist=None*, *side='front'*, *masked=True*, *tapered_powerlaw=True*, *include_cavity=False*, *p0=None*, *nwalkers=64*, *nburnin=1000*, *nsteps=500*, *scatter=0.001*, *priors=None*, *returns=None*, *plots=None*, *curve_fit_kwargs=None*, *niter=1*)

Fit the inferred emission surface with a tapered power law of the form

$$z(r) = z_0 \left(\frac{r}{r_0}\right)^{\psi} \times \exp\left(-\left[\frac{r}{r_{\rm taper}}\right]^{q_{\rm taper}}\right)$$

where a single power law profile is recovered when $r_{\rm taper} \rightarrow \infty$, and can be forced using the tapered_powerlaw=False argument.

We additionally allow for an inner cavity, $r_{\rm cavity}$, inside which all emission heights are set to zero, and the radial range is shifted such that $r' = r - r_{\rm cavity}$. This can be toggled with the include_cavity argument.

The fitting (or more acurately the estimation of the posterior distributions) is performed with emcee. If starting positions are not provided, will use fit_emission_surface to estimate starting positions.

The priors are provided by a dictionary where the keys are the relevant argument names. Each param is described by two values and the type of prior. For a flat prior, priors['name']=[min_val, max_val, 'flat'], while for a Gaussian prior, priors['name']=[mean_val, std_val, 'gaussian'].

Parameters

- **r0** (*Optional[float]*) – The reference radius for z_0. Defaults to 1 arcsec, unless dist is provided, then defaults to 100 au.
- **dist** (*Optional[float]*) – Convert all distances from [arcsec] to [au] for the fitting. If this is provided, r_ref will change to 100 au unless specified by the user.
- **tapered_powerlaw** (*optional[bool]*) – Whether to include a tapered component to the powerlaw.
- **include_cavity** (*optional[bool]*) – Where to include an inner cavity.
- **p0** (*optional[list]*) – Starting guesses for the fit. If nothing is provided, will try to guess from the results of fit_emission_surface.
- **nwalkers** (*optional[int]*) – Number of walkers for the MCMC.

- **nburnin** (*optional[int]*) – Number of steps to take to burn in.
- **nsteps** (*optional[int]*) – Number of steps used to sample the PDF.
- **scatter** (*optional[float]*) – Relative scatter used to randomize the starting positions of the walkers.
- **priors** (*optional[dict]*) – A dictionary of priors to use for the fitting.
- **returns** (*optional[list]*) – A list of properties to return. Can include: `'samples'`, for the array of PDF samples (default); `'percentiles'`, for the 16th, 50th and 84th percentiles of the PDF; `'lnprob'` for values of the log-probablity for each of the PDF samples; 'median' for the median value of the PDFs and `'walkers'` for the walkers.
- **plots** (*optional[list]*) – A list of plots to make, including `'corner'` for the standard corner plot, or `'walkers'` for the trace of the walkers.
- **curve_fit_kwargs** (*optional[dict]*) – Kwargs to pass to `scipy.optimize.curve_fit` if the `p0` values are estimated through `fit_emision_surface`.

Returns Dependent on the `returns` argument.

plot_surface(*ax=None*, *side='both'*, *reflect=False*, *masked=True*, *plot_fit=False*, *tapered_powerlaw=True*, *include_cavity=False*, *return_fig=False*)

Plot the emission surface.

Parameters

- **ax** (*Optional[Matplotlib axis]*) – Axes used for plotting.
- **masked** (*Optional[bool]*) – Whether to plot the maske data or not. Default is `True`.
- **side** (*Optional[str]*) – Which emission side to plot, must be `'front'`, `'back'` or `'both'`.
- **reflect** (*Optional[bool]*) – If plotting the `'back'` side of the disk, whether to reflect it about disk midplane.
- **tapered_powerlaw** (*Optional[bool]*) – TBD
- **include_cavity** (*Optional[bool]*) – TBD
- **return_fig** (*Optional[bool]*) – Whether to return the Matplotlib figure if `ax=None`.

Returns If `return_fig=True`, the Matplotlib figure used for plotting.

INDEX