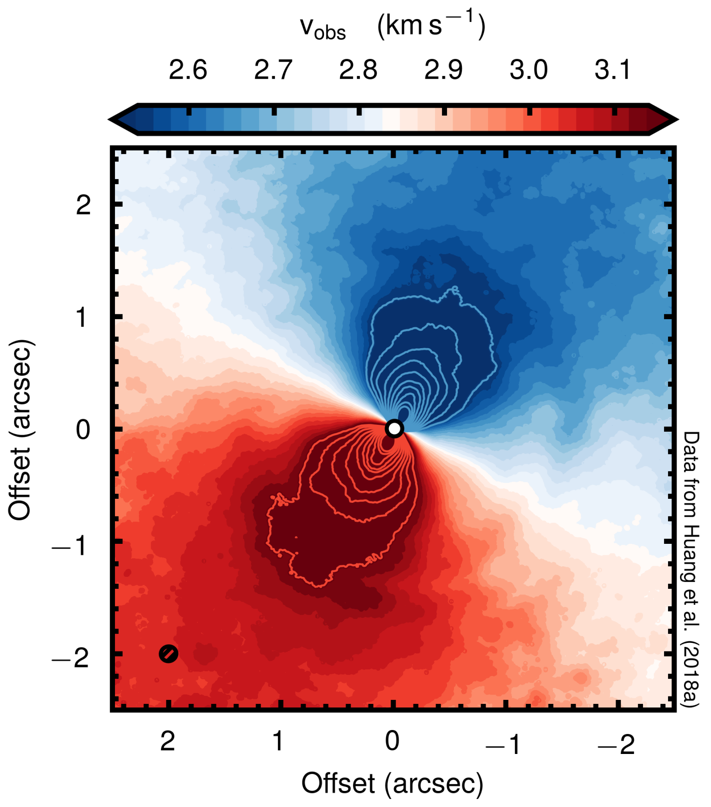
bettermoments

Aug 05, 2022

Contents



`bettermoments` creates moment maps of spectral line data and their associated uncertainties. The command-line interface makes it as seamless as possible to make all the traditional moment maps, in addition other, oftentimes more useful, maps. In addition to the many traditional statistical moments, `bettermoments` contains many alternative ways collapse the cube.

The `quadratic` method is described in detail in Teague & Foreman-Mackey (2018) while the uncertainties associated with more typical moment maps can be found in Teague (2019).

CHAPTER 1

Installation

Installation is as easy as cloning the git repository then move to the home directory,

```
pip install bettermoments
```

and you're done!

1.1 API

Here we describe all the functions used to collapse the spectral cube which are typically called by the command line interface. However, importing these into your workflow may be useful.

In general, for the generated moment maps, `MX`, where `X` is an integer denotes a statistical moment. For the non-traditional methods, `v0`, `dV` and `Fnu` represent the line center, width and peak, respectively.

Note: The convolution for `smooththreshold` is currently experimental and is work in progress. If things look suspicious, please raise an issue.

1.1.1 Moment Maps

Implementation of traditional moment-map methods. See the CASA documentation for more information.

`bettermoments.methods.`**`collapse_zeroth`**(*velax*, *data*, *rms*)

Collapses the cube by integrating along the spectral axis. It will return the integrated intensity along the spectral axis, `M0`, and the associated uncertainty, `dM0`. Following Teague (2019) these are calculated by,

$$M_0 = \sum_i^N I_i \, \Delta v_{\mathrm{chan},\, i}$$

and

$$M_0 = \sqrt{\sum_{i\,(I_i > 0)}^{N} \sigma_i^2 \cdot \Delta v_{\mathrm{chan},\,i}^2}$$

where Δv_i and I_i are the chanenl width and flux density at the i^{th} channel, respectively and the sum goes over the whole `axis`.

Parameters

- **`velax`** (*`ndarray`*) – Velocity axis of the cube.
- **`data`** (*`ndarray`*) – Flux densities or brightness temperature array. Assumes that the first axis is the velocity axis.
- **`rms`** (*`float`*) – Noise per pixel in same units as `data`.

Returns `M0`, the integrated intensity along provided axis and `dM0`, the uncertainty on `M0` in the same units as `M0`.

Return type `M0` (*ndarray*), `dM0` (*ndarray*)

`bettermoments.methods.`**`collapse_first`**(*velax*, *data*, *rms*)

Collapses the cube using the intensity weighted average velocity (or first moment map). For a symmetric line profile this will be the line center, however for highly non-symmetric line profiles, this will not give a meaningful result. Following Teague (2019), the line center is given by,

$$M_1 = \frac{\sum_i^N I_i v_i}{\sum_i^N I_i}$$

where v_i and I_i are the velocity and flux density at the i^{th} channel, respectively and the sum goes over the whole `axis`. In addition, the uncertainty is given by,

$$\delta M_1 = \sqrt{\sum_{i\,(I_i > 0)}^{N} \sigma_i^2 \cdot (v_i - M_1)^2}$$

where σ_i is the rms noise.

Parameters

- **`velax`** (*`ndarray`*) – Velocity axis of the cube.
- **`data`** (*`ndarray`*) – Flux densities or brightness temperature array. Assumes that the zeroth axis is the velocity axis.
- **`rms`** (*`float`*) – Noise per pixel in same units as `data`.

Returns `M1`, the intensity weighted average velocity in units of `velax` and `dM1`, the uncertainty in the intensity weighted average velocity with same units as `v0`.

Return type `M1` (*ndarray*), `dM1` (*ndarray*)

`bettermoments.methods.`**`collapse_second`**(*velax*, *data*, *rms*)

Collapses the cube using the intensity-weighted average velocity dispersion (or second moment). For a symmetric line profile this will be a measure of the line width. Following Teague (2019) this is calculated by,

$$M_2 = \sqrt{\frac{\sum_i^N I_i (v_i - M_1)^2}{\sum_i^N I_i}}$$

where M_1 is the first moment and v_i and I_i are the velocity and flux density at the $i^{\rm th}$ channel, respectively. The uncertainty is given by,

$$\delta M_2 = \frac{1}{2M_2} \cdot \sqrt{\sum_{i\,(I_i>0)}^{N} \sigma_i^2 \cdot \big[(v_i - M_1)^2 - M_2^2\big]^2}$$

where σ_i is the rms noise in the $i^{\rm th}$ channel.

Parameters

- **`velax`** (*`ndarray`*) – Velocity axis of the cube.
- **`data`** (*`ndarray`*) – Flux densities or brightness temperature array. Assumes that the first axis is the velocity axis.
- **`rms`** (*`float`*) – Noise per pixel in same units as `data`.

Returns `M2` is the intensity weighted velocity dispersion with units of `velax`. `dM2` is the uncerat-inty of `M2` in the same units.

Return type `M2` (ndarray), `dM2` (ndarray)

`bettermoments.methods.`**`collapse_eighth`**(*velax*, *data*, *rms*)

Take the peak value along the provided axis. The uncertainty is the RMS noise of the image.

Parameters

- **`velax`** (*`ndarray`*) – Velocity axis of the cube. Not needed.
- **`data`** (*`ndarray`*) – Flux densities or brightness temperature array. Assumes that the first axis is the velocity axis.
- **`rms`** (*`float`*) – Noise per pixel in same units as `data`.

Returns The peak value, `M8`, and the associated uncertainty, `dM8`.

Return type `M8` (*ndarray*), `dM8` (*ndarray*)

`bettermoments.methods.`**`collapse_ninth`**(*velax*, *data*, *rms*)

Take the velocity of the peak intensity along the provided axis. The uncertainty is half the channel width.

Parameters

- **`velax`** (*`ndarray`*) – Velocity axis of the cube.
- **`data`** (*`ndarray`*) – Flux densities or brightness temperature array. Assumes that the first axis is the velocity axis.
- **`rms`** (*`float`*) – Noise per pixel in same units as `data`.

Returns The velocity value of the peak value, `M9`, and the associated uncertainty, `dM9`.

Return type `M9` (*ndarray*), `dM9` (*ndarray*)

`bettermoments.methods.`**`collapse_maximum`**(*velax*, *data*, *rms*)

A wrapper returning the result of both `bettermoments.collapse_cube.collapse_eighth()` and `bettermoments.collapse_cube.collapse_ninth()`.

Parameters

- **`velax`** (*`ndarray`*) – Velocity axis of the cube.
- **`data`** (*`ndarray`*) – Flux densities or brightness temperature array. Assumes that the first axis is the velocity axis.
- **`rms`** (*`float`*) – Noise per pixel in same units as `data`.

Returns The peak value, `M8`, and the associated uncertainty, `dM8`. The velocity value of the peak value, `M9`, and the associated uncertainty, `dM9`.

Return type `M8` (*ndarray*), `dM8` (*ndarray*), `M9` (*ndarray*), `dM9` (*ndarray*)

1.1.2 Non-Traditional Methods

`bettermoments.methods.`**`collapse_quadratic`**`(`*velax*, *data*, *rms*`)`

Collapse the cube using the quadratic method presented in Teague & Foreman-Mackey (2018). Will return the line center, `v0`, and the uncertainty on this, `dv0`, as well as the line peak, `Fnu`, and the uncertainty on that, `dFnu`. This provides the sub-channel precision of `bettermoments.collapse_cube.collapse_first()` with the robustness to noise from `bettermoments.collapse_cube.collapse_ninth()`.

Parameters

- **`velax`** (*`ndarray`*) – Velocity axis of the cube.
- **`data`** (*`ndarray`*) – Flux density or brightness temperature array. Assumes that the zeroth axis is the velocity axis.
- **`rms`** (*`float`*) – Noise per pixel in same units as `data`.

Returns `v0`, the line center in the same units as `velax` with `dv0` as the uncertainty on `v0` in the same units as `velax`. `Fnu` is the line peak in the same units as the `data` with associated uncertainties, `dFnu`.

Return type `v0` (*ndarray*), `dv0` (*ndarray*), `Fnu` (*ndarray*), `dFnu` (*ndarray*)

`bettermoments.methods.`**`collapse_width`**`(`*velax*, *data*, *rms*`)`

Returns an effective width, a rescaled ratio of the integrated intensity and the line peak. For a Gaussian line profile this would be the Doppler width as the total intensity is given by,

$$M_0 = \sum_i^N I_i \, \Delta v_{\mathrm{chan},\, i}$$

where Δv_i and I_i are the chanenl width and flux density at the i^{th} channel. If the line profile is Gaussian, then equally

$$M_0 = \sqrt{\pi} \times F_{\nu} \times \Delta V$$

where F_{ν} is the peak value of the line and ΔV is the Doppler width of the line. As M_0 and F_{ν} are readily calculated using `bettermoments.collapse_cube.collapse_zeroth()` and `bettermoments.collapse_cube.collapse_quadratic()`, respectively, ΔV can calculated through $\Delta V = M_0 \, / \, (\sqrt{\pi} \, F_{\nu})$. This should be more robust against noise than second moment maps.

Parameters

- **`velax`** (*`ndarray`*) – Velocity axis of the cube.
- **`data`** (*`ndarray`*) – Flux densities or brightness temperature array. Assumes that the first axis is the velocity axis.
- **`rms`** (*`float`*) – Noise per pixel in same units as `data`.

Returns The effective velocity dispersion, `dV` and `ddV`, the associated uncertainty.

Return type `dV` (*ndarray*), `ddV` (*ndarray*)

1.1.3 (Higher Order) Gaussian Fits

`bettermoments.methods.`**`collapse_gaussian`**(*velax*, *data*, *rms*, *indices=None*, *chunks=1*, ***kwargs*)

Collapse the cube by fitting a Gaussian line profile to each pixel. This function is a wrapper of *collapse_analytical* which provides more details about the arguments.

Parameters

- **`velax`** (*`ndarray`*) – Velocity axis of the cube.
- **`data`** (*`ndarray`*) – Maksed intensity or brightness temperature array. The first axis must be the velocity axis.
- **`rms`** (*`float`*) – Noise per pixel in same units as `data`.
- **`indices`** (*`Optional[list]`*) – A list of pixels described by `(y_idx, x_idx)` tuples to fit. If none are provided, will fit all pixels.
- **`chunks`** (*`Optional[int]`*) – Split the cube into `chunks` sections and run the fits with separate processes through `multiprocessing.pool`.

Returns

`gv0` (*ndarray*), `dgv0` (*ndarray*), `gFnu` (*ndarray*), `dgFnu` (*ndarray*), `gdV` (*ndarray*), `dgdV` (*ndarray*):

The Gaussian center, `gv0`, the line peak, `gFnu` and the Doppler width, `gdV`, all with associated uncertainties, `dg*`.

`bettermoments.methods.`**`collapse_gaussthick`**(*velax*, *data*, *rms*, *indices=None*, *chunks=1*, ***kwargs*)

Collapse the cube by fitting a Gaussian line profile with an optically thick core to each pixel. This function is a wrapper of *collapse_analytical* which provides more details about the arguments.

Parameters

- **`velax`** (*`ndarray`*) – Velocity axis of the cube.
- **`data`** (*`ndarray`*) – Maksed intensity or brightness temperature array. The first axis must be the velocity axis.
- **`rms`** (*`float`*) – Noise per pixel in same units as `data`.
- **`indices`** (*`Optional[list]`*) – A list of pixels described by `(y_idx, x_idx)` tuples to fit. If none are provided, will fit all pixels.
- **`chunks`** (*`Optional[int]`*) – Split the cube into `chunks` sections and run the fits with separate processes through `multiprocessing.pool`.

Returns

`gtv0` (*ndarray*), `dgtv0` (*ndarray*), `gtFnu` (*ndarray*), `dgtFnu` (*ndarray*), `gtdV` (*ndarray*), `dgtdV` (*ndarray*), `gttau` (*ndarray*), `dgttau` (*ndarray*):

The Gaussian center, `gtv0`, the line peak, `gtFnu`, the Dopler width, `gtdV`, and the effective optical depth, `gttau`, all with associated uncertainties, `dgt*`.

`bettermoments.methods.`**`collapse_gausshermite`**(*velax*, *data*, *rms*, *indices=None*, *chunks=1*, ***kwargs*)

Collapse the cube by fitting a Gaussian line profile with an optically thick core to each pixel. This function is a wrapper of *collapse_analytical* which provides more details about the arguments.

Parameters

- **velax** (*ndarray*) – Velocity axis of the cube.
- **data** (*ndarray*) – Maksed intensity or brightness temperature array. The first axis must be the velocity axis.
- **rms** (*float*) – Noise per pixel in same units as `data`.
- **indices** (*Optional[list]*) – A list of pixels described by `(y_idx, x_idx)` tuples to fit. If none are provided, will fit all pixels.
- **chunks** (*Optional[int]*) – Split the cube into `chunks` sections and run the fits with separate processes through `multiprocessing.pool`.

Returns

`ghv0` (*ndarray*), `dghv0` (*ndarray*), `ghFnu` (*ndarray*), `dghFnu` (*ndarray*), `ghdV` (*ndarray*), `dghdV` (*ndarray*), `ghh3` (*ndarray*), `dghh3` (*ndarray*), `ghh4` (*ndarray*), `dghh4` (*ndarray*):

The Gaussian center, `ghv0`, the line peak, `ghFnu`, the Dopler width, `ghdV`, with additional expansion terms `ghh3`, the assymetry of the line and ``ghh4`, the saturation of the line core., All values come with their associated uncertainties, `dgt*`.

1.2 Command Line Interface

This is the preferred way to interact with `bettermoments`. If the install has gone successfully, you should be able to run `bettermoments` from any directory using,

```
bettermoments path/to/cube.fits
```

Which, by default, will use the `collapse_quadratic()` function to calculate line center and line peak maps. This will automatically extract the data array and spectral axis from the cube and provide them to the appropriate functions.

Warning: The command line interface will automatically overwrite any files with the same name. Make sure that you move or rename old files which you want to keep or use the `--nooverwrite` flag.

1.2.1 Different Methods

To change the method applied to collapse the cube, use the `-method [name]` flag, where the names of the functions are found in the *API*. For example, to calculate the zeroth moment map,

```
bettermoments path/to/cube.fits -method zeroth
```

which will produce a `*M0.fits` file with the uncertainties in `*dM0.fits`.

1.2.2 Smoothing

It is sometimes useful to smooth along the spectral axis of the data prior to calculating the requested moment map. While this can remove high frequency noise and usually lead to a better determination of the desired statistic, any level of smoothing will reduce peak values of a spectrum, so any statistic based on the absolute value of the data will be under-estimated.

Smoothing is achieved with the `-smooth [window]` flag, where the window size as the number of channels is given. By default this is a top hat function which is applied along the spectral axis prior to any other calculations (including the estimation of the RMS).

It is possible to request a Savitzky-Golay filter using the additional `-polyorder [order]` flag which denotes the order of the polynomal used in the filter. Note that this needs to be greater than 1, but also two less than the window size.

```
bettermoments path/to/cube/fits -smooth 5
```

will smooth the data with a top-hat kernel with a width of 5 channels while

```
bettermoments path/to/cube/fits -smooth 5 -polyorder 2
```

will smooth the data with Savitzky-Golay filter with a window size of 5 channels and use a polynomial of order 2.

1.2.3 Masking

When making a moment map it is often useful, sometimes necessary, to mask the data to reduce the noise in the resulting image. There are a couple of different options in `bettermoments` to do this.

Channel Selection

The most straight forward is a simple channel selection using the `-firstchannel [chan]` and `-lastchannel [chan]` arguments. By default these span the entire cube range. For example,

```
bettermoments path/to/cube.fits -method zeroth -firstchannel 5 -lastchannel 10
```

would create a zeroth moment map using the channels 5 to 10 inclusively.

Threshold Clipping

One of the most common approaches is to apply a ‘sigma clip’, essentially masking any pixels below some user-specified threshold, usual in untis of the background RMS. In `bettermoments` this is applied with the `-clip [value]` argument. For example,

```
bettermoments path/to/cube.fits -method zeroth -clip 2
```

would calculate a zeroth moment map out of all the pixels which have an absolute value of greater than or equal to `2 * RMS`. The background RMS is automatically calculated using the central 50% of the pixels in the first and last 5 channels. The number of channels used for this estimation can be changed with the `-noisechannels` argument. Rather than calculating the RMS automatically, you can specify their own value with the `-rms` argument. Note that internally the RMS is assumed to be homogeneous, both spatially and spectrally.

If you want include asymmetric bounds you can include two `-clip` values. For example,

```
bettermoments path/to/cube.fits -method zeroth -clip -3 2
```

would mask out all pixel values between `-3 * RMS` and `2 * RMS`.

A threshold mask like the above can sometimes leave sharp boundaries if you have large spatial gradients in the intensity. To counter this it is possible to convolve the threshold mask with a 2D Gaussian kernel to smooth these edges with the `-smooththreshold [width]` argument where the width is given in units of the beam FWHM (or pixel scale if a beam isn’t provided). Internally this will make a copy of the data, convolve with the appropriate kernel, then generate a boolean mask where the convolved map meets the specified `-clip` criteria.

Warning: If you choose to smooth the threshold map, remember that the RMS in this image will be reduced due to the smoothing. The automatic calculation of the RMS is done before the smoothing of the map so it will be appropriate to provide a user-specified one with `-rms [value]`.

User-Defined Masks

Sometimes you may want to include a user-defined mask, such at the CLEAN mask used when imaging interferometric data. As long as the mask has the same shape as the data in the image cube you can include this with,

```
bettermoments path/to/cube.fits -mask path/to/mask.fits
```

Combing Masks

If you've specified both a user-defined mask and provided a `clip` value then `bettermoments` will combine the two masks by default using `AND`. If you would rather choose a less conservative `OR` combination then you can include the `-combine or` argument.

Returning Masks

It is often useful to have a copy of the mask used to generate the moment map such that you can overplot it in channel maps to help make sense of what you're seeing. To do this, use the `--returnmask` flag.

1.2.4 Help

For help with the exact command line options, use

```
bettermoments --help
```

1.3 Scripting `bettermoments`

In this Notebook, we will step through how to integrate the moment map making process (in this case, a zeroth moment map, or integrated intensity map), into your workflow. This should elucidate the steps that are taken automatically when using the command line interface.

1.3.1 Standard Imports

```
[1]: import bettermoments as bm
```

1.3.2 Load Up the Data

Here the `load_cube` function will return a 3D array for the data and a 1D array of the velocity axis (this should automatically convert any frequency axis to a velocity axis). Note that as we are collapsing along the velocity axis, we have no need for spatial axes, so we do not bother creating them.

```
[2]: path = '../../../gofish/docs/user/TWHya_CS_32.fits'
     data, velax = bm.load_cube(path)
```

1.3.3 Spectrally Smooth the Data

If you have relatively noisy data, a low level of smoothing along the spectral axis can be useful. `bettermoments` allows for two different methods: a convolution with a simple top-hat function, or the use of a Savitzky-Golay filter. For a top-hat convolution, you need only specify `smooth`, which describes the kernel size in the number of channels. For a Savitzky-Golay filter, you must also provide `polyorder` which describes the polynomial order which is used for the fitting. Note that `smooth` must always be larger than `polyorder`.

It is important to remember that while a small level of smoothing can help with certain aspects of moment map creation, it also distorts the line profile (for example broadening the line in the case of a simple top-hat convolution). Such systematic effects must be considered when analysing the resulting moment maps.

Here we just consider a smoothing with a top-hat kernel that is 3 channels wide.

```
[3]: smoothed_data = bm.smooth_data(data=data, smooth=3, polyorder=0)
```

1.3.4 Estimate the Noise of the Data

We require an estimate of the noise of the data for two reasons:

1. For the estimation of the uncertainties of the moment maps.
2. For applying anything threshold clipping.

To make this estimate, we assume that the noise in the image is constant both spatially (such that the primary beam correction is minimal) and spectrally. To avoid line emission, we consider the RMS of the line-free channels, defined as the first `N` and last `N` channels in the data cube.

```
[4]: rms = bm.estimate_RMS(data=data, N=5)
```

Note that the noise estimated this way will differ whether you use the `smoothed_data` or the original `data` array. When using the command line interface for `bettermoments`, the RMS will be estimated on the *smoothed* data.

```
[5]: rms_smoothed = bm.estimate_RMS(data=smoothed_data, N=5)

     print('RMS = {:.1f} mJy/beam (original)'.format(rms * 1e3))
     print('RMS = {:.1f} mJy/beam (smoothed)'.format(rms_smoothed * 1e3))
```

```
RMS = 2.8 mJy/beam (original)
RMS = 2.2 mJy/beam (smoothed)
```

1.3.5 User-Defined Mask

Sometimes you will want to mask particular regions within your PPV cube in order to disentangle various components, for example if you have multiple hyperfine components that you want to distinguish. Often the easiest way to do this is to define a mask elsewhere and apply it to the data you are collapsing (see for example the keplerian_mask.py routine to generate a Keplerian mask).

Through the `get_user_mask` function, you can load a mask (a 3D array of 1s and 0s) saved as a FITS file, and apply that to the data. If no `user_mask_path` is provided, then this simply returns an array with the same shape as `data` filled with 1s.

Note that the user-defined mask *must* share the same pixel and channel resolution, and be the same shape as the data. No aligning or reshaping is done internally with `bettermoments`.

```
[6]: user_mask = bm.get_user_mask(data=data, user_mask_path=None)
```

1.3.6 Threshold Mask

A threshold mask, or a 'sigma-clip', is one of the most common approaches to masking used in moment map creation. The `get_threshold_mask` provides several features which will help you optimize your threshold masking.

The `clip` argument takes a tuple of values, `clip=(-3.0, 3.0)` describing the minimum and maximum SNR of the pixels that will be removed (this is very similar to the `excludepix` argument in CASA's immoments task, but with values given in units of sigma, the noise, rather than flux units). `clip` also accepts just a single value, and will convert that to a symmetric clip as above, for example `clip=(-2.0, 2.0)` and `clip=2.0` are equivalent. The option to provide a tuple allows the options to have asymmetric clip ranges, for example, `clip=(-np.inf, 3.0)`, to remove all pixels below 3 sigma, including high significance but negative pixels.

It has been found that threshold masks can lead to large artifacts in the resulting moment map if there are large intensity gradients in low SNR regions of the PPV cube. To combate this, users have the option to first smooth the data (only temporarily to generate the threshold mask) which will allow for more conservative contours in the threshold mask. This can be achived by providing the FWHM of the Gaussian kernel used for this spatial smooth as `smooth_threshold_mask` in number of pixels. Note that because the data is smoothed, the effective RMS will drop and so the RMS is re-estimated interally on the smoothed image.

Here we mask all pixels with a SNR less than 2 sigma, i.e., $|I\,/\,\sigma| < 2$.

```
[7]: threshold_mask = bm.get_threshold_mask(data=data,
                                            clip=2.0,
                                            smooth_threshold_mask=0.0)
```

1.3.7 Channel Mask

For many PPV cubes, the line emission of interest only spans a small range of velocity axis. This region can be easily selected using the `firstchannel` and `lastchannel` arguments in `get_channel_mask`. Note that the `lastchannel` argument also accepts negative values, following the standard Python indexing convention, i.e., `lastchannel=-1` results in the final channel being the last.

`get_channel_mask` also accepts a `user_mask` argument, which is an array the same size as the velocity axis of the data, specifying which channels to include. This may be useful if you want to integrate over several hyperfine components while excluding the line-free regions between them.

```
[8]: channel_mask = bm.get_channel_mask(data=data,
                                        firstchannel=0,
                                        lastchannel=-1)
```

1.3.8 Mask Combination

All the masks can be easily combined, either with `AND` or `OR`, with the `get_combined_mask` function. This can then be applied to the data used for the moment map creation through a simple multiplication. Note for all collapse functions, pixels with a 0 value will be ignored.

[9]:
```
mask = bm.get_combined_mask(user_mask=user_mask,
                            threshold_mask=threshold_mask,
                            channel_mask=channel_mask,
                            combine='and')
masked_data = smoothed_data * mask
```

1.3.9 Collapse the Data

Now that we have a smoothed and masked dataset, we can collapse it along the velocity axis through several different methods. (https://bettermoments.readthedocs.io/en/latest/user/collapse_cube.html#). In general, most functions require the velocity axis, `velax`, the masked data data, `data`, and the RMS of the data, `rms`. The available functions can be checked through the `available_collapse_methods` function such that the desired function is `collapse_{methodname}`.

[10]:
```
bm.available_collapse_methods()
```

```
Available methods are:

         zeroth       (integrated intensity)
         first        (intensity weighted average velocity)
         second       (intensity weighted velocity dispersion)
         eighth       (peak intensity)
         ninth        (velocity channel of peak intensity)
         maximum      (both collapse_eighth and collapse_ninth)
         quadratic    (quadratic fit to peak intensity)
         width        (effective width for a Gaussian profile)
         gaussian     (gaussian fit)
         gaussthick   (gaussian with optically thick core fit)
         gausshermite (gaussian-hermite expansion fit)

Call the function with `collapse_{method_name}`.
```

Each function will return `moments`, an `(N, Y, X)` shaped array, where `(Y, X)` is the shape of a single channel of the data and `N` is twice the number of statistics (with the uncertainty of each value interleaved). To see which parameters are returned for each `collapse_method`, we can use the `collapse_method_products` function. For the `'zeroth'` method:

[11]:
```
bm.collapse_method_products('zeroth')
```

[11]:
```
'M0, dM0'
```

So we have the zeroth moment, `M0`, and it's associated uncertainty `dM0`.

Here we will collapse the cube to a zeroth (integrated intensity) map.

[12]:
```
moments = bm.collapse_zeroth(velax=velax, data=masked_data, rms=rms)
```

1.3.10 Save the Data to FITS

It's possible to work with the data directly, however it's often useful to save these for later. The `save_to_FITS` function will split up the `moments` array and save each one as a new FITS file, replacing the `.fits` exention with `_{moment_name}.fits` for easy identification. The header will be copied from the original file.

[13]:
```
bm.save_to_FITS(moments=moments, method='zeroth', path=path)
```

1.4 Citations

If you use `bettermoments` in your research, please make sure to cite the RNASS article Teague & Foreman-Mackey as,

```
@ARTICLE{2018RNAAS...2c.173T,
       author = {{Teague}, Richard and {Foreman-Mackey}, Daniel},
        title = "{A Robust Method to Measure Centroids of Spectral Lines}",
      journal = {Research Notes of the American Astronomical Society},
         year = 2018,
        month = Sep,
       volume = {2},
          eid = {173},
        pages = {173},
          doi = {10.3847/2515-5172/aae265},
       adsurl = {https://ui.adsabs.harvard.edu/abs/2018RNAAS...2c.173T},
      adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

Additionally, if you make use of the uncertainties for the traditional moment maps, then citing Teague (2019) would be appreciated,

```
@ARTICLE{2019RNAAS...3e..74T,
       author = {{Teague}, Richard},
        title = "{Statistical Uncertainties in Moment Maps of Line Emission}",
      journal = {Research Notes of the American Astronomical Society},
         year = "2019",
        month = "May",
       volume = {3},
       number = {5},
          eid = {74},
        pages = {74},
          doi = {10.3847/2515-5172/ab2125},
       adsurl = {https://ui.adsabs.harvard.edu/abs/2019RNAAS...3e..74T},
      adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

Python Module Index

Index