Manual of plotastrodata

Yusuke Aso

1 Read Data

The data class AstroData can take a fits file.

```
from plotastrodata.analysis_utils import AstroData

d = AstroData(fitsimage='file_name.fits')
```

AstroData can take more arguments, such as `Tb` and `sigma`. `Tb=True` means the data values will be converted from flux densities in the unit of Jy beam^{-1} to brightness temperatures in the unit of K. `sigma` specifies how to measure the noise level of the data values: for example, the default option of 'hist' means to use the histgram of the data values. The argument of `pvpa` is the position angle of the PV cut direction in the unit of degree, which will be used to calculate the spatial resolution of the PV diagram.

The data class AstroFrame is necessary to form the AstroData instance in a useful format. AstroFrame can take quantities related to coordinate ranges.

```
from plotastrodata.analysis_utils import AstroFrame

f = AstroFrame(vmin=-5.0, vmax=5.0, vsys=2.0,
               center='00h00m00s 00d00m00s', rmax=3.0)
```

`vmin`, `vmax`, and `vsys` are in the unit of km s^{-1}. `rmax` is in the unit of arcsec. Instead of `rmax`, other arguments (`xmin`, `xmax`, `ymin`, `ymax`, `xoff`, `yoff`) can be used to adjust the x and y ranges in more detail. When an argument of `fitsimage` is given, the central coordinates `center` is read from the given fits file; the rest frequency is also read from the fits file, which is used for the frequency-velocity conversion. Moreover, an argument of `dist` can specifies the distance used to change the unit of spatial coordinates from arcsec to au. An argument of `swapxy` is used to swap the x and y coordinates. An argument of `pv` must be `True` when the AstroData instance to be read is a position-velocity (PV) diagram, i.e., the first and second axes are the spatial and velocity coordinates. When `quadrants=True`, the first and third (or second and fourth) quadrants of the PV diagram will be averaged. `xflip` and `yflip` will be used to determine the plotting direction of the x and y axes, respectively; these have a meaning only when a figure is made from the data set.

Forming the AstroData instance needs the following command.

```
f.read(d)
```

After this command, the AstroData instance has useful attributes in the format of numpy array: a 1D array of `d.x` (as well as `d.y` and `d.v`), a 2D or 3D array of `d.data`, a 1D array of `d.beam`, and a float of `d.sigma`. `d.x` and `d.y` are the relative coordinates in the unit of arcsec from the given `center`. Similarly, `d.v` is the relative coordinate in the unit of km s^{-1} from the given `vsys`. `d.beam` is the beam components `array([bmaj, bmin, bpa])`, where `bmaj` and `bmin` are in the unit of arcsec, while `bpa` is in the unit of degree. When `Tb=True` above, the data values are converted to the brightness temperature and stored as `d.data`. `d.sigma` is the noise level measured in the way specified above in the same unit as `d.data`. AstroData can take two more arguments, `restfreq` and `cfactor`. `restfreq` is used to explicitly set the rest frequency in the unit of Hz, which is used to convert frequency to velocity and Jy beam^{-1} to K. When this argument is zero (`restfreq=0`), the frequency axis is not converted to velocity. When the unit of the frequency/velocity axis in the input fits header is m/s, `f.read(d)` divides the coordinate by 1000. The argument `cfactor` specifies a constant factor that `d.data` is multiplied by. This will be useful when one wants to change the intensity unit from Jy beam^{-1} to mJy beam^{-1}.

In addition to the input attributes, `f.read(d)` adds three attributes to the AstroData instance `d`: `fitsimage_org`, `sigma_org`, and `fitsheader`. `fitsimage_org` saves the input `fitsimage`, while `fitsimage` is updated to None after `f.read(d)`. Similarly, `sigma_org` saves the input `sigma`, which may be a string, while `sigma` is updated to the calculated value. `fitsheader` is the fits header in the format of astropy.io.fits.open(fitsimage)[0].

When `pv=True` in `AstroFrame`, the `read` method changes the `beam` attribute of the AstroData instance. The first element will be the velocity resolution. The second element of `beam` will be $1/\sqrt{\cos^2(\mathrm{bpa}-\mathrm{pvpa})/\mathrm{bmaj}^2+\sin^2(\mathrm{bpa}-\mathrm{pvpa})/\mathrm{bmin}^2}$. This is calculated from the intersection of the beam ellipse and the PV cut line. The third element will be 0; thus, the velocity resolution (first element) is regarded as the vertical length by default when this "beam" is plotted in a PV diagram. The original beam will be stored in `beam_org`.

2 Analyze Data

AstroData also has handy methods to analyze the 2D/3D data.

The `binning` method rebins the 2D/3D data with a given width for each coordinate.

```
d.binning(width=[5, 4, 2])
```

This command takes an average over 5 channels in the velocity direction, 4 pixels in the y direction, and 2 pixels in the x direction. The number of channels and pixels are decreased accordingly after this method. If the length of `width` is 2, the two values are regarded as the widths for the y and x directions.

The `centering` method sets the coordinates so that the spatial center and the systemic velocity have the exact zero coordinates by interpolation.

```
d.centering(includexy=True, includev=False)
```

This command adjusts the x and y coordinates but does not adjust the velocity coordinate. This method will be useful when one wants to quickly get a radial profile along a line passing the center or a PV diagram (using the `rotate` method together).

The `circularbeam` method makes the beam shape circular by additional 2D Gaussian convolution. This method takes no argument.

```
d.circularbeam()
```

The new beam has the major and minor axes same as the old major axis. This method also updates the attribute of `d.beam` accordingly.

The `deproject` method deprojects the 2D/3D data with a given position angle (P.A.) and inclination angle.

```
d.deproject(pa=45, incl=45)
```

`pa` is the position angle from the north to the east in the unit of degree. `incl` is the inclination angle; `incl=0` means the face-on configuration and thus no deprojection. This command replaces `d.data` and `d.beam` with the deprojected data and the deprojected beam, respectively.

The `fit2d` method performs MCMC fitting to the 2D image of `d.data` or `d.data[chan]`, where `chan` is the channel number, by using emcee or ptemcee through the class of `plotastrodata.fitting_utils.EmceeCorner`.

```
res = d.fit2d(chan=12, model=model, bounds=bounds,
              kwargs_fit={'nwalkersperdim': 2, 'nsteps': 1000, 'nburnin': 500},
              kwargs_plotcorner={'savefig': 'corner.png'})
```

`model` is a function with the shape of `model(par, x, y)`, where `par` is the list of parameters. `bounds` is the 2D list of the parameter boundary in the shape of $[[p_{0,\mathrm{min}}, p_{1,\mathrm{min}}, ...], [p_{0,\mathrm{max}}, p_{1,\mathrm{max}}, ...]]$. `kwargs_fit` and `kwargs_plotcorner` are the arguments for the methods of `EmceeCorner.fit` and `EmceeCorner.plotcorner`, respectively. `nsteps` includes the number the burn-in steps. The output `res` is a dictionary consisting of `popt`, `plow`, `pmid`, `phigh`, `model`, and `residual`. The first four keys provide the best, lower percentile, 50 percentile, and higher percentile parameters. The last two keys provide the model and residual 2D images.

The `histogram` method returns the bins and the histogram in the bins of the attribute of `d.data` by using numpy.histogram().

```
hbin, hist = d.histogram(bins=10)
```

The arguments are the same as those for numpy.histogram(). The bins (`hbin`) have the same length as the histogram (`hist`), which is different from the original numpy.histogram.

The `gaussfit2d` method performs the 2D Gaussian fitting to the 2D image of `d.data` or `d.data[chan]`, where `chan` is the channel number, by using `scipy.optimize.curve_fit`.

```
res = d.gaussfit2d(chan=12)
```

This command performs the fitting to the channel of 12 in `d.data`. For 2D data, the `chan` argument can be omitted. The output (`res` here) is a dictionary having keys of `popt`, `pcov`, `model`, and `residual`. `popt` and `pcov` are the optimized parameters (peak intensity, central x, central y, major FWHM, minor FWHM, and P.A.) and their covariance. The `model` and `residual` are 2D arrays with the same shape of the fitted 2D array.

The `mask` method puts numpy.nan on pixels that satisfies a given condition.

```
d.mask(dataformask=d2.data, includepix=[1e-3, 100], excludepix=[50, 200])
```

This command puts numpy.nan on pixels of `d.data` where `d2.data` is outside [1e-3, 100] or inside [50, 200], where `d2` is an AstroData instance different from `d`. This method will be useful when one wants to put a mask on a moment 1 map using the values of a moment 0 map.

The `profile` method makes line profiles at given spatial positions.

```
v, f, g = d.profile(xlist=[-0.5, 0.2], ylist=[1, 0.8], ellipse=[0.3, 0.2, 45])
```

This command makes line profiles at (x,y)=(-0.5, 1) and (0.2, 0.8) in the unit of arcsec. The intensity of each profile is an average over a boxcar ellipse with the major axis of 0.3 arcsec, the minor axis of 0.2 arcsec, and the P.A. of 45 degrees. Instead of `xlist` and `ylist`, coordinate strings can be input using an argument of `coords`: `coords=['00h00m00.0s 00d00m00.0s', '11h11m11.1s 11d11m11.1s']`. When `ellipse` is omitted, the profile is made by picking up the values at the closest pixel to the given position. When the ellipse size is not so large compared to the pixel size, the integer argument `ninterp` may be useful; this makes the pixel size `ninterp` times finer by interpolartion. When the boolian argument `flux` is True, the output profile has the unit of Jy. Additionally, when the boolian argument `gaussfit` is True, the profiles will be fitted with a 1D Gaussian function. The output `v` above is `d.v`. The output `f` above is a list of 1D-array profiles, i.e., 2D array. The output `g` is a list of dictionaries; each dictionary has keys of `best` and `error` (square root of the diagonal components of the covarience).

The `rotate` method rotates the attribute `d.data` by the given angle in the unit of degrees by interpolation.

```
d.rotate(pa=45)
```

When a pixel refers to a position outside the original image, this pixel has numpy.nan after this method.

The `slice` method makes a radial profile for a 2D image or a PV diagram for a 3D image along a given direction and length by interpolation.

```
r, f = d.slice(length=3, pa=45, dx=0.2)
```

This commands makes a radial profile (or PV diagram) along a cut with a length of 3 arcsec at P.A.=45 degrees with a separation of 0.2 arcsec. The output `r` and `f` are both 1D arrays of the positional offsets and the intensity at the positions. When the separation `dx` is omitted, the absolute value of the x pixel size is adopted.

The `todict` method returns the attributes as a dictionary.

```
a = d.todict()
```

This output includes keys of `data`, `x`, `y`, `v`, `fitsimage`, `beam`, `Tb`, `restfreq`, `cfactor`, `sigma`, and `center`. This dictionary can be input to methods of PlotAstroData as `**a`.

The `writetofits` method exports the instance as a fits file.

```
d.writetofits(fitsimage='new_file_name.fits')
```

The output fits file reuses the header components of the fits file used to make the AstroData instance (`'file_name.fits'` above); some header components are automatically updated after the above-mentioned methods for analysis, such as `CDELT1`.

3 Plot Data

The class PlotAstroData can take an AstroData instance through the method of `d.todict()`.

```
from plotastrodata.plot_utils import PlotAstroData

p = PlotAstroData(rmax=3.0)
p.add_color(**d.todict())
p.add_scalebar(length=50 / 140, label='50 au')
p.set_axis()
p.savefig('figure_name.png')
```

These commands make a color map using the AstroData instance `d`. PlotAstroData can take the same arguments as AstroFrame to define the plotting ranges; particularly `rmax` is necessary. The method `p.add_color()` can take a fits file directly instead of the AstroData instance, as `p.add_color(fitsimage='file_name.fits')`. This method can actually take the same arguments as AstroData to specify the data to be plotted as a color map; `**d.todict()` does this indirectly. The command `p.add_scalebar()` can be omitted if the map does not need to show a scale bar. The command `p.set_axis()` (or `p.set_axis_radec()`) is necessary even without any argument. More detailed usage can be found in the `example.py` file and `https://plotastrodata.readthedocs.io/en/latest/#`. The following is the explanation of each method (each type of maps).

The `add_color` method plots the given data in the format of a color map.

```
p.add_color(**d.todict(), stretch='log', show_cbar=True, cblabel=r'mJy beam$^{-1}$',
            cbticks=[0.01, 0.03, 0.1, 0.3], cbticklabels=['10', '30', '100', '300'],
            cblocation='right')
```

The `stretch` argument can be 'linear', 'log', 'asinh', or 'power'. The Arcsin hyperbolic stretch can be adjusted by another argument of `stretchscale` as asinh(data / stretchscale). The power-law stretch can be adjusted by another argument of `stretchpower` as ((data / vmin)^(1 - stretchpower) - 1) / (1 - stretchpower) / ln(10), where vmin is a given minimum value. The arguments starting with 'cb' adjust the colorbar. In addition, two arguments, `xskip` and `yskip`, can be used to skip spatial pixels in this method as well as `add_contour`, `add_segment`, and `add_rgb`. Similarly, `show_beam` and `beamcolor` can be used in these four methods.

The `add_contour` method plots the given data in the format of a contour map.

```
p.add_contour(**d.todict(), levels=[-3, 3, 6, 9])
```

The `levels` argument spedifies the contour levels in the unit of `d.sigma`.

The `add_rgb` method plots the given data in the format of three-color maps. The input three data sets are mixed as Red, Green, and Blue.

```
p.add_rgb(**d.todict(), stretch=['linear', 'log', 'linear'])
```

For this method, the input is a combination of three data sets, and thus `d.data` here must be a 1D list (not a numpy array) of three 2D/3D numpy arrays. In the same way as `add_color`, `stretchscale` and `stretchpower` are available in this method.

The `add_segment` method plots the given data in the format of a segment map, as is often used for polarization maps.

```
p.add_segment(Qfits='Qfile_name.fits', Ufits='Ufits_name.fits',
              ampfactor=3.8, angonly=False, rotation=90, cutoff=3.0)
```

The input format of the data is different from that for `add_color` and `add_contour`. One of the following pairs must be given: (`ampfits`, `angfits`), (`Qfits`, `Ufits`), (`amp`, `ang`), or (`stQ`, `stU`). The latter two pairs are supposed to be in the numpy.array format. The `ampfactor` argument can be used to adjust the segment length. When `angonly=True`, the segment length is set to be uniform. The `rotation` can be used to rotate the segments in the unit of degrees. The `cutoff` argument specifies the intensity threshold to calculate the amplitude and angle from the Stokes Q and U intensities.