APLpy Documentation

Release 0.9.10

The APLpy Developers

December 11, 2013

Contents

This is the documentation for APLpy. The APLpy homepage is located at http://aplpy.github.com

Part I

Using APLpy

CHAPTER 1

Beginner Tutorial

The following tutorial will take you, step by step, through the main features of APLpy. You will need to download the following file.

First, unpack the example files and go to the `tutorial` directory:

```
tar -xvzf tutorial.tar.gz
cd tutorial
```

Now, start up `ipython`:

```
   ipython --pylab

and import the ``aplpy`` module as follows::

   import aplpy
```

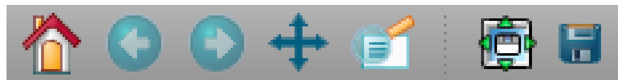
To start off, use the `FITSfigure` class to create a canvas to where your data will be plotted:

```
gc = aplpy.FITSFigure('fits/2MASS_k.fits')
```

and display the image using a grayscale stretch:

```
gc.show_grayscale()
```

Check out the buttons at the top or bottom of the canvas (depending on the version of matplotlib). You will see seven buttons:



The first five are of interest to us here. The button with the magnifying glass will allow you to select an area on the plot and zoom in. To zoom out you can click on the home button (first one on the left). When you're zoomed in you can pan around by clicking on the button with the arrows (fourth button from the left).

Use the following command to show a colorscale image instead:

```
gc.show_colorscale()
```

The colormap used for the colorscale image can be changed. Try the following:

```
gc.show_colorscale(cmap='gist_heat')
```

to show the image showing a 'heat' map. You can find more information in the show_grayscale() and show_colorscale() documentation. For example, you can control the minimum and maximum pixel values to show and the stretch function to use.

It is possible to use APLpy to show 3-color images. To do this, you need a FITS file with the image - this is used for the information relating to the coordinates - and a PNG file containing the 3-color image. Both files need to have exactly the same dimensions and the pixels from the PNG file have to match those from the FITS file. An example is provided in the tutorial files. Try the following:

```
gc.show_rgb('graphics/2MASS_arcsinh_color.png')
```

It is very easy to modify the font properties of the various labels. For example, in this case, we can change the font size of the tick labels to be smaller than the default:

```
gc.tick_labels.set_font(size='small')
```

APLpy can be used to overlay contours onto a grayscale/colorscale/3-color image. Try typing the following command:

```
gc.show_contour('fits/mips_24micron.fits', colors='white')
```

There are a number of arguments that can be passed to show_contour() to control the appearance of the contours as well as the number of levels to show. For more information, see the see the show_contour() documentation.

Display a coordinate grid using:

```
gc.add_grid()
```

and hide it again using:

```
gc.remove_grid()
```

Let's overplot positions from a source list. Here we will use loadtxt to read in the coordinates from a file, but in general you can pass any pair of lists or numpy arrays that are already defined:

```
import numpy
data = numpy.loadtxt('data/yso_wcs_only.txt')
ra, dec = data[:, 0], data[:, 1]
gc.show_markers(ra, dec, edgecolor='green', facecolor='none',
                marker='o', s=10, alpha=0.5)
```

For more information, see the show_markers() documentation.

It's often the case that you might want to change the look of a contour or markers, but if you run show_contour() or show_markers() a second time, it will overplot rather than replacing. To solve this problem APLpy has 'layers' which can be manipulated in a basic way. Type:

```
gc.list_layers()
```

which will print out something like this:

```
There are 2 layers in this figure:

-> contour_set_1
-> marker_set_1
```

You can use remove_layer(), hide_layer(), and show_layer() to manipulate the layers, and you can also specify the layer=name argument to show_contour() or show_markers(). Using the latter forces APLpy to name the layer you are creating with the name provided, and can also be used to replace an existing layer. For example, let's change the color of the markers from green to red:

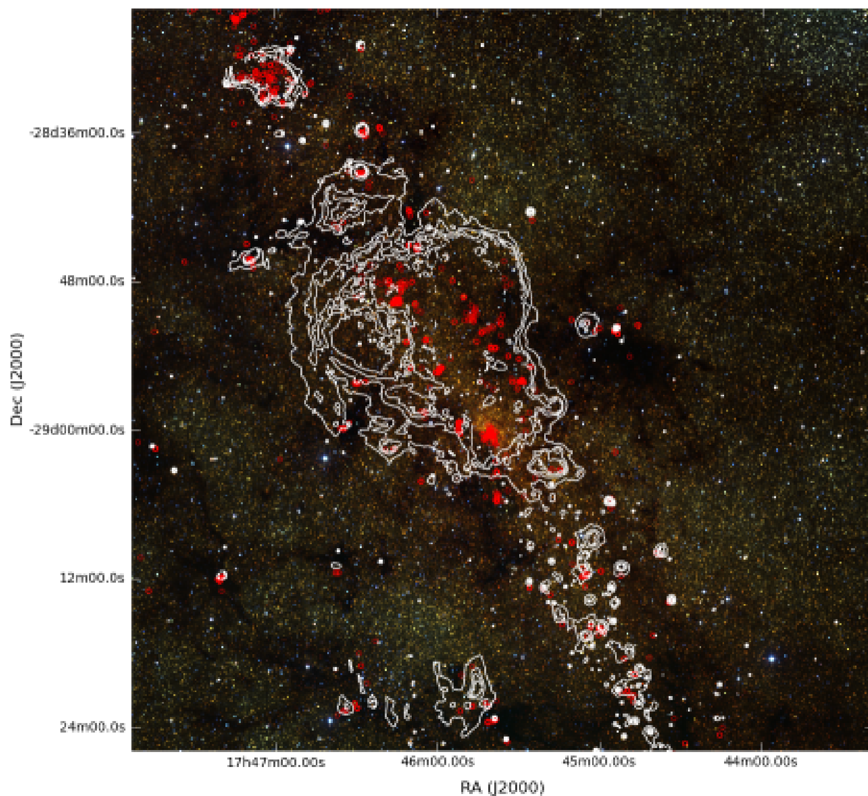
```
gc.show_markers(ra, dec, layer='marker_set_1', edgecolor='red',
                facecolor='none', marker='o', s=10, alpha=0.5)
```

Note the presence of the `layer='marker_set_1'` which means that the current markers plot will be replaced. To view active layers, you can use the `list_layers()` documentation.

To wrap up this tutorial, we will save the plot to a file. Type the following:

```
gc.save('myfirstplot.png')
```

This will produce the following file:



You can of course save it as a PS/EPS, PDF, or SVG file instead. The EPS output is suitable for publication.

To summarize, the above plot was made using the following commands:

```
import aplpy
import numpy

gc = aplpy.FITSFigure('fits/2MASS_k.fits')
gc.show_rgb('graphics/2MASS_arcsinh_color.png')

gc.tick_labels.set_font(size='small')

gc.show_contour('fits/mips_24micron.fits', colors='white')

data = numpy.loadtxt('data/yso_wcs_only.txt')
ra, dec = data[:, 0], data[:, 1]

gc.show_markers(ra, dec, layer='marker_set_1', edgecolor='red',
                facecolor='none', marker='o', s=10, alpha=0.5)
```

```
gc.save('myfirstplot.png')
```

There are many more methods and options, from setting the tick spacing and format to controlling the label fonts. For more information, see the *Quick reference Guide* or the *Reference/API*.

CHAPTER 2

Quick reference Guide

Note: Rather than showing the generic call signature for the methods on this page, these are given in the form of example values. In addition, not all of the optional keywords are mentioned here. For more information on a specific command and all the options available, type `help` followed by the method name, e.g. `help fig.show_grayscale`. When multiple optional arguments are given, this does not mean that all of them have to be specified, but just shows all the options available.

2.1 Introduction

To import APLpy:

```
import aplpy
```

To create a figure of a FITS file:

```
fig = aplpy.FITSFigure('myimage.fits')
```

Here and in the remainder of this reference guide, we use the variable name `fig` for the figure object, but any name can be used.

A grayscale or colorscale representation of the FITS image can be shown and hidden using the following methods:

```
fig.show_grayscale()
fig.hide_grayscale()
fig.show_colorscale()
fig.hide_colorscale()
```

To show a three-color image, use the following method, specifying the filename of the color image:

```
fig.show_rgb('m17.jpeg')
```

The figure can be interactively explored by zooming and panning. To recenter on a specific region programmatically, use the following method, specifying either a radius:

```
fig.recenter(33.23, 55.33, radius=0.3)  # degrees
```

or a separate width and height:

```
fig.recenter(33.23, 55.33, width=0.3, height=0.2)  # degrees
```

To overlay contours, use:

```
fig.show_contour('co_data.fits')
```

To save the current figure, use:

```
fig.save('myplot.eps')
```

Other formats such as PDF, PNG, etc. can be used.

2.2 Labels

Labels can be added either in world coordinates:

```
fig.add_label(34.455, 54.112, 'My favorite star')
```

or relative to the axes:

```
fig.add_label(0.1, 0.9, '(a)', relative=True)
```

2.3 Shapes

To overlay different shapes, the following methods are available:

```
fig.show_markers(x_world, y_world)
fig.show_circles(x_world, y_world, radius)
fig.show_ellipses(x_world, y_world, width, height)
fig.show_rectangles(x_world, y_world, width, height)
fig.show_arrows(x_world, y_world, dx, dy)
```

where `x_world`, `y_world`, `radius`, `width`, `height`, `dx`, and `dy` should be 1D arrays specified in degrees.

It is also possible to plot lines and polygons using:

```
fig.show_lines(line_list)
fig.show_polygons(polygon_list)
```

For these methods, `line_list` and `polygon_list` should be lists of 2xN Numpy arrays describing the coordinates of the vertices in degrees.

2.4 DS9 Regions

DS9 region files can be overlaid with:

```
fig.show_regions('myregions.reg')
```

2.5 Layers

Markers, shapes, regions and text labels are stored in layers. These layers can be listed using:

```
fig.list_layers()
```

Layers can be hidden and shown with the following methods:

```
fig.hide_layer('regions')
fig.show_layer('regions')
```

Any layer can be retrieved using:

```
layer = fig.get_layer('circles')
```

Finally, layers can be removed using:

```
fig.remove_layer('rectangles')
```

2.6 Coordinates

Two methods are provided to help transform coordinates between world and pixel coordinates. These accept either scalars or arrays in degrees:

```
x_pix, y_pix = fig.world2pixel(45.3332, 22.1932)
x_world, y_world = fig.pixel2world(np.array([1., 2., 3]), np.array([1., 3., 5.]))
```

2.7 Frame

To set the look of the frame around the figure, use:

```
fig.frame.set_linewidth(1)  # points
fig.frame.set_color('black')
```

2.8 Colorbar

A grid can be added and removed using the following commands:

```
fig.add_colorbar()
fig.remove_colorbar()
```

Once `add_colorbar()` has been called, the `fig.colorbar` object is created and the following methods are then available:

- Show and hide the colorbar:

  ```
  fig.colorbar.show()
  fig.colorbar.hide()
  ```

- Set where to place the colorbar:

  ```
  fig.colorbar.set_location('right')
  ```

  This can be one of `left`, `right`, `bottom` or `top`.

- Set the width of the colorbar:

```
fig.colorbar.set_width(0.1)  # arbitrary units, default is 0.2
```

- Set the amount of padding between the colorbar and the parent axes:

```
fig.colorbar.set_pad(0.03)  # arbitrary units, default is 0.05
```

- Set the font properties of the labels:

```
fig.colorbar.set_font(size='medium', weight='medium', \
                      stretch='normal', family='sans-serif', \
                      style='normal', variant='normal')
```

- Add a colorbar label:

```
f.colorbar.set_axis_label_text('Flux (Jy/beam)')
```

- Set some of the colorbar label properties:

```
f.colorbar.set_axis_label_font(size=12, weight='bold')
```

- Set the padding between the colorbar and the label in points:

```
f.colorbar.set_axis_label_pad(10)
```

- Change the rotation of the colorbar label, in degrees:

```
f.colorbar.set_axis_label_rotation(270)
```

2.9 Coordinate Grid

A coordinate grid can be added and removed using the following commands:

```
fig.add_grid()
fig.remove_grid()
```

Once `add_grid()` has been called, the `fig.grid` object is created and the following methods are then available:

- Show and hide the grid:

```
fig.grid.show()
fig.grid.hide()
```

- Set the x and y spacing for the grid:

```
fig.grid.set_xspacing(0.2)  # degrees
fig.grid.set_yspacing(0.2)  # degrees
```

- Set the color of the grid lines:

```
fig.grid.set_color('white')
```

- Set the transparency level of the grid lines:

```
fig.grid.set_alpha(0.8)
```

- Set the line style and width for the grid lines:

```
fig.grid.set_linestyle('solid')
fig.grid.set_linewidth(1)  # points
```

2.10 Scalebar

A scalebar can be added and removed using the following commands:

```
fig.add_scalebar()
fig.remove_scalebar()
```

Once `add_scalebar()` has been called, the `fig.scalebar` object is created and the following methods are then available:

- Show and hide the scalebar:

  ```
  fig.scalebar.show(0.2)  # length in degrees
  fig.scalebar.hide()
  ```

- Change the length of the scalebar:

  ```
  fig.scalebar.set_length(0.02)  # degrees
  ```

- Change the label of the scalebar:

  ```
  fig.scalebar.set_label('5 pc')
  ```

- Change the corner that the beam is shown in:

  ```
  fig.scalebar.set_corner('top right')
  ```

  This can be one of `top right`, `top left`, `bottom right`, `bottom left`, `left`, `right`, `bottom` or `top`.

- Set whether or not to show a frame around the beam:

  ```
  fig.scalebar.set_frame(False)
  ```

- Set the transparency level of the scalebar and label:

  ```
  fig.scalebar.set_alpha(0.7)
  ```

- Set the color of the scalebar and label:

  ```
  fig.scalebar.set_color('white')
  ```

- Set the font properties of the label:

  ```
  fig.scalebar.set_font(size='medium', weight='medium', \
                        stretch='normal', family='sans-serif', \
                        style='normal', variant='normal')
  ```

- Set the line style and width for the scalebar:

  ```
  fig.scalebar.set_linestyle('solid')
  fig.scalebar.set_linewidth(3)  # points
  ```

- Set multiple properties at once:

  ```
  fig.scalebar.set(linestyle='solid', color='red', ...)
  ```

2.11 Beam

A beam can be added and removed using the following commands:

```
fig.add_beam()
fig.remove_beam()
```

Once `add_beam()` has been called, the `fig.beam` object is created and the following methods are then available:

- Show and hide the beam:

  ```
  fig.beam.show()
  fig.beam.hide()
  ```

- Change the major and major axes, and the position angle:

  ```
  fig.beam.set_major(0.03)  # degrees
  fig.beam.set_minor(0.02)  # degrees
  fig.beam.set_angle(45.)  # degrees
  ```

- Change the corner that the beam is shown in:

  ```
  fig.beam.set_corner('top left')
  ```

  This can be one of `top right`, `top left`, `bottom right`, `bottom left`, `left`, `right`, `bottom` or `top`.

- Set whether or not to show a frame around the beam:

  ```
  fig.beam.set_frame(False)
  ```

- Set the transparency level of the beam:

  ```
  fig.beam.set_alpha(0.5)
  ```

- Set the color of the whole beam, or the edge and face color individually:

  ```
  fig.beam.set_color('white')
  fig.beam.set_edgecolor('white')
  fig.beam.set_facecolor('green')
  ```

- Set the line style and width for the edge of the beam:

  ```
  fig.beam.set_linestyle('dashed')
  fig.beam.set_linewidth(2)  # points
  ```

- Set the hatch style of the beam:

  ```
  fig.beam.set_hatch('/')
  ```

- Set multiple properties at once:

  ```
  fig.beam.set(facecolor='red', linestyle='dashed', ...)
  ```

2.12 Coordinate types

APLpy supports three types of coordinates: longitudes (in the range 0 to 360 with wrap-around), latitudes (in the range -90 to 90), and scalars (any arbitrary value). APLpy tries to guess the correct type of coordinate for each axis, but it is possible to override this:

```
fig.set_xaxis_coord_type('scalar')
fig.set_yaxis_coord_type('longitude')
```

Valid options are `longitude`, `latitude`, and `scalar`.

2.13 Axis labels

The methods to control the x- and y- axis labels are the following

- Show/hide both axis labels:

```
fig.axis_labels.show()
fig.axis_labels.hide()
```

- Show/hide the x-axis label:

```
fig.axis_labels.show_x()
fig.axis_labels.hide_x()
```

- Show/hide the y-axis label:

```
fig.axis_labels.show_y()
fig.axis_labels.hide_y()
```

- Set the text for the x- and y-axis labels:

```
fig.axis_labels.set_xtext('Right Ascension (J2000)')
fig.axis_labels.set_ytext('Declination (J2000)')
```

- Set the displacement of the x- and y-axis labels from the x- and y-axis respectively:

```
fig.axis_labels.set_xpad(...)
fig.axis_labels.set_ypad(...)
```

- Set where to place the x-axis label:

```
fig.axis_labels.set_xposition('bottom')
```

- Set where to place the y-axis label:

```
fig.axis_labels.set_yposition('right')
```

- Set the font properties of the labels:

```
fig.axis_labels.set_font(size='medium', weight='medium', \
                         stretch='normal', family='sans-serif', \
                         style='normal', variant='normal')
```

2.14 Tick labels

The methods to control the numerical labels below each tick are the following

- Show/hide all tick labels:

```
fig.tick_labels.show()
fig.tick_labels.hide()
```

- Show/hide the x-axis labels:

```
fig.tick_labels.show_x()
fig.tick_labels.hide_x()
```

- Show/hide the y-axis labels:

```
fig.tick_labels.show_y()
fig.tick_labels.hide_y()
```

- Set the format for the x-axis labels (e.g hh:mm, hh:mm:ss.s, etc.):

```
fig.tick_labels.set_xformat('hh:mm:ss.ss')
```

- Set the format for the y-axis labels (e.g dd:mm, dd:mm:ss.s, etc.):

```
fig.tick_labels.set_yformat('dd:mm:ss.s')
```

- Set where to place the x-axis tick labels:

```
fig.tick_labels.set_xposition('top')
```

- Set where to place the y-axis tick labels:

```
fig.tick_labels.set_yposition('left')
```

- Set the style of the labels (‘colons’ or ‘plain’):

```
fig.tick_labels.set_style('colons')
```

- Set the font properties of the labels:

```
fig.tick_labels.set_font(size='medium', weight='medium', \
                         stretch='normal', family='sans-serif', \
                         style='normal', variant='normal')
```

2.15 Ticks

The methods to control properties relating to the tick marks are the following:

- Show/hide all ticks:

```
fig.ticks.show()
fig.ticks.hide()
```

- Show/hide the x-axis ticks:

```
fig.ticks.show_x()
fig.ticks.hide_x()
```

- Show/hide the y-axis ticks:

```
fig.ticks.show_y()
fig.ticks.hide_y()
```

- Change the tick spacing for the x- and y-axis:

```
fig.ticks.set_xspacing(0.04)  # degrees
fig.ticks.set_yspacing(0.03)  # degrees
```

- Change the length of the ticks:

```
fig.ticks.set_length(10)  # points
```

- Change the color of the ticks:

```
fig.ticks.set_color('black')
```

- Set the line width for the ticks:

```
fig.ticks.set_linewidth(2)  # points
```

- Set the number of minor ticks per major tick:

```
fig.ticks.set_minor_frequency(5)
```

Set this to 1 to get rid of minor ticks

2.16 Advanced

To control whether to refresh the display after each command, use the following method:

```
fig.set_auto_refresh(False)
```

To force a refresh, use:

```
fig.refresh()
```

To use the system LaTeX instead of the matplotlib LaTeX, use:

```
fig.set_system_latex(True)
```

The color for NaN values can be controlled using the following method:

```
fig.set_nan_color('black')
```

Finally, to change the look of the plot using pre-set themes, use:

```
fig.set_theme('publication')
```

CHAPTER 3

Arbitrary coordinate systems

3.1 Coordinate type

In addition to standard longitude/latitude coordinates, APLpy supports any valid quantities for coordinates (e.g. velocity, frequency, ...) as long as the WCS header information is valid. To differentiate between longitudes, latitudes, and arbitrary scalar values, APLpy keeps track of the axis coordinate 'type' for each axis, and will try and guess these based on the header. However, it is possible to explicitly specify the coordinate type with the `set_xaxis_coord_type()` and `set_yaxis_coord_type()` methods in the `FITSFigure` object:

```
f = FITSFigure('2MASS_k.fits')
f.set_xaxis_coord_type('scalar')
f.set_xaxis_coord_type('latitude')
```

Valid coordinate types are `longitude`, `latitude`, and `scalar`. Longitudes are forced to be in the 0 to 360 degree range, latitudes are forced to be in the -90 to 90 degree range, and scalars are not constrained.

3.2 Label formatting

How label formats are specified depends on the coordinate type. If the coordinate is a longitude or latitude, then the label format is specified using a special syntax which describes whether the label should be decimal or sexagesimal, in hours or degrees, and indicates the number of decimal places. For example:

- `ddd.ddddd` means decimal degrees, where the number of decimal places can be varied
- hh or dd means hours (or degrees)
- `hh:mm` or `dd:mm` means hours and minutes (or degrees and arcminutes)
- `hh:mm:ss` or `dd:mm:ss` means hours, minutes, and seconds (or degrees, arcminutes, and arcseconds)
- `hh:mm:ss.ss` or `dd:mm:ss.ss` means hours, minutes, and seconds (or degrees, arcminutes, and arcseconds), where the number of decimal places can be varied.

If the coordinate type is scalar, then the format should be specified as a valid Python format. For example, `%g` is the default Python format, `%10.3f` means decimal notation with three decimal places, etc. For more information, see String Formatting Operations.

In both cases, the default label format can be overridden:

```
f.tick_labels.set_xformat('dd.ddddd')
f.tick_labels.set_yformat('%11.3f')
```

3.3 Aspect ratio

When plotting images in sky coordinates, APLpy makes pixel square by default, but it is possible to change this, which can be useful for non-sky coordinates. When calling `show_grayscale()` or `show_colorscale()`, simply add `aspect='auto'` which will override the `aspect='equal'` default.

CHAPTER 4

Slicing multi-dimensional data cubes

4.1 How to slice data cubes

APLpy supports extracting a slice from n-dimensional FITS cubes, and re-ordering dimensions. The two key arguments to `FITSFigure` to control this are `dimensions` and `slices`. These arguments can also be passed to `show_contour()`.

The `dimensions` argument is used to specify which dimensions should be used for the x- and y-axis respectively (zero based). The default values are `[0, 1]` which means that the x-axis should use the first dimension in the FITS cube, and the y-axis should use the second dimension. For a 2-dimensional FITS file, this means that one can use `[1, 0]` to flip the axes. For a FITS cube with R.A., Declination, and Velocity, `[0, 2]` would make a R.A.-Velocity plot.

The `slices` argument gives the pixels slice to extract from the remaining dimensions, skipping the dimensions used, so `slices` should be a list with length n-2 where n is the number of dimensions in the FITS file. For example, if one has a FITS file with R.A., Declination, Velocity, and Time (in that order), then:

- `dimensions=[0, 1]` means the plot will be an R.A-Declination plot, and `slices=[33, 56]` means that pixel slices 33 and 56 will be used in Velocity and Time respectively (in this case, `dimensions` does not need to be specified since it defaults to `[0, 1]`)
- `dimensions=[0, 2]` means the plot will be an R.A-Velocity plot, and `slices=[22, 56]` means that pixel slices 22 and 56 will be used in Declination and Time respectively.
- `dimensions=[3, 2]` means the plot will be a Time-Velocity plot, and `slices=[10, 22]` means that pixel slices 10 and 22 will be used in R.A and Declination respectively.

See *Arbitrary coordinate systems* for information on formatting the labels when non-longitude/latitude coordinates are used.

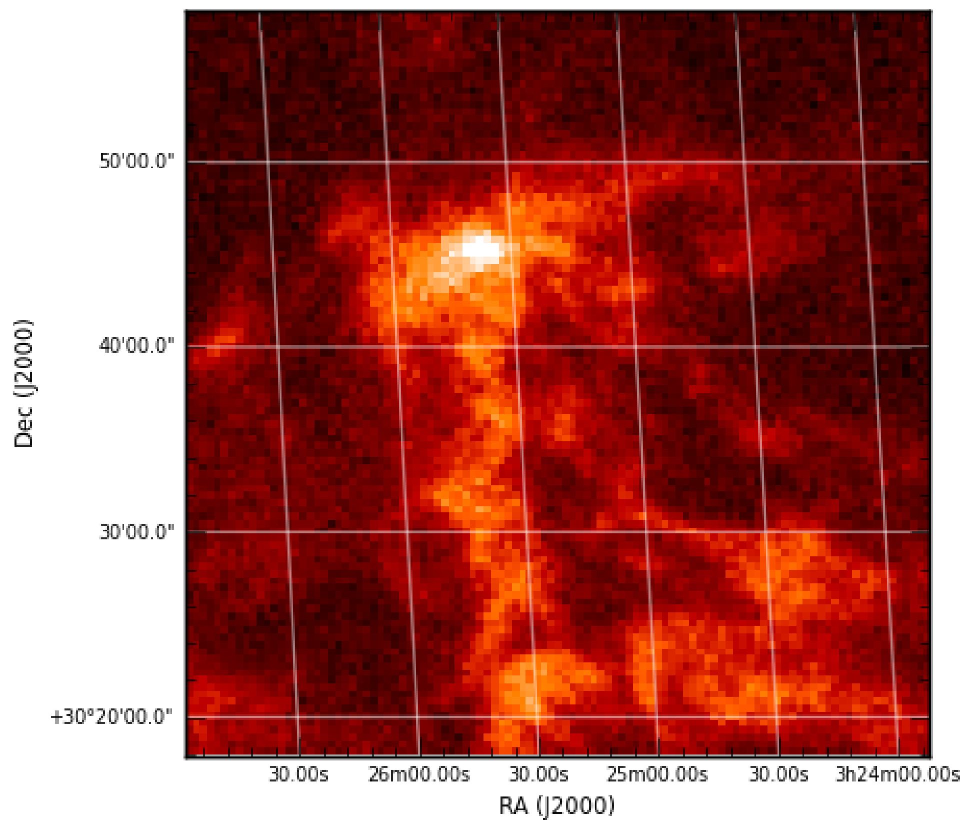
4.2 Aspect ratio

When plotting images in sky coordinates, APLpy makes pixel square by default, but it is possible to change this. When calling `show_grayscale()` or `show_colorscale()`, simply add `aspect='auto'` which will override the `aspect='equal'` default. The `aspect='auto'` is demonstrated below.

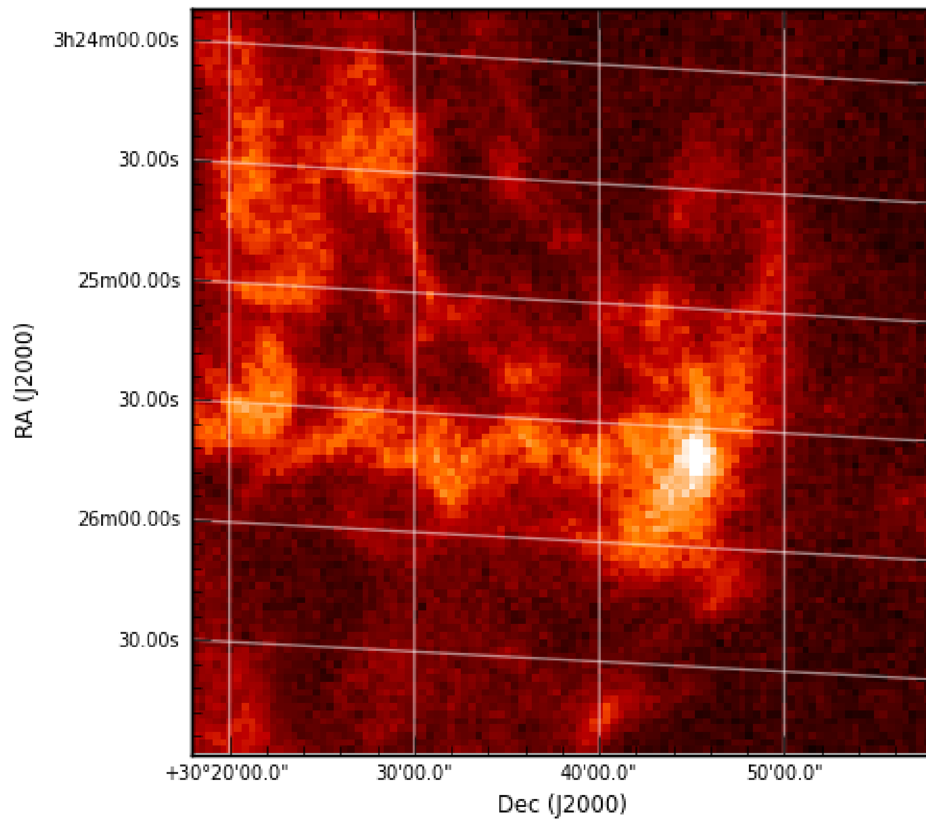
4.3 Example

The following script demonstrates this functionality in use:

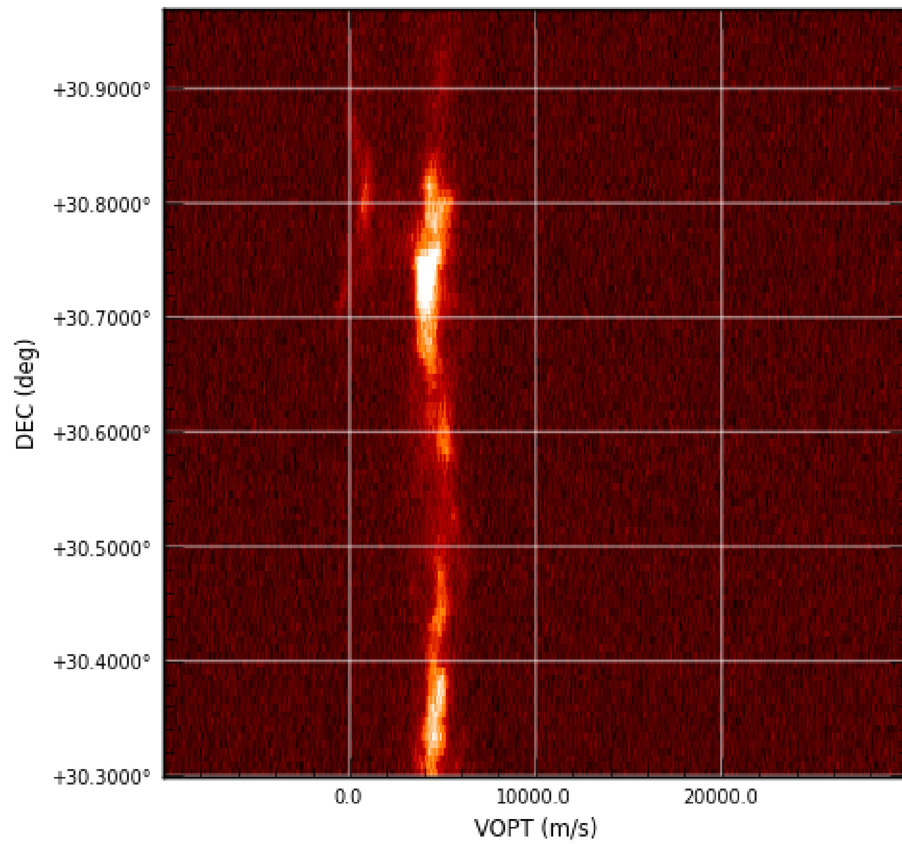
```
import matplotlib
matplotlib.use('Agg')

import aplpy

f = aplpy.FITSFigure('L1448_13CO.fits.gz', slices=[222],
                     figsize=(5,5))
f.show_colorscale()
f.add_grid()
f.tick_labels.set_font(size='xx-small')
f.axis_labels.set_font(size='x-small')
f.save('slicing_1.png')
```




```
f = aplpy.FITSFigure('L1448_13CO.fits.gz', slices=[222],
                     dimensions=[1, 0], figsize=(5,5))
f.show_colorscale()
f.add_grid()
f.tick_labels.set_font(size='xx-small')
f.axis_labels.set_font(size='x-small')
f.save('slicing_2.png')
```




```
f = aplpy.FITSFigure('L1448_13CO.fits.gz', dimensions=[2, 1],
                     slices=[50], figsize=(5,5))
f.show_colorscale(aspect='auto')
f.add_grid()
f.tick_labels.set_font(size='xx-small')
f.axis_labels.set_font(size='x-small')
f.tick_labels.set_xformat('%.1f')
f.save('slicing_3.png')
```


+30.9000°
+30.8000°
+30.7000°
+30.6000°
+30.5000°
+30.4000°
+30.3000°
DEC (deg)
0.0
10000.0
20000.0
VOPT (m/s)

CHAPTER 5

HOWTOs

5.1 APLpy in non-interactive mode

While APLpy can be easily used to interactively make plots, it is also possible to make plots without opening up a display. This can be useful to run APLpy remotely, or to write non-interactive scripts to plot one or many FITS files.

5.1.1 Running APLpy in non-interactive mode

To run APLpy in non-interactive mode, you will need to change the 'backend' used by matplotlib from an interactive (e.g. WxAgg, TkAgg, MacOS X) to a non-interactive (e.g. Agg, Cairo, PS, PDF) backend. The default backend is typically set in your `.matplotlibrc` file (or if you do not have such a file, an interactive backend is usually chosen by default). The easiest way to change the backend temporarily is to use the `matplotlib.use()` function:

```
import matplotlib
matplotlib.use('Agg')
```

It is important to change the backend via `matplotlib.use` before the `aplpy` module is imported. Once the backend has been changed, any call to `FITSFigure()` will no longer make a figure window appear. The following script can be used to make a PNG plot:

```
import matplotlib
matplotlib.use('Agg')

import aplpy

f = aplpy.FITSFigure('mips_24micron.fits')
f.show_grayscale()
f.save('mips_24.png')
```

5.1.2 Pan/Zoom in non-interactive mode

One of the advantages of using APLpy in interactive mode is the ability to zoom in on a given region of interest. To replicate this functionality in non-interactive mode, the `FITSFigure.recenter()` method can be used. This method takes a central position, and either a radius (to make a square plot) or a width/height (to make a rectangular plot). This is illustrated in the following example:

```
import matplotlib
matplotlib.use('Agg')

import aplpy

f = aplpy.FITSFigure('mips_24micron.fits')
f.show_grayscale()
f.recenter(266.2142,-29.1832,width=0.5,height=0.3)
f.save('mips_24_zoomin.png')
```

5.1.3 Batch scripting

Using APLpy non-interactively can be useful for making plots of many FITS files. Given a directory `fits/` containing FITS files, the following code will generate on plot for each `.fits` file:

```
import matplotlib
matplotlib.use('Agg')

import aplpy

import glob
import os

for fits_file in glob.glob(os.path.join('fits/','*.fits')):

    f = aplpy.FITSFigure(fits_file)
    f.show_grayscale()
    f.save(fits_file.replace('.fits','.png'))
    f.close()
```

5.2 Installing Montage

Montage is a package developed by IPAC designed to handle the reprojection of FITS files. APLpy relies on Montage for several functions, such as `make_rgb_cube()` or the `north=True` argument in `FITSFigure`. To use these features, Montage needs to be installed.

5.2.1 Obtaining and installing Montage

Montage can be downloaded from http://montage.ipac.caltech.edu/docs/download.html

Installation instructions are provided at http://montage.ipac.caltech.edu/docs/build.html

Once Montage is correctly installed, you are ready to use the Montage-dependent features of APLpy!

5.3 Making RGB images

While the APLpy `show_rgb()` can be used to display an RGB image with the same projection as a given FITS file, it is also possible to use the `make_rgb_cube()` and `make_rgb_image()` to generate RGB images from scratch, even for files with initially different projections/resolutions.

5.3.1 Reprojecting three images to the same projection

If you are starting from three images with different projections/resolutions, the first step is to reproject these to a common projection/resolution. The following code shows how this can be done using the `make_rgb_cube()` function:

```
aplpy.make_rgb_cube(['2mass_k.fits', '2mass_h.fits',
                     '2mass_j.fits'], '2mass_cube.fits')
```

This method makes use of Montage to reproject the images to a common projection. For more information on installing Montage, see here. The above example produces a FITS cube named `2mass_cube.fits` which contains the three channels in the same projection. This can be used to then produce an RGB image (see next section)

5.3.2 Producing an RGB image from images in a common projection

The `make_rgb_image()` function can be used to produce an RGB image from either three FITS files in the exact same projection, or a FITS cube containing the three channels (such as that output by `make_rgb_cube()`). The following example illustrates how to do this:

```
aplpy.make_rgb_image('2mass_cube.fits','2mass_rgb.png')
```

In the above example, APLpy will automatically adjust the stretch of the different channels to try and produce a reasonable image, but it is of course possible to specify one or more of the lower/upper limits of the scale to use for each channel. For each lower/upper scale limit, this can be done in two ways. The first is to use the vmin_r/g/b or vmax_r/g/b argument which specifies the pixel value to use for the lower or upper end of the scale respectively. The alternative is to use the pmin_r/g/b or pmax_r/g/b arguments, which specify the percentile value to use instead of an absolute value. The two can of course be mixed, such as in the following example:

```
aplpy.make_rgb_image('2mass_cube.fits', '2mass_rgb.png',
                      vmin_r=0., pmax_g=90.)
```

5.3.3 Plotting the resulting RGB image

If PyAVM is installed (which is recommended but not required), then if you produce a JPEG or PNG image, you can plot it directly with:

```
f = aplpy.FITSFigure('2mass_rgb.png')
f.show_rgb()
```

For more information, see *AVM tagging*.

On the other hand, if you are not able to use PyAVM, or need to use a format other than JPEG or PNG, then you need to instantiate the `FITSFigure` class with a FITS file with exactly the same dimensions and WCS as the RGB image. The `make_rgb_cube()` function will in fact produce a file with the same name as the main output, but including a `_2d` suffix, and this can be used to instantiate `FITSFigure`:

```
# Reproject the images to a common projection - this will also produce
# ``2mass_cube_2d.fits``
aplpy.make_rgb_cube(['2mass_k.fits', '2mass_h.fits',
                     '2mass_j.fits'], '2mass_cube.fits')

# Make an RGB image
aplpy.make_rgb_image('2mass_cube.fits', '2mass_rgb.png')

# Plot the RGB image using the 2d image to indicate the projection
f = aplpy.FITSFigure('2mass_cube_2d.fits')
f.show_rgb('2mass_rgb.png')
```

5.4 AVM tagging

The latest developer version of APLpy has support for reading/writing AVM tags from/to RGB images (via PyAVM). To make a plot with an AVM-tagged RGB image, say 'example.jpg', you can use:

```
import aplpy
f = aplpy.FITSFigure('example.jpg')
f.show_rgb()
```

Note that no filename is required for `show_rgb()` in this case.

If PyAVM is installed, `make_rgb_image()` can embed AVM meta-data into RGB images it creates, although only JPEG and PNG files support this:

```
import aplpy
aplpy.make_rgb_image('2mass_cube.fits', '2mass_rgb.jpg')
```

If PyAVM 0.9.1 or later is installed, the above file `2mass_rgb.jpg` will include AVM meta-data and can then be plotted with:

```
import aplpy
f = aplpy.FITSFigure('2mass_rgb.jpg')
f.show_rgb()
```

In other words, this means that when creating an RGB image with APLpy, it is no longer necessary to initialize `FITSFigure` with a FITS file and then use `show_rgb()` with the RGB image filename - instead, `FITSFigure` can be directly initialized with the AVM-tagged image.

To disable the embedding of AVM tags, you can use the `embed_avm_tags=False` option for `make_rgb_image()`.

These features require PyAVM 0.9.1 or later. Please report any issues you encounter here.

5.5 Creating subplots

By default, `FITSFigure` creates a figure with a single subplot that occupies the entire figure. However, APLpy can be used to place a subplot in an existing matplotlib figure instance. To do this, `FITSFigure` should be called with the `figure=` argument as follows:

```
import aplpy
import matplotlib.pyplot as mpl

 fig = mpl.figure()
 f = aplpy.FITSFigure('some_image.fits', figure=fig)
```

The above will place a subplot inside the `fig` figure instance. The `f` object can be used as normal to control the FITS figure inside the subplot. The above however is not very insteresting compared to just creating a FITSFigure instance from scratch. What this is useful for is only using sub-regions of the figure to display the FITS data, to leave place for other subplots, whether histograms, scatter, or other matplotlib plots, or another FITS Figure. This can be done using the `subplot` argument. From the docstring for FITSFigure:

```
*subplot*: [ list of four floats ]
    If specified, a subplot will be added at this position. The list
    should contain [xmin, ymin, dx, dy] where xmin and ymin are the
    position of the bottom left corner of the subplot, and dx and dy are
    the width and height of the subplot respectively. These should all be
    given in units of the figure width and height. For example, [0.1, 0.1,
```

```
0.8, 0.8] will almost fill the entire figure, leaving a 10 percent
margin on all sides.
```

The following code outline illustrates how to create a rectangular figure with two FITS images:

```
import aplpy
import matplotlib.pyplot as mpl

fig = mpl.figure(figsize=(15, 7))

f1 = aplpy.FITSFigure('image_1.fits', figure=fig, subplot=[0.1,0.1,0.35,0.8])
f1.set_tick_labels_font(size='x-small')
f1.set_axis_labels_font(size='small')
f1.show_grayscale()

f2 = aplpy.FITSFigure('image_2.fits', figure=fig, subplot=[0.5,0.1,0.35,0.8])
f2.set_tick_labels_font(size='x-small')
f2.set_axis_labels_font(size='small')
f2.show_grayscale()

f2.hide_yaxis_label()
f2.hide_ytick_labels()

fig.canvas.draw()
```

The `hide` methods shown above are especially useful when working with subplots, as in some cases there is no need to repeat the tick labels. Alternatively figures can be constructed from both APLpy figures and normal matplotlib axes:

```
import aplpy
import matplotlib.pyplot as mpl

fig = mpl.figure(figsize=(15, 7))

f1 = aplpy.FITSFigure('image_1.fits', figure=fig, subplot=[0.1,0.1,0.35,0.8])
f1.set_tick_labels_font(size='x-small')
f1.set_axis_labels_font(size='small')
f1.show_grayscale()

ax2 = fig.add_axes([0.5,0.1,0.35,0.8])

# some code here with ax2

fig.canvas.draw()
```

CHAPTER 6

APLpy Normalizations

APLpy uses a different set of normalization techniques than other image display tools. The default options available for the `stretch` argument are `linear`, `sqrt`, `power`, `log`, and `arcsinh`.

In all cases, the transformations below map the values from the image onto a scale from 0 to 1 which then corresponds to the endpoints of the colormap being used.

6.1 Log Scale

APLpy defines the log scale with a `vmid` parameter such that

$$y = \frac{\log_{10}(x \cdot (m-1) + 1)}{\log_{10}(m)}$$

where:

$$m = \frac{v_{\rm max} - v_{\rm mid}}{v_{\rm min} - v_{\rm mid}}$$

and:

$$x = \frac{v - v_{\rm min}}{v_{\rm max} - v_{\rm min}}$$

and v is the image pixel values. Note that $v_{\rm mid}$ should be smaller than $v_{\rm min}$, which means that m will always be larger than one. By default, $v_{\rm mid} = 0$ and $v_{\rm min}$ is therefore required to be positive (but negative values of $v_{\rm min}$ are allowed if $v_{\rm mid}$ is specified).

For reference, in the FITS display program ds9, the log scale is defined by

$$y = \frac{\log_{10}(ax + 1)}{\log_{10}(a)}$$

where a defaults to 1000 and recommended values are from 100-10000 (see here).

The two stretches are *nearly* (but not completely) identical for $m \approx a + 1$. If you want to convert from the ds9 a parameter to APLpy's $v_{\rm mid}$, you can use this formula:

$$v_{\rm mid} = \frac{m \cdot v_{\rm min} - v_{\rm max}}{m - 1} \approx \frac{(a+1) \cdot v_{\rm min} - v_{\rm max}}{a}$$

6.2 Arcsinh Scale

The arcsinh scale is defined by

$$y = \frac{{\rm asinh}(x/m)}{{\rm asinh}(1/m)}$$

where

$$m = \frac{v_{\rm mid} - v_{\rm min}}{v_{\rm max} - v_{\rm min}}$$

By default, $v_{\rm mid}$ is chosen such that $1/m = -30$.

For reference, the ds9 definition is simply

$$y = \frac{{\rm asinh}(10x)}{3}$$

6.3 Linear Scale

The linear scale is only concerned with the `vmin` and `vmax` keywords:

$$y = \frac{x - v_{\rm min}}{v_{\rm max} - v_{\rm min}}$$

6.4 Square Root Scale

The square root scale is given by

$$y = \sqrt{\frac{x - v_{\rm min}}{v_{\rm max} - v_{\rm min}}}$$

6.5 Power Scale

The power root scale is

$$y = \left[\frac{x - v_{\rm min}}{v_{\rm max} - v_{\rm min}}\right]^a$$

where a is passed in as the `exponent` keyword.

Part II

Reference/API

CHAPTER 7

aplpy Module

7.1 Functions

`make_rgb_cube`(files, output[, north, ...])	Make an RGB data cube from a list of three FITS images.
`make_rgb_image`(data, output[, indices, ...])	Make an RGB image from a FITS RGB cube or from three FITS files.
`test`([package, test_path, args, plugins, ...])	Run the tests using py.test.

7.1.1 make_rgb_cube

`aplpy.rgb.make_rgb_cube`(*files*, *output*, *north=False*, *system=None*, *equinox=None*)

Make an RGB data cube from a list of three FITS images.

This method can read in three FITS files with different projections/sizes/resolutions and uses Montage to reproject them all to the same projection.

Two files are produced by this function. The first is a three-dimensional FITS cube with a filename give by `output`, where the third dimension contains the different channels. The second is a two-dimensional FITS image with a filename given by `output` with a `_2d` suffix. This file contains the mean of the different channels, and is required as input to FITSFigure if show_rgb is subsequently used to show a color image generated from the FITS cube (to provide the correct WCS information to FITSFigure).

Parameters

files : tuple or list

A list of the filenames of three FITS filename to reproject. The order is red, green, blue.

output : str

The filename of the output RGB FITS cube.

north : bool, optional

By default, the FITS header generated by Montage represents the best fit to the images, often resulting in a slight rotation. If you want north to be straight up in your final mosaic, you should use this option.

system : str, optional

Specifies the system for the header (default is EQUJ). Possible values are: EQUJ EQUB ECLJ ECLB GAL SGAL

equinox : str, optional

If a coordinate system is specified, the equinox can also be given in the form YYYY. Default is J2000.

7.1.2 make_rgb_image

`aplpy.rgb.make_rgb_image`(*data*, *output*, *indices=(0, 1, 2)*, *vmin_r=None*, *vmax_r=None*, *pmin_r=0.25*, *pmax_r=99.75*, *stretch_r='linear'*, *vmid_r=None*, *exponent_r=2*, *vmin_g=None*, *vmax_g=None*, *pmin_g=0.25*, *pmax_g=99.75*, *stretch_g='linear'*, *vmid_g=None*, *exponent_g=2*, *vmin_b=None*, *vmax_b=None*, *pmin_b=0.25*, *pmax_b=99.75*, *stretch_b='linear'*, *vmid_b=None*, *exponent_b=2*, *embed_avm_tags=True*)

Make an RGB image from a FITS RGB cube or from three FITS files.

Parameters

data : str or tuple or list

If a string, this is the filename of an RGB FITS cube. If a tuple or list, this should give the filename of three files to use for the red, green, and blue channel.

output : str

The output filename. The image type (e.g. PNG, JPEG, TIFF, ...) will be determined from the extension. Any image type supported by the Python Imaging Library can be used.

indices : tuple, optional

If data is the filename of a FITS cube, these indices are the positions in the third dimension to use for red, green, and blue respectively. The default is to use the first three indices.

vmin_r, vmin_g, vmin_b : float, optional

Minimum pixel value to use for the red, green, and blue channels. If set to None for a given channel, the minimum pixel value for that channel is determined using the corresponding pmin_x argument (default).

vmax_r, vmax_g, vmax_b : float, optional

Maximum pixel value to use for the red, green, and blue channels. If set to None for a given channel, the maximum pixel value for that channel is determined using the corresponding pmax_x argument (default).

pmin_r, pmin_r, pmin_g : float, optional

Percentile values used to determine for a given channel the minimum pixel value to use for that channel if the corresponding vmin_x is set to None. The default is 0.25% for all channels.

pmax_r, pmax_g, pmax_b : float, optional

Percentile values used to determine for a given channel the maximum pixel value to use for that channel if the corresponding vmax_x is set to None. The default is 99.75% for all channels.

stretch_r, stretch_g, stretch_b : { 'linear', 'log', 'sqrt', 'arcsinh', 'power' }

The stretch function to use for the different channels.

vmid_r, vmid_g, vmid_b : float, optional

Baseline values used for the log and arcsinh stretches. If set to None, this is set to zero for log stretches and to vmin - (vmax - vmin) / 30. for arcsinh stretches

exponent_r, exponent_g, exponent_b : float, optional

If stretch_x is set to 'power', this is the exponent to use.

embed_avm_tags : bool, optional

Whether to embed AVM tags inside the image - this can only be done for JPEG and PNG files, and only if PyAVM is installed.

7.1.3 test

`aplpy.test`(*package=None*, *test_path=None*, *args=None*, *plugins=None*, *verbose=False*, *pastebin=None*, *remote_data=False*, *pep8=False*, *pdb=False*, *coverage=False*, *open_files=False*, ***kwargs*)

Run the tests using py.test. A proper set of arguments is constructed and passed to `pytest.main`.

Parameters

package : str, optional

The name of a specific package to test, e.g. 'io.fits' or 'utils'. If nothing is specified all default tests are run.

test_path : str, optional

Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.

args : str, optional

Additional arguments to be passed to `pytest.main` in the `args` keyword argument.

plugins : list, optional

Plugins to be passed to `pytest.main` in the `plugins` keyword argument.

verbose : bool, optional

Convenience option to turn on verbose output from py.test. Passing True is the same as specifying `-v` in `args`.

pastebin : {'failed','all',None}, optional

Convenience option for turning on py.test pastebin output. Set to 'failed' to upload info for failed tests, or 'all' to upload info for all tests.

remote_data : bool, optional

Controls whether to run tests marked with @remote_data. These tests use online data and are not run by default. Set to True to run these tests.

pep8 : bool, optional

Turn on PEP8 checking via the pytest-pep8 plugin and disable normal tests. Same as specifying `--pep8 -k pep8` in `args`.

pdb : bool, optional

Turn on PDB post-mortem analysis for failing tests. Same as specifying `--pdb` in `args`.

coverage : bool, optional

Generate a test coverage report. The result will be placed in the directory htmlcov.

open_files : bool, optional

Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Works only on platforms with a working `lsof` command.

parallel : int, optional

When provided, run the tests in parallel on the specified number of CPUs. If parallel is negative, it will use the all the cores on the machine. Requires the `pytest-xdist` plugin is installed. Only available when using Astropy 0.3 or later.

kwargs :

Any additional keywords passed into this function will be passed on to the astropy test runner. This allows use of test-related functionality implemented in later versions of astropy without explicitly updating the package template.

See Also:

`pytest.main`
py.test function wrapped by `run_tests`.

7.2 Classes

`AxisLabels`(parent)	
`Beam`(parent)	
`Colorbar`(parent)	
`FITSFigure`(data[, hdu, figure, subplot, ...])	A class for plotting FITS files.
`Frame`(parent)	
`Grid`(parent)	
`Scalebar`(parent)	
`TickLabels`(parent)	
`Ticks`(parent)	

7.2.1 AxisLabels

class `aplpy.axis_labels.AxisLabels`(*parent*)
Bases: `object`

Methods Summary

`hide`()	Hide the x- and y-axis labels.
`hide_x`()	Hide the x-axis label.
`hide_y`()	Hide the y-axis label.
`set_font`([family, style, variant, stretch, ...])	Set the font of the axis labels.
`set_xpad`(pad)	Set the x-axis label displacement, in points.
`set_xposition`(position)	Set the position of the x-axis label (‘top’ or ‘bottom’)
`set_xtext`(label)	Set the x-axis label text.
`set_ypad`(pad)	Set the y-axis label displacement, in points.
`set_yposition`(position)	Set the position of the y-axis label (‘left’ or ‘right’)
`set_ytext`(label)	Set the y-axis label text.
`show`()	Show the x- and y-axis labels.

Continued on next page

Table 7.3 – continued from previous page

show_x()	Show the x-axis label.
show_y()	Show the y-axis label.

Methods Documentation

`hide()`
Hide the x- and y-axis labels.

`hide_x()`
Hide the x-axis label.

`hide_y()`
Hide the y-axis label.

`set_font(`*family=None*, *style=None*, *variant=None*, *stretch=None*, *weight=None*, *size=None*, *fontproperties=None*`)`
Set the font of the axis labels.

Parameters

family : str, optional

The family of the font to use. This can either be a generic font family name, either 'serif', 'sans-serif', 'cursive', 'fantasy', or 'monospace', or a list of font names in decreasing order of priority.

style : str, optional

The font style. This can be 'normal', 'italic' or 'oblique'.

variant : str, optional

The font variant. This can be 'normal' or 'small-caps'

stretch : str or int or float, optional

The stretching (spacing between letters) for the font. This can either be a numeric value in the range 0-1000 or one of 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded' or 'ultra-expanded'.

weight : str or int or float, optional

The weight (or boldness) of the font. This can either be a numeric value in the range 0-1000 or one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

size : str or int or float, optional

The size of the font. This can either be a numeric value (e.g. 12), giving the size in points, or one of 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', or 'xx-large'.

Notes

Default values are set by matplotlib or previously set values if set_font has already been called. Global default values can be set by editing the matplotlibrc file.

`set_xpad(`*pad*`)`
Set the x-axis label displacement, in points.

`set_xposition(position)`
Set the position of the x-axis label ('top' or 'bottom')

`set_xtext(label)`
Set the x-axis label text.

`set_ypad(pad)`
Set the y-axis label displacement, in points.

`set_yposition(position)`
Set the position of the y-axis label ('left' or 'right')

`set_ytext(label)`
Set the y-axis label text.

`show()`
Show the x- and y-axis labels.

`show_x()`
Show the x-axis label.

`show_y()`
Show the y-axis label.

7.2.2 Beam

class `aplpy.overlays.Beam(parent)`
Bases: `object`

Methods Summary

hide()	Hide the beam
set(**kwargs)	Modify the beam properties.
set_alpha(alpha)	Set the alpha value (transparency).
set_angle(angle)	Set the position angle of the beam on the sky, in degrees.
set_borderpad(borderpad)	Set the amount of padding within the beam object, relative to the canvas size.
set_color(color)	Set the beam color.
set_corner(corner)	Set the beam location.
set_edgecolor(edgecolor)	Set the color for the edge of the beam.
set_facecolor(facecolor)	Set the color for the interior of the beam.
set_frame(frame)	Set whether to display a frame around the beam.
set_hatch(hatch)	Set the hatch pattern.
set_linestyle(linestyle)	Set the line style for the edge of the beam.
set_linewidth(linewidth)	Set the line width for the edge of the beam, in points.
set_major(major)	Set the major axis of the beam, in degrees.
set_minor(minor)	Set the minor axis of the beam, in degrees.
set_pad(pad)	Set the amount of padding between the beam object and the image corner/edge, relati
show([major, minor, angle, corner, frame, ...])	Display the beam shape and size for the primary image.

Methods Documentation

`hide()`
Hide the beam

`set(**kwargs)`
Modify the beam properties. All arguments are passed to the matplotlib Ellipse classe. See the matplotlib documentation for more details.

`set_alpha(alpha)`
Set the alpha value (transparency).

This should be a floating point value between 0 and 1.

`set_angle(angle)`
Set the position angle of the beam on the sky, in degrees.

`set_borderpad(borderpad)`
Set the amount of padding within the beam object, relative to the canvas size.

`set_color(color)`
Set the beam color.

`set_corner(corner)`
Set the beam location.

Acceptable values are ‘left’, ‘right’, ‘top’, ‘bottom’, ‘top left’, ‘top right’, ‘bottom left’ (default), and ‘bottom right’.

`set_edgecolor(edgecolor)`
Set the color for the edge of the beam.

`set_facecolor(facecolor)`
Set the color for the interior of the beam.

`set_frame(frame)`
Set whether to display a frame around the beam.

`set_hatch(hatch)`
Set the hatch pattern.

This should be one of ‘/’, ‘’, ‘|’, ‘-‘, ‘+’, ‘x’, ‘o’, ‘O’, ‘.’, or ‘*’.

`set_linestyle(linestyle)`
Set the line style for the edge of the beam.

This should be one of ‘solid’, ‘dashed’, ‘dashdot’, or ‘dotted’.

`set_linewidth(linewidth)`
Set the line width for the edge of the beam, in points.

`set_major(major)`
Set the major axis of the beam, in degrees.

`set_minor(minor)`
Set the minor axis of the beam, in degrees.

`set_pad(pad)`
Set the amount of padding between the beam object and the image corner/edge, relative to the canvas size.

`show(major='BMAJ', minor='BMIN', angle='BPA', corner='bottom left', frame=False, borderpad=0.4, pad=0.5, **kwargs)`
Display the beam shape and size for the primary image.

By default, this method will search for the BMAJ, BMIN, and BPA keywords in the FITS header to set the major and minor axes and the position angle on the sky.

Parameters
major : float, optional

Major axis of the beam in degrees (overrides BMAJ if present)

minor : float, optional

Minor axis of the beam in degrees (overrides BMIN if present)

angle : float, optional

Position angle of the beam on the sky in degrees (overrides BPA if present) in the anti-clockwise direction.

corner : int, optional

The beam location. Acceptable values are 'left', 'right', 'top', 'bottom', 'top left', 'top right', 'bottom left' (default), and 'bottom right'.

frame : str, optional

Whether to display a frame behind the beam (default is False)

kwargs :

Additional arguments are passed to the matplotlib Ellipse classe. See the matplotlib documentation for more details.

7.2.3 Colorbar

class `aplpy.colorbar.Colorbar`(*parent*)

Bases: `object`

Methods Summary

`hide`()	
`set_axis_label_font`([family, style, ...])	Set the font of the tick labels.
`set_axis_label_pad`(axis_label_pad)	Set the colorbar label displacement, in points.
`set_axis_label_rotation`(axis_label_rotation)	Set the colorbar label rotation.
`set_axis_label_text`(axis_label_text)	Set the colorbar label text.
`set_box`(box[, box_orientation])	Set the box within which to place the colorbar.
`set_font`([family, style, variant, stretch, ...])	Set the font of the tick labels.
`set_frame_color`(color)	Set the color of the colorbar frame, in points.
`set_frame_linewidth`(linewidth)	Set the linewidth of the colorbar frame, in points.
`set_label_properties`(*args, **kwargs)	
`set_labels`(labels)	Set whether to show numerical labels.
`set_location`(location)	Set the location of the colorbar.
`set_pad`(pad)	Set the spacing between the colorbar and the image relative to the canvas size.
`set_ticks`(ticks)	Set the position of the ticks on the colorbar.
`set_width`(width)	Set the width of the colorbar relative to the canvas size.
`show`([location, width, pad, ticks, labels, ...])	Show a colorbar on the side of the image.
`update`()	

Methods Documentation

`hide`()

`set_axis_label_font`(*family=None, style=None, variant=None, stretch=None, weight=None, size=None, fontproperties=None*)

Set the font of the tick labels.

Parameters

family : str, optional

The family of the font to use. This can either be a generic font family name, either 'serif', 'sans-serif', 'cursive', 'fantasy', or 'monospace', or a list of font names in decreasing order of priority.

style : str, optional

The font style. This can be 'normal', 'italic' or 'oblique'.

variant : str, optional

The font variant. This can be 'normal' or 'small-caps'

stretch : str or int or float, optional

The stretching (spacing between letters) for the font. This can either be a numeric value in the range 0-1000 or one of 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded' or 'ultra-expanded'.

weight : str or int or float, optional

The weight (or boldness) of the font. This can either be a numeric value in the range 0-1000 or one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

size : str or int or float, optional

The size of the font. This can either be a numeric value (e.g. 12), giving the size in points, or one of 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', or 'xx-large'.

Notes

Default values are set by matplotlib or previously set values if set_font has already been called. Global default values can be set by editing the matplotlibrc file.

`set_axis_label_pad`(*axis_label_pad*)

Set the colorbar label displacement, in points.

`set_axis_label_rotation`(*axis_label_rotation*)

Set the colorbar label rotation.

`set_axis_label_text`(*axis_label_text*)

Set the colorbar label text.

`set_box`(*box, box_orientation='vertical'*)

Set the box within which to place the colorbar.

This should be in the form [xmin, ymin, dx, dy] and be in relative figure units. The orientation of the colorbar within the box can be controlled with the box_orientation argument.

`set_font`(*family=None, style=None, variant=None, stretch=None, weight=None, size=None, fontproperties=None*)

Set the font of the tick labels.

Parameters

family : str, optional

The family of the font to use. This can either be a generic font family name, either 'serif', 'sans-serif', 'cursive', 'fantasy', or 'monospace', or a list of font names in decreasing order of priority.

style : str, optional

The font style. This can be 'normal', 'italic' or 'oblique'.

variant : str, optional

The font variant. This can be 'normal' or 'small-caps'

stretch : str or int or float, optional

The stretching (spacing between letters) for the font. This can either be a numeric value in the range 0-1000 or one of 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded' or 'ultra-expanded'.

weight : str or int or float, optional

The weight (or boldness) of the font. This can either be a numeric value in the range 0-1000 or one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

size : str or int or float, optional

The size of the font. This can either be a numeric value (e.g. 12), giving the size in points, or one of 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', or 'xx-large'.

Notes

Default values are set by matplotlib or previously set values if set_font has already been called. Global default values can be set by editing the matplotlibrc file.

`set_frame_color(`*color*`)`
Set the color of the colorbar frame, in points.

`set_frame_linewidth(`*linewidth*`)`
Set the linewidth of the colorbar frame, in points.

`set_label_properties(`**args*, ***kwargs*`)`

`set_labels(`*labels*`)`
Set whether to show numerical labels.

`set_location(`*location*`)`
Set the location of the colorbar.

Should be one of 'left', 'right', 'top', 'bottom'.

`set_pad(`*pad*`)`
Set the spacing between the colorbar and the image relative to the canvas size.

`set_ticks(`*ticks*`)`
Set the position of the ticks on the colorbar.

`set_width(width)`
Set the width of the colorbar relative to the canvas size.

`show(location='right', width=0.2, pad=0.05, ticks=None, labels=True, box=None, box_orientation='vertical', axis_label_text=None, axis_label_rotation=None, axis_label_pad=5)`
Show a colorbar on the side of the image.

Parameters

location : str, optional

Where to place the colorbar. Should be one of 'left', 'right', 'top', 'bottom'.

width : float, optional

The width of the colorbar relative to the canvas size.

pad : float, optional

The spacing between the colorbar and the image relative to the canvas size.

ticks : list, optional

The position of the ticks on the colorbar.

labels : bool, optional

Whether to show numerical labels.

box : list, optional

A custom box within which to place the colorbar. This should be in the form [xmin, ymin, dx, dy] and be in relative figure units. This overrides the location argument.

box_orientation str, optional :

The orientation of the colorbar within the box. Can be 'horizontal' or 'vertical'

axis_label_text str, optional :

Optional text label of the colorbar.

`update()`

7.2.4 FITSFigure

class `aplpy.aplpy.FITSFigure(data, hdu=0, figure=None, subplot=(1, 1, 1), downsample=False, north=False, convention=None, dimensions=[0, 1], slices=[], auto_refresh=True, **kwargs)`
Bases: `aplpy.layers.Layers`, `aplpy.regions.Regions`, `aplpy.deprecated.Deprecated`

A class for plotting FITS files.

Methods Summary

add_beam(*args, **kwargs)	Add a beam to the current figure.
add_colorbar(*args, **kwargs)	Add a colorbar to the current figure.
add_grid(*args, **kwargs)	Add a coordinate to the current figure.
add_label(x, y, text[, relative, color, ...])	Add a text label.
add_scalebar(length, *args, **kwargs)	Add a scalebar to the current figure.

Continued on next page

Table 7.6 – continued from previous page

close()	Close the figure and free up the memory.
hide_colorscale()	
hide_grayscale(*args, **kwargs)	
pixel2world(xp, yp)	Convert pixel to world coordinates.
recenter(x, y[, radius, width, height])	Center the image on a given position and with a given radius.
refresh([force])	Refresh the display.
remove_beam([beam_index])	Removes the beam from the current figure.
remove_colorbar()	Removes the colorbar from the current figure.
remove_grid()	Removes the grid from the current figure.
remove_scalebar()	Removes the scalebar from the current figure.
save(filename[, dpi, transparent, ...])	Save the current figure to a file.
set_auto_refresh(refresh)	Set whether the display should refresh after each method call.
set_nan_color(color)	Set the color for NaN pixels.
set_system_latex(usetex)	Set whether to use a real LaTeX installation or the built-in matplotlib LaTeX.
set_theme(theme)	Set the axes, ticks, grid, and image colors to a certain style (experimental).
set_xaxis_coord_type(coord_type)	Set the type of x coordinate.
set_yaxis_coord_type(coord_type)	Set the type of y coordinate.
show_arrows(x, y, dx, dy[, width, ...])	Overlay arrows on the current plot.
show_circles(xw, yw, radius[, layer, zorder])	Overlay circles on the current plot.
show_colorscale([vmin, vmid, vmax, pmin, ...])	Show a colorscale image of the FITS file.
show_contour(data[, hdu, layer, levels, ...])	Overlay contours on the current plot.
show_ellipses(xw, yw, width, height[, ...])	Overlay ellipses on the current plot.
show_grayscale([vmin, vmid, vmax, pmin, ...])	Show a grayscale image of the FITS file.
show_lines(line_list[, layer, zorder])	Overlay lines on the current plot.
show_markers(xw, yw[, layer])	Overlay markers on the current plot.
show_polygons(polygon_list[, layer, zorder])	Overlay polygons on the current plot.
show_rectangles(xw, yw, width, height[, ...])	Overlay rectangles on the current plot.
show_rgb([filename, interpolation, ...])	Show a 3-color image instead of the FITS file data.
world2pixel(xw, yw)	Convert world to pixel coordinates.

Methods Documentation

add_beam(*args*, ***kwargs*)

Add a beam to the current figure.

Once this method has been run, a beam attribute becomes available, and can be used to control the aspect of the beam:

```
>>> f = aplpy.FITSFigure(...)
>>> ...
>>> f.add_beam()
>>> f.beam.set_color('white')
>>> f.beam.set_hatch('+')
>>> ...
```

If more than one beam is added, the beam object becomes a list. In this case, to control the aspect of one of the beams, you will need tp specify the beam index:

```
>>> ...
>>> f.beam[2].set_hatch('/')
>>> ...
```

add_colorbar(*args*, ***kwargs*)

Add a colorbar to the current figure.

Once this method has been run, a colorbar attribute becomes available, and can be used to control the aspect of the colorbar:

```
>>> f = aplpy.FITSFigure(...)
>>> ...
>>> f.add_colorbar()
>>> f.colorbar.set_width(0.3)
>>> f.colorbar.set_location('top')
>>> ...
```

`add_grid`(**args*, ***kwargs*)

Add a coordinate to the current figure.

Once this method has been run, a grid attribute becomes available, and can be used to control the aspect of the grid:

```
>>> f = aplpy.FITSFigure(...)
>>> ...
>>> f.add_grid()
>>> f.grid.set_color('white')
>>> f.grid.set_alpha(0.5)
>>> ...
```

`add_label`(*x*, *y*, *text*, *relative=False*, *color='black'*, *family=None*, *style=None*, *variant=None*, *stretch=None*, *weight=None*, *size=None*, *fontproperties=None*, *horizontalalignment='center'*, *verticalalignment='center'*, *layer=None*, ***kwargs*)

Add a text label.

Parameters

x, y : float

Coordinates of the text label

text : str

The label

relative : str, optional

Whether the coordinates are to be interpreted as world coordinates (e.g. RA/Dec or longitude/latitude), or coordinates relative to the axes (where 0.0 is left or bottom and 1.0 is right or top).

family : str, optional

The family of the font to use. This can either be a generic font family name, either 'serif', 'sans-serif', 'cursive', 'fantasy', or 'monospace', or a list of font names in decreasing order of priority.

style : str, optional

The font style. This can be 'normal', 'italic' or 'oblique'.

variant : str, optional

The font variant. This can be 'normal' or 'small-caps'

stretch : str or int or float, optional

The stretching (spacing between letters) for the font. This can either be a numeric value in the range 0-1000 or one of 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded' or 'ultra-expanded'.

weight : str or int or float, optional

The weight (or boldness) of the font. This can either be a numeric value in the range 0-1000 or one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

size : str or int or float, optional

The size of the font. This can either be a numeric value (e.g. 12), giving the size in points, or one of 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', or 'xx-large'.

`add_scalebar`(*length*, **args*, ***kwargs*)

Add a scalebar to the current figure.

Once this method has been run, a scalebar attribute becomes available, and can be used to control the aspect of the scalebar:

```
>>> f = aplpy.FITSFigure(...)
>>> ...
>>> f.add_scalebar(0.01) # length has to be specified
>>> f.scalebar.set_label('100 AU')
>>> ...
```

`close`()

Close the figure and free up the memory.

`hide_colorscale`()

`hide_grayscale`(**args*, ***kwargs*)

`pixel2world`(*xp*, *yp*)

Convert pixel to world coordinates.

Parameters

xp : float or list or `ndarray`

x pixel coordinate

yp : float or list or `ndarray`

y pixel coordinate

Returns

xw : float or list or `ndarray`

x world coordinate

yw : float or list or `ndarray`

y world coordinate

`recenter`(*x*, *y*, *radius=None*, *width=None*, *height=None*)

Center the image on a given position and with a given radius.

Either the radius or width/heigh arguments should be specified. The units of the radius or width/height should be the same as the world coordinates in the WCS. For images of the sky, this is often (but not always) degrees.

Parameters

x, y : float

Coordinates to center on

radius : float, optional

Radius of the region to view. This produces a square plot.

width : float, optional

Width of the region to view. This should be given in conjunction with the height argument.

height : float, optional

Height of the region to view. This should be given in conjunction with the width argument.

`refresh`(*force=True*)

Refresh the display.

Parameters

force : str, optional

If set to False, refresh() will only have an effect if auto refresh is on. If set to True, the display will be refreshed whatever the auto refresh setting is set to. The default is True.

`remove_beam`(*beam_index=None*)

Removes the beam from the current figure.

If more than one beam is present, the index of the beam should be specified using beam_index=

`remove_colorbar()`

Removes the colorbar from the current figure.

`remove_grid()`

Removes the grid from the current figure.

`remove_scalebar()`

Removes the scalebar from the current figure.

`save`(*filename*, *dpi=None*, *transparent=False*, *adjust_bbox=True*, *max_dpi=300*, *format=None*)

Save the current figure to a file.

Parameters

filename : str or fileobj

The name of the file to save the plot to. This can be for example a PS, EPS, PDF, PNG, JPEG, or SVG file. Note that it is also possible to pass file-like object.

dpi : float, optional

The output resolution, in dots per inch. If the output file is a vector graphics format (such as PS, EPS, PDF or SVG) only the image itself will be rasterized. If the output is a PS or EPS file and no dpi is specified, the dpi is automatically calculated to match the resolution of the image. If this value is larger than max_dpi, then dpi is set to max_dpi.

transparent : str, optional

Whether to preserve transparency

adjust_bbox : str, optional

Auto-adjust the bounding box for the output

max_dpi : float, optional

The maximum resolution to output images at. If no maximum is wanted, enter None or 0.

format : str, optional

By default, APLpy tries to guess the file format based on the file extension, but the format can also be specified explicitly. Should be one of 'eps', 'ps', 'pdf', 'svg', 'png'.

`set_auto_refresh`(*refresh*)

Set whether the display should refresh after each method call.

Parameters

refresh : str

Whether to refresh the display every time a FITSFigure method is called. The default is True. If set to false, the display can be refreshed manually using the refresh() method

`set_nan_color`(*color*)

Set the color for NaN pixels.

Parameters

color : str

This can be any valid matplotlib color

`set_system_latex`(*usetex*)

Set whether to use a real LaTeX installation or the built-in matplotlib LaTeX.

Parameters

usetex : str

Whether to use a real LaTex installation (True) or the built-in matplotlib LaTeX (False). Note that if the former is chosen, an installation of LaTex is required.

`set_theme`(*theme*)

Set the axes, ticks, grid, and image colors to a certain style (experimental).

Parameters

theme : str

The theme to use. At the moment, this can be 'pretty' (for viewing on-screen) and 'publication' (which makes the ticks and grid black, and displays the image in inverted grayscale)

`set_xaxis_coord_type`(*coord_type*)

Set the type of x coordinate.

Options are:

- `scalar`: treat the values are normal decimal scalar values
- `longitude`: treat the values as a longitude in the 0 to 360 range
- `latitude`: treat the values as a latitude in the -90 to 90 range

`set_yaxis_coord_type`(*coord_type*)

Set the type of y coordinate.

Options are:

- `scalar`: treat the values are normal decimal scalar values
- `longitude`: treat the values as a longitude in the 0 to 360 range
- `latitude`: treat the values as a latitude in the -90 to 90 range

`show_arrows`(*x*, *y*, *dx*, *dy*, *width='auto'*, *head_width='auto'*, *head_length='auto'*, *length_includes_head=True*, *layer=False*, *zorder=None*, ***kwargs*)

Overlay arrows on the current plot.

Parameters

x, y, dx, dy : float or list or `ndarray`

Origin and displacement of the arrows in world coordinates. These can either be scalars to plot a single arrow, or lists or arrays to plot multiple arrows.

width : float, optional

The width of the arrow body, in pixels (default: 2% of the arrow length)

head_width : float, optional

The width of the arrow head, in pixels (default: 5% of the arrow length)

head_length : float, optional

The length of the arrow head, in pixels (default: 5% of the arrow length)

length_includes_head : bool, optional

Whether the head includes the length

layer : str, optional

The name of the arrow(s) layer. This is useful for giving custom names to layers (instead of line_set_n) and for replacing existing layers.

kwargs :

Additional keyword arguments (such as facecolor, edgecolor, alpha, or linewidth) are passed to Matplotlib `PatchCollection` class, and can be used to control the appearance of the arrows.

`show_circles`(*xw*, *yw*, *radius*, *layer=False*, *zorder=None*, ***kwargs*)

Overlay circles on the current plot.

Parameters

xw : list or `ndarray`

The x postions of the circles (in world coordinates)

yw : list or `ndarray`

The y positions of the circles (in world coordinates)

radius : int or float or list or `ndarray`

The radii of the circles (in world coordinates)

layer : str, optional

The name of the circle layer. This is useful for giving custom names to layers (instead of circle_set_n) and for replacing existing layers.

kwargs :

Additional keyword arguments (such as facecolor, edgecolor, alpha, or linewidth) are passed to Matplotlib `PatchCollection` class, and can be used to control the appearance of the circles.

`show_colorscale`(*vmin=None*, *vmid=None*, *vmax=None*, *pmin=0.25*, *pmax=99.75*, *stretch='linear'*, *exponent=2*, *cmap='default'*, *smooth=None*, *kernel='gauss'*, *aspect='equal'*, *interpolation='nearest'*)

Show a colorscale image of the FITS file.

Parameters

vmin : None or float, optional

Minimum pixel value to use for the colorscale. If set to None, the minimum pixel value is determined using pmin (default).

vmax : None or float, optional

Maximum pixel value to use for the colorscale. If set to None, the maximum pixel value is determined using pmax (default).

pmin : float, optional

Percentile value used to determine the minimum pixel value to use for the colorscale if vmin is set to None. The default value is 0.25%.

pmax : float, optional

Percentile value used to determine the maximum pixel value to use for the colorscale if vmax is set to None. The default value is 99.75%.

stretch : { 'linear', 'log', 'sqrt', 'arcsinh', 'power' }, optional

The stretch function to use

vmid : None or float, optional

Baseline value used for the log and arcsinh stretches. If set to None, this is set to zero for log stretches and to vmin - (vmax - vmin) / 30. for arcsinh stretches

exponent : float, optional

If stretch is set to 'power', this is the exponent to use

cmap : str, optional

The name of the colormap to use

smooth : int or tuple, optional

Default smoothing scale is 3 pixels across. User can define whether they want an NxN kernel (integer), or NxM kernel (tuple). This argument corresponds to the 'gauss' and 'box' smoothing kernels.

kernel : { 'gauss', 'box', numpy.array }, optional

Default kernel used for smoothing is 'gauss'. The user can specify if they would prefer 'gauss', 'box', or a custom kernel. All kernels are normalized to ensure flux retention.

aspect : { 'auto', 'equal' }, optional

Whether to change the aspect ratio of the image to match that of the axes ('auto') or to change the aspect ratio of the axes to match that of the data ('equal'; default)

interpolation : str, optional

The type of interpolation to use for the image. The default is 'nearest'. Other options include 'none' (no interpolation, meaning that if exported to a postscript file, the colorscale will be output at native resolution irrespective of the dpi setting), 'bilinear', 'bicubic', and many more (see the matplotlib documentation for imshow).

`show_contour`(*data, hdu=0, layer=None, levels=5, filled=False, cmap=None, colors=None, returnlevels=False, convention=None, dimensions=[0, 1], slices=[], smooth=None, kernel='gauss', overlap=False, **kwargs*)

Overlay contours on the current plot.

Parameters

data : see below

The FITS file to plot contours for. The following data types can be passed:

string astropy.io.fits.PrimaryHDU astropy.io.fits.ImageHDU pyfits.PrimaryHDU pyfits.ImageHDU astropy.wcs.WCS np.ndarray

hdu : int, optional

By default, the image in the primary HDU is read in. If a different HDU is required, use this argument.

layer : str, optional

The name of the contour layer. This is useful for giving custom names to layers (instead of contour_set_n) and for replacing existing layers.

levels : int or list, optional

This can either be the number of contour levels to compute (if an integer is provided) or the actual list of contours to show (if a list of floats is provided)

filled : str, optional

Whether to show filled or line contours

cmap : str, optional

The colormap to use for the contours

colors : str or tuple, optional

If a single string is provided, all contour levels will be shown in this color. If a tuple of strings is provided, each contour will be colored according to the corresponding tuple element.

returnlevels : str, optional

Whether to return the list of contours to the caller.

convention : str, optional

This is used in cases where a FITS header can be interpreted in multiple ways. For example, for files with a -CAR projection and CRVAL2=0, this can be set to 'wells' or 'calabretta' to choose the appropriate convention.

dimensions : tuple or list, optional

The index of the axes to use if the data has more than three dimensions.

slices : tuple or list, optional

If a FITS file with more than two dimensions is specified, then these are the slices to extract. If all extra dimensions only have size 1, then this is not required.

smooth : int or tuple, optional

Default smoothing scale is 3 pixels across. User can define whether they want an NxN kernel (integer), or NxM kernel (tuple). This argument corresponds to the 'gauss' and 'box' smoothing kernels.

kernel : { 'gauss' , 'box' , numpy.array }, optional

Default kernel used for smoothing is 'gauss'. The user can specify if they would prefer 'gauss', 'box', or a custom kernel. All kernels are normalized to ensure flux retention.

overlap str, optional :

Whether to include only contours that overlap with the image area. This significantly speeds up the drawing of contours and reduces file size when using a file for the contours covering a much larger area than the image.

kwargs :

Additional keyword arguments (such as alpha, linewidths, or linestyles) will be passed on directly to Matplotlib's `contour()` or `contourf()` methods. For more information on these additional arguments, see the *Optional keyword arguments* sections in the documentation for those methods.

`show_ellipses`(*xw*, *yw*, *width*, *height*, *angle=0*, *layer=False*, *zorder=None*, ***kwargs*)

Overlay ellipses on the current plot.

Parameters

xw : list or `ndarray`

The x postions of the ellipses (in world coordinates)

yw : list or `ndarray`

The y positions of the ellipses (in world coordinates)

width : int or float or list or `ndarray`

The width of the ellipse (in world coordinates)

height : int or float or list or `ndarray`

The height of the ellipse (in world coordinates)

angle : int or float or list or `ndarray`, optional

rotation in degrees (anti-clockwise). Default angle is 0.0.

layer : str, optional

The name of the ellipse layer. This is useful for giving custom names to layers (instead of ellipse_set_n) and for replacing existing layers.

kwargs :

Additional keyword arguments (such as facecolor, edgecolor, alpha, or linewidth) are passed to Matplotlib `PatchCollection` class, and can be used to control the appearance of the ellipses.

`show_grayscale`(*vmin=None*, *vmid=None*, *vmax=None*, *pmin=0.25*, *pmax=99.75*, *stretch='linear'*, *exponent=2*, *invert='default'*, *smooth=None*, *kernel='gauss'*, *aspect='equal'*, *interpolation='nearest'*)

Show a grayscale image of the FITS file.

Parameters

vmin : None or float, optional

Minimum pixel value to use for the grayscale. If set to None, the minimum pixel value is determined using pmin (default).

vmax : None or float, optional

Maximum pixel value to use for the grayscale. If set to None, the maximum pixel value is determined using pmax (default).

pmin : float, optional

Percentile value used to determine the minimum pixel value to use for the grayscale if vmin is set to None. The default value is 0.25%.

pmax : float, optional

Percentile value used to determine the maximum pixel value to use for the grayscale if vmax is set to None. The default value is 99.75%.

stretch : { 'linear', 'log', 'sqrt', 'arcsinh', 'power' }, optional

The stretch function to use

vmid : None or float, optional

Baseline value used for the log and arcsinh stretches. If set to None, this is set to zero for log stretches and to vmin - (vmax - vmin) / 30. for arcsinh stretches

exponent : float, optional

If stretch is set to 'power', this is the exponent to use

invert : str, optional

Whether to invert the grayscale or not. The default is False, unless set_theme is used, in which case the default depends on the theme.

smooth : int or tuple, optional

Default smoothing scale is 3 pixels across. User can define whether they want an NxN kernel (integer), or NxM kernel (tuple). This argument corresponds to the 'gauss' and 'box' smoothing kernels.

kernel : { 'gauss', 'box', numpy.array }, optional

Default kernel used for smoothing is 'gauss'. The user can specify if they would prefer 'gauss', 'box', or a custom kernel. All kernels are normalized to ensure flux retention.

aspect : { 'auto', 'equal' }, optional

Whether to change the aspect ratio of the image to match that of the axes ('auto') or to change the aspect ratio of the axes to match that of the data ('equal'; default)

interpolation : str, optional

The type of interpolation to use for the image. The default is 'nearest'. Other options include 'none' (no interpolation, meaning that if exported to a postscript file, the grayscale will be output at native resolution irrespective of the dpi setting), 'bilinear', 'bicubic', and many more (see the matplotlib documentation for imshow).

`show_lines`(*line_list*, *layer=False*, *zorder=None*, ***kwargs*)

Overlay lines on the current plot.

Parameters

line_list : list

A list of one or more 2xN numpy arrays which contain the [x, y] positions of the vertices in world coordinates.

layer : str, optional

The name of the line(s) layer. This is useful for giving custom names to layers (instead of line_set_n) and for replacing existing layers.

kwargs :

Additional keyword arguments (such as color, offsets, linestyle, or linewidth) are passed to Matplotlib `LineCollection` class, and can be used to control the appearance of the lines.

`show_markers`(*xw*, *yw*, *layer=False*, ***kwargs*)

Overlay markers on the current plot.

Parameters

xw : list or `ndarray`

The x postions of the markers (in world coordinates)

yw : list or `ndarray`

The y positions of the markers (in world coordinates)

layer : str, optional

The name of the scatter layer. This is useful for giving custom names to layers (instead of marker_set_n) and for replacing existing layers.

kwargs :

Additional keyword arguments (such as marker, facecolor, edgecolor, alpha, or linewidth) will be passed on directly to Matplotlib's `scatter()` method (in particular, have a look at the *Optional keyword arguments* in the documentation for that method).

`show_polygons`(*polygon_list*, *layer=False*, *zorder=None*, ***kwargs*)

Overlay polygons on the current plot.

Parameters

polygon_list : list or tuple

A list of one or more 2xN or Nx2 Numpy arrays which contain the [x, y] positions of the vertices in world coordinates. Note that N should be greater than 2.

layer : str, optional

The name of the circle layer. This is useful for giving custom names to layers (instead of circle_set_n) and for replacing existing layers.

kwargs :

Additional keyword arguments (such as facecolor, edgecolor, alpha, or linewidth) are passed to Matplotlib `PatchCollection` class, and can be used to control the appearance of the polygons.

`show_rectangles`(*xw*, *yw*, *width*, *height*, *layer=False*, *zorder=None*, ***kwargs*)

Overlay rectangles on the current plot.

Parameters

xw : list or `ndarray`

The x postions of the rectangles (in world coordinates)

yw : list or `ndarray`

The y positions of the rectangles (in world coordinates)

width : int or float or list or `ndarray`

The width of the rectangle (in world coordinates)

height : int or float or list or `ndarray`

The height of the rectangle (in world coordinates)

layer : str, optional

The name of the rectangle layer. This is useful for giving custom names to layers (instead of rectangle_set_n) and for replacing existing layers.

kwargs :

Additional keyword arguments (such as facecolor, edgecolor, alpha, or linewidth) are passed to Matplotlib `PatchCollection` class, and can be used to control the appearance of the rectangles.

`show_rgb`(*filename=None*, *interpolation='nearest'*, *vertical_flip=False*, *horizontal_flip=False*, *flip=False*)

Show a 3-color image instead of the FITS file data.

Parameters

filename, optional :

The 3-color image should have exactly the same dimensions as the FITS file, and will be shown with exactly the same projection. If FITSFigure was initialized with an AVM-tagged RGB image, the filename is not needed here.

vertical_flip : str, optional

Whether to vertically flip the RGB image

horizontal_flip : str, optional

Whether to horizontally flip the RGB image

`world2pixel`(*xw*, *yw*)

Convert world to pixel coordinates.

Parameters

xw : float or list or `ndarray`

x world coordinate

yw : float or list or `ndarray`

y world coordinate

Returns

xp : float or list or `ndarray`

x pixel coordinate

yp : float or list or `ndarray`

y pixel coordinate

7.2.5 Frame

class `aplpy.frame.Frame`(*parent*)

Bases: `object`

Methods Summary

`set_color`(color)	Set color of the frame.
`set_linewidth`(linewidth)	Set line width of the frame.

Methods Documentation

`set_color`(*color*)

Set color of the frame.

Parameters

color: :

The color to use for the frame.

`set_linewidth(`*linewidth*`)`
Set line width of the frame.

Parameters
linewidth: :

The linewidth to use for the frame.

7.2.6 Grid

class `aplpy.grid.Grid(`*parent*`)`
Bases: `object`

Methods Summary

hide()	
set_alpha(alpha)	Set the alpha (transparency) of the grid lines :Parameters: **alpha** : float The alpha value of the grid.
set_color(color)	Set the color of the grid lines
set_linestyle(linestyle)	
set_linewidth(linewidth)	
set_xspacing(xspacing)	Set the grid line spacing in the longitudinal direction :Parameters: **xspacing** : { float, str } The spacing i
set_yspacing(yspacing)	Set the grid line spacing in the latitudinal direction :Parameters: **yspacing** : { float, str } The spacing in
show()	

Methods Documentation

`hide()`

`set_alpha(`*alpha*`)`
Set the alpha (transparency) of the grid lines

Parameters
alpha : float

The alpha value of the grid. This should be a floating point value between 0 and 1, where 0 is completely transparent, and 1 is completely opaque.

`set_color(`*color*`)`
Set the color of the grid lines

Parameters
color : str

The color of the grid lines

`set_linestyle(`*linestyle*`)`

`set_linewidth(`*linewidth*`)`

`set_xspacing(`*xspacing*`)`
Set the grid line spacing in the longitudinal direction

Parameters
xspacing : { float, str }

The spacing in the longitudinal direction. To set the spacing to be the same as the ticks, set this to 'tick'

`set_yspacing(yspacing)`
Set the grid line spacing in the latitudinal direction

Parameters
yspacing : { float, str }

The spacing in the latitudinal direction. To set the spacing to be the same as the ticks, set this to 'tick'

`show()`

7.2.7 Scalebar

class `aplpy.overlays.Scalebar(`*parent*`)`
Bases: `object`

Methods Summary

`hide()`	Hide the scalebar.
`set(**kwargs)`	Modify the scalebar and scalebar properties.
`set_alpha(alpha)`	Set the alpha value (transparency).
`set_color(color)`	Set the label and scalebar color.
`set_corner(corner)`	Set where to place the scalebar.
`set_font([family, style, variant, stretch, ...])`	Set the font of the tick labels :Parameters: **common: family, style, variant, stretch, w**
`set_font_family(family)`	
`set_font_size(size)`	
`set_font_style(style)`	
`set_font_weight(weight)`	
`set_frame(frame)`	Set whether to display a frame around the scalebar.
`set_label(label)`	Set the label of the scale bar.
`set_length(length)`	Set the length of the scale bar.
`set_linestyle(linestyle)`	Set the linestyle of the scalebar.
`set_linewidth(linewidth)`	Set the linewidth of the scalebar, in points.
`show(length[, label, corner, frame, ...])`	Overlay a scale bar on the image.

Methods Documentation

`hide()`
Hide the scalebar.

`set(**kwargs)`
Modify the scalebar and scalebar properties.

All arguments are passed to the matplotlib Rectangle and Text classes. See the matplotlib documentation for more details. In cases where the same argument exists for the two objects, the argument is passed to both the Text and Rectangle instance.

`set_alpha`(*alpha*)
Set the alpha value (transparency).

This should be a floating point value between 0 and 1.

`set_color`(*color*)
Set the label and scalebar color.

`set_corner`(*corner*)
Set where to place the scalebar.

Acceptable values are 'left', 'right', 'top', 'bottom', 'top left', 'top right', 'bottom left' (default), and 'bottom right'.

`set_font`(*family=None*, *style=None*, *variant=None*, *stretch=None*, *weight=None*, *size=None*, *fontproperties=None*)
Set the font of the tick labels

Parameters
common: family, style, variant, stretch, weight, size, fontproperties :

Notes

Default values are set by matplotlib or previously set values if set_font has already been called. Global default values can be set by editing the matplotlibrc file.

`set_font_family`(*family*)

`set_font_size`(*size*)

`set_font_style`(*style*)

`set_font_weight`(*weight*)

`set_frame`(*frame*)
Set whether to display a frame around the scalebar.

`set_label`(*label*)
Set the label of the scale bar.

`set_length`(*length*)
Set the length of the scale bar.

`set_linestyle`(*linestyle*)
Set the linestyle of the scalebar.

Should be one of 'solid', 'dashed', 'dashdot', or 'dotted'.

`set_linewidth`(*linewidth*)
Set the linewidth of the scalebar, in points.

`show`(*length*, *label=None*, *corner='bottom right'*, *frame=False*, *borderpad=0.4*, *pad=0.5*, ***kwargs*)
Overlay a scale bar on the image.

Parameters
length : float

The length of the scalebar

label : str, optional

Label to place below the scalebar

corner : int, optional

Where to place the scalebar. Acceptable values are:, 'left', 'right', 'top', 'bottom', 'top left', 'top right', 'bottom left' (default), 'bottom right'

frame : str, optional

Whether to display a frame behind the scalebar (default is False)

kwargs :

Additional arguments are passed to the matplotlib Rectangle and Text classes. See the matplotlib documentation for more details. In cases where the same argument exists for the two objects, the argument is passed to both the Text and Rectangle instance.

7.2.8 TickLabels

class `aplpy.labels.TickLabels`(*parent*)

Bases: `object`

Methods Summary

`hide`()	Hide the x- and y-axis tick labels.
`hide_x`()	Hide the x-axis tick labels.
`hide_y`()	Hide the y-axis tick labels.
`set_font`([family, style, variant, stretch, ...])	Set the font of the tick labels.
`set_style`(style)	Set the format of the x-axis tick labels.
`set_xformat`(format)	Set the format of the x-axis tick labels.
`set_xposition`(position)	Set the position of the x-axis tick labels ('top' or 'bottom')
`set_yformat`(format)	Set the format of the y-axis tick labels.
`set_yposition`(position)	Set the position of the y-axis tick labels ('left' or 'right')
`show`()	Show the x- and y-axis tick labels.
`show_x`()	Show the x-axis tick labels.
`show_y`()	Show the y-axis tick labels.

Methods Documentation

`hide`()

Hide the x- and y-axis tick labels.

`hide_x`()

Hide the x-axis tick labels.

`hide_y`()

Hide the y-axis tick labels.

`set_font`(*family=None*, *style=None*, *variant=None*, *stretch=None*, *weight=None*, *size=None*, *fontproperties=None*)

Set the font of the tick labels.

Parameters

family : str, optional

The family of the font to use. This can either be a generic font family name, either 'serif', 'sans-serif', 'cursive', 'fantasy', or 'monospace', or a list of font names in decreasing order of priority.

style : str, optional

The font style. This can be 'normal', 'italic' or 'oblique'.

variant : str, optional

The font variant. This can be 'normal' or 'small-caps'

stretch : str or int or float, optional

The stretching (spacing between letters) for the font. This can either be a numeric value in the range 0-1000 or one of 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded' or 'ultra-expanded'.

weight : str or int or float, optional

The weight (or boldness) of the font. This can either be a numeric value in the range 0-1000 or one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'.

size : str or int or float, optional

The size of the font. This can either be a numeric value (e.g. 12), giving the size in points, or one of 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', or 'xx-large'.

Notes

Default values are set by matplotlib or previously set values if set_font has already been called. Global default values can be set by editing the matplotlibrc file.

`set_style`(*style*)

Set the format of the x-axis tick labels.

This can be 'colons' or 'plain':

- 'colons' uses colons as separators, for example 31:41:59.26 +27:18:28.1
- 'plain' uses letters and symbols as separators, for example 31h41m59.26s +27º18'28.1"

`set_xformat`(*format*)

Set the format of the x-axis tick labels.

If the x-axis type is `longitude` or `latitude`, then the options are:

- `ddd.ddddd` - decimal degrees, where the number of decimal places can be varied
- `hh` or `dd` - hours (or degrees)
- `hh:mm` or `dd:mm` - hours and minutes (or degrees and arcminutes)
- `hh:mm:ss` or `dd:mm:ss` - hours, minutes, and seconds (or degrees, arcminutes, and arcseconds)
- `hh:mm:ss.ss` or `dd:mm:ss.ss` - hours, minutes, and seconds (or degrees, arcminutes, and arcseconds), where the number of decimal places can be varied.

If the x-axis type is `scalar`, then the format should be a valid python string format beginning with a `%`.

If one of these arguments is not specified, the format for that axis is left unchanged.

`set_xposition`(*position*)
Set the position of the x-axis tick labels ('top' or 'bottom')

`set_yformat`(*format*)
Set the format of the y-axis tick labels.

If the y-axis type is `longitude` or `latitude`, then the options are:

- `ddd.ddddd` - decimal degrees, where the number of decimal places can be varied
- `hh` or `dd` - hours (or degrees)
- `hh:mm` or `dd:mm` - hours and minutes (or degrees and arcminutes)
- `hh:mm:ss` or `dd:mm:ss` - hours, minutes, and seconds (or degrees, arcminutes, and arcseconds)
- `hh:mm:ss.ss` or `dd:mm:ss.ss` - hours, minutes, and seconds (or degrees, arcminutes, and arcseconds), where the number of decimal places can be varied.

If the y-axis type is `scalar`, then the format should be a valid python string format beginning with a `%`.

If one of these arguments is not specified, the format for that axis is left unchanged.

`set_yposition`(*position*)
Set the position of the y-axis tick labels ('left' or 'right')

`show`()
Show the x- and y-axis tick labels.

`show_x`()
Show the x-axis tick labels.

`show_y`()
Show the y-axis tick labels.

7.2.9 Ticks

class `aplpy.ticks.Ticks`(*parent*)
Bases: `object`

Methods Summary

`hide`()	Hide the x- and y-axis ticks
`hide_x`()	Hide the x-axis ticks
`hide_y`()	Hide the y-axis ticks
`set_color`(color)	Set the color of the ticks
`set_length`(length[, minor_factor])	Set the length of the ticks (in points)
`set_linewidth`(linewidth)	Set the linewidth of the ticks (in points)
`set_minor_frequency`(frequency)	Set the number of subticks per major tick.
`set_xspacing`(spacing)	Set the x-axis tick spacing, in degrees.
`set_yspacing`(spacing)	Set the y-axis tick spacing, in degrees.
`show`()	Show the x- and y-axis ticks
`show_x`()	Show the x-axis ticks
`show_y`()	Show the y-axis ticks

Methods Documentation

`hide()`
Hide the x- and y-axis ticks

`hide_x()`
Hide the x-axis ticks

`hide_y()`
Hide the y-axis ticks

`set_color(`*color*`)`
Set the color of the ticks

`set_length(`*length*, *minor_factor=0.5*`)`
Set the length of the ticks (in points)

`set_linewidth(`*linewidth*`)`
Set the linewidth of the ticks (in points)

`set_minor_frequency(`*frequency*`)`
Set the number of subticks per major tick. Set to one to hide minor ticks.

`set_xspacing(`*spacing*`)`
Set the x-axis tick spacing, in degrees. To set the tick spacing to be automatically determined, set this to 'auto'.

`set_yspacing(`*spacing*`)`
Set the y-axis tick spacing, in degrees. To set the tick spacing to be automatically determined, set this to 'auto'.

`show()`
Show the x- and y-axis ticks

`show_x()`
Show the x-axis ticks

`show_y()`
Show the y-axis ticks

Python Module Index

a

aplpy, **??**

Python Module Index

a

`aplpy`, **??**