pyregion Documentation

Release 1.2

Jae-Joon Lee

August 18, 2016

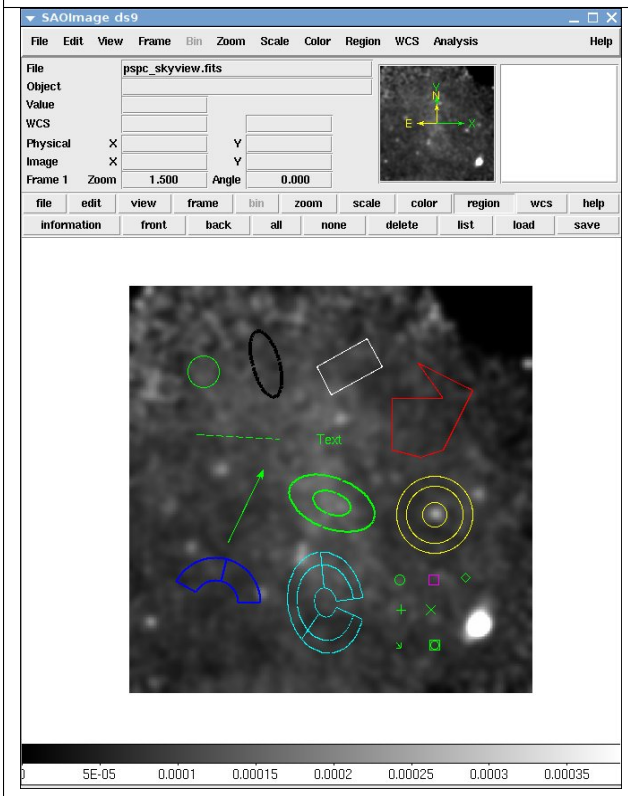
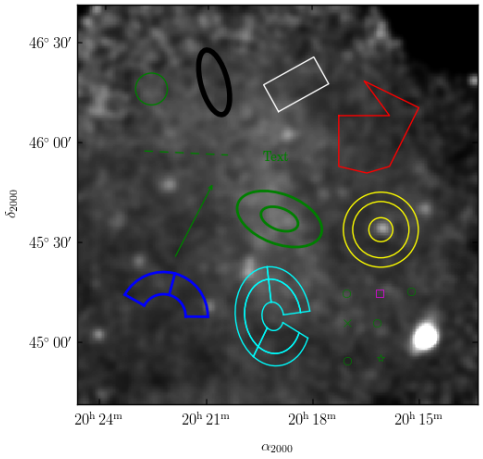
Contents

Release
1.2

Date
August 18, 2016

pyregion is a python module to parse ds9 region files. It also supports ciao region files.

Note: See also the in-development `regions` package at https://github.com/astropy/regions a new astronomy package for regions based on Astropy.

ds9	pyregion + matplotlib
	

Part I

Documentation

CHAPTER 1

Installation

1.1 Instructions

pyregion is registered in pypi, thus you can install it by

```
pip install pyregion
```

This will also install pyparsing if not installed.

The source file can be downloaded directly from the following page.

- PyPI page

To install with necessary dependency (pyparsing, see below), you may do

- in Python 2

```
  pip install "pyparsing<2"
  pip install pyregion
```

- in Python 3

```
  pip install "pyparsing>=2"
  pip install pyregion
```

The development version of pyregion can be found on the github page.

- Download

To fetch the source code by cloning my git repository for any recent bug fix.

```
git clone git://github.com/astropy/pyregion.git
cd pyregion
python setup.py install
```

For any bug reporting or any suggestion, please use the github issue tracker.

1.2 Dependencies

Requirements

Being based on pyparsing module, pyparsing need to be installed to use pyregion. Optionally, you also requires pywcs module installed for coordinate conversion. Displaying regions is supported for matplotlib. Some example uses wcsaxes.

By default, pyregion build a filtering module, which requires a C compiler. If you don't want, edit "setup.py"

```
WITH_FILTER = False
```

1.2.1 pyparsing

- REQUIRED
- Homepage
- PyPI page
- pyparsing version >= 2.0 suports Python 3. But it seems that it does not support Python 2.
- For Python 2, install older version (v1.5.7).

1.2.2 astropy

- OPTIONAL
- Astropy

1.2.3 matplotlib

- OPTIONAL
- Homepage

1.2.4 wcsaxes

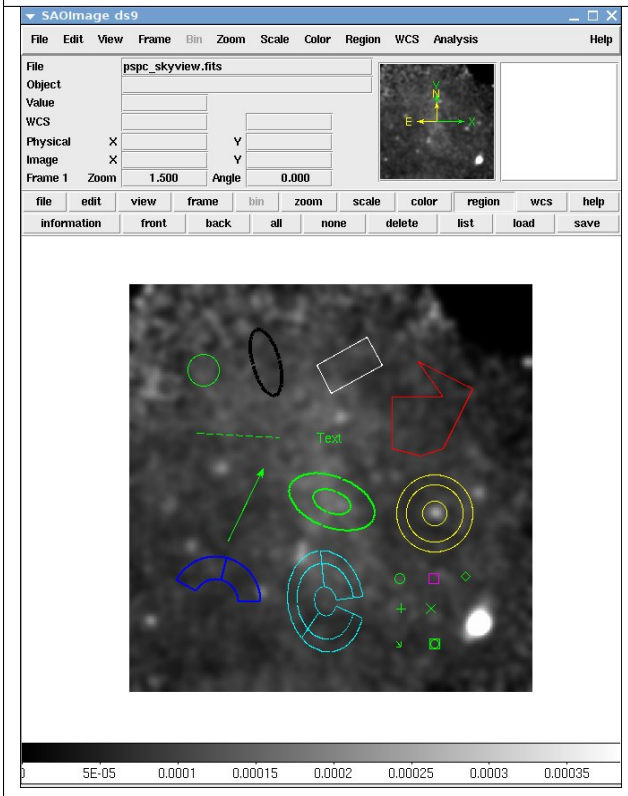
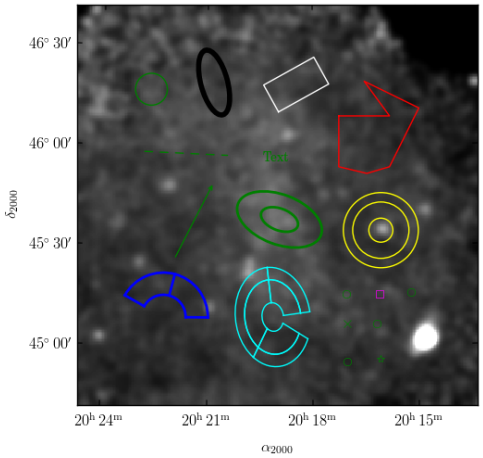
- OPTIONAL
- Homepage

CHAPTER 2

Getting started

pyregion is a python module to parse ds9 region files. It also supports ciao region files.

Please note that the main emphasis of the package is to read in the regions files generated by ds9 itself. It reads most of the region files created by ds9. However, it may fail to read some of the user-created (or created by other programs) region files, even if they can be successfully read by ds9. Ruler, Compass and Projection types are ignored.

ds9	pyregion + matplotlib
	

- *Read Region Files*
- *Draw Regions with Matplotlib*
- *Use Regions for Spatial Filtering*

2.1 Read Region Files

`pyregion.open` takes the region name as an argument and returns a `ShapeList` object, which is basically a list of `Shape` objects (`ShapeList` is a sub-class of the Python built-in `list` class).

```
import pyregion
region_name = "ds9.reg"
r = pyregion.open(region_name)
```

You may use `pyregion.parse` if you have a string that defines a region

```
region = 'fk5;circle(290.96388,14.019167,843.31194")'
r = pyregion.parse(region)
```

The shape object is a python representation of each region definition. For example,:

```
import pyregion

region_string = """
# Region file format: DS9 version 4.1
# Filename: test01.fits
global color=green dashlist=8 3 width=1 font="helvetica 10 normal" select=1 highlite=1 dash=0 fixed=0 edit=1 move=1 delet
fk5
circle(11:24:24.230,-59:15:02.20,18.5108") # color=cyan background
box(11:24:39.213,-59:16:53.91,42.804",23.616",19.0384) # width=4
"""

r = pyregion.parse(region_string)
```

And you have:

```
>>> print r[0]
Shape : circle ( HMS(11:24:24.230),DMS(-59:15:02.20),Ang(18.5108") )
>>> print r[1]
Shape : box ( HMS(11:24:39.213),DMS(-59:16:53.91),Ang(42.804"),Ang(23.616"),Number(19.0384) )
```

The shape object has the following attributes,

- `name` : name of the shape. e.g., circle, box, etc..

```
>>> print r[0].name
circle
```

- `coord_format` : coordinate format. e.g., "fk5", "image", "physical", etc...

```
>>> print r[0].coord_format
fk5
```

- `coord_list` : list of coordinates in *coord_format*. The coordinate value for sky coordinates is degree.

```
>>> print r[0].coord_list
[171.10095833333332, -59.250611111111112, 0.0051418888888888886]
```

- `comment` : comment string associated with the shape (can be None)

```
>>> print r[0].comment
color=cyan background
```

- `attr` : attributes of the shape. This includes global attributes defined by the global command and local attributes defined in the comment. The first item is a list of key-only attributes without associated values (e.g., background..) and the second item is a dictionary of attributes of key-value pairs.

```
>>> print r[0].attr[0]
['background']
>>> print r[0].attr[1]
{'color': 'cyan',
 'dash': '0 ',
 'dashlist': '8 3 ',
 'delete': '1 ',
 'edit': '1 ',
 'fixed': '0 ',
 'font': '"helvetica 10 normal"',
 'highlite': '1 ',
 'include': '1 ',
 'move': '1 ',
 'select': '1 ',
 'source': '1',
 'width': '1 '}
```

Some attributes like "tag" allow multiple items, but this is not currently supported (the last definition overrides any previous ones).

The `pyregion.ShapeList` class have a few methods that could be useful. `ShapeList.as_imagecoord` returns a new `ShapeList` instance with the coordinates converted to the image coordinate system. It requires an `astropy.io.fits.Header` instance.

```
from astropy.io import fits
f = fits.open("t1.fits")
r2 = pyregion.parse(region_string).as_imagecoord(f[0].header)
```

The return value is a new `ShapeList` instance, but the coordinate is converted to image coordinates.

```
>>> print r2[0].coord_format
image

>>> print r2[0].coord_list
[482.27721401429852, 472.76641383805912, 18.811792596807045]
```

`ShapeList.as_imagecoord` can take a `astropy.wcs.WCS` object instead of a header. This is useful if your FITS file is not a simple 2D image, as you can then use only the celestial subset of the co-ordinates to parse the region:

```
from astropy.io import fits
from astropy.WCS import WCS
f = fits.open("t1.fits")
w = WCS(f[0].header)
w_im = w.celestial
r2 = pyregion.parse(region_string).as_imagecoord(w_im)
```

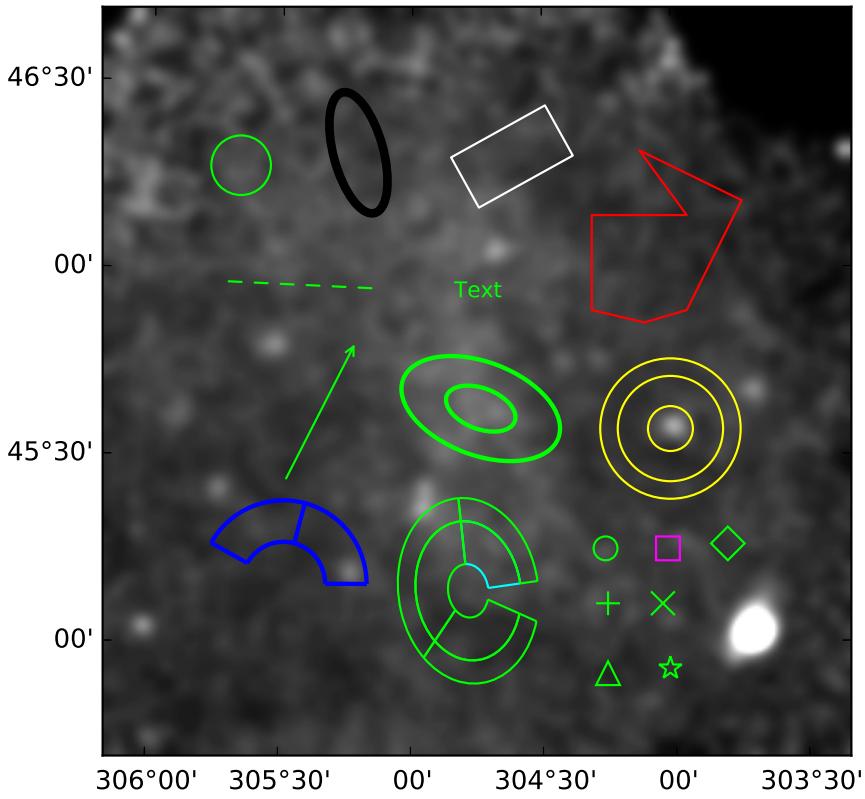
2.2 Draw Regions with Matplotlib

pyregion can help you draw ds9 regions with matplotlib. `ShapeList.get_mpl_patches_texts` returns a list of `matplotlib.artist.Artist` objects

```
r2 = pyregion.parse(region_string).as_imagecoord(f[0].header)
patch_list, artist_list = r2.get_mpl_patches_texts()
```

The first item is a list of `matplotlib.patches.Patch`, and the second one is other kinds of artists (usually Text). It is your responsibility to add these to the axes.

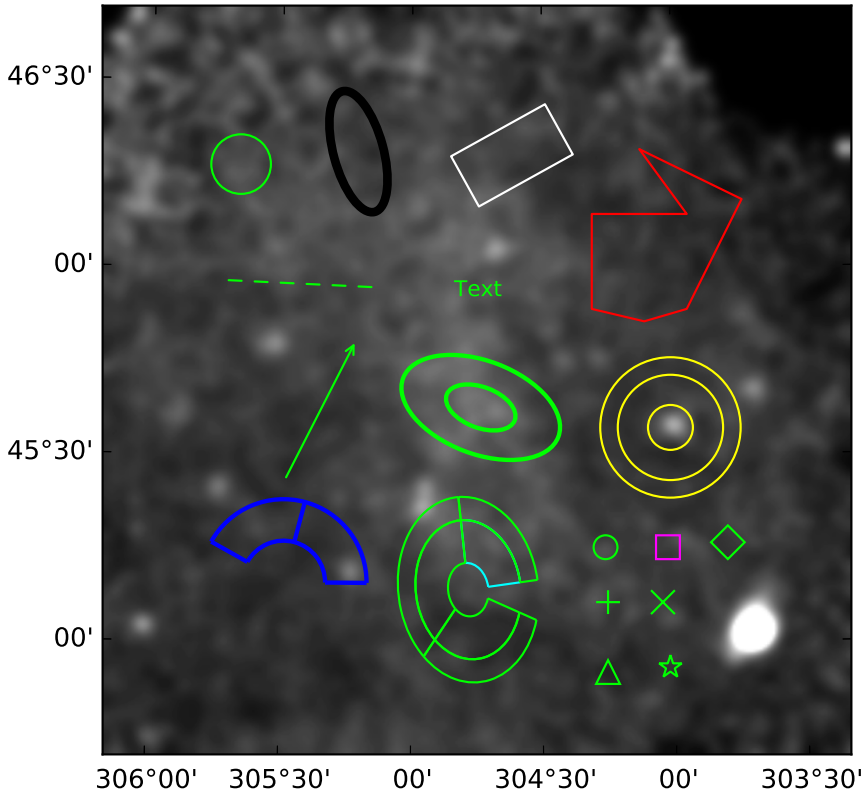
```
# ax is a mpl Axes object
for p in patch_list:
    ax.add_patch(p)
for t in artist_list:
    ax.add_artist(t)
```



The (optional) argument of the `get_mpl_patches_texts` method is a callable object that takes the shape object as an argument and returns a dictionary object that will be used as a keyword arguments (e.g., colors and line width) for creating the mpl artists. By default, it uses `pyregion.mpl_helper.properties_func_default`, which tries to respect the ds9 attributes. However, the colors (and other attributes) of some complex shapes are not correctly handled as shown in above example, and you need to manually adjust the associated attributes of patches.

```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from astropy.io import fits
import pyregion

# read in the image
xray_name = "pspc_skyview.fits"
f_xray = fits.open(xray_name)

try:
    from astropy.wcs import WCS
    from wcsaxes import WCSAxes

    wcs = WCS(f_xray[0].header)
```

```
    fig = plt.figure()
    ax = WCSAxes(fig, [0.1, 0.1, 0.8, 0.8], wcs=wcs)
    fig.add_axes(ax)
except ImportError:
    ax = plt.subplot(111)

ax.imshow(f_xray[0].data, cmap=cm.gray, vmin=0., vmax=0.00038, origin="lower")

reg_name = "test.reg"
r = pyregion.open(reg_name).as_imagecoord(header=f_xray[0].header)

from pyregion.mpl_helper import properties_func_default


# Use custom function for patch attribute
def fixed_color(shape, saved_attrs):
    attr_list, attr_dict = saved_attrs
    attr_dict["color"] = "red"
    kwargs = properties_func_default(shape, (attr_list, attr_dict))

    return kwargs


# select region shape with tag=="Group 1"
r1 = pyregion.ShapeList([rr for rr in r if rr.attr[1].get("tag") == "Group 1"])
patch_list1, artist_list1 = r1.get_mpl_patches_texts(fixed_color)

r2 = pyregion.ShapeList([rr for rr in r if rr.attr[1].get("tag") != "Group 1"])
patch_list2, artist_list2 = r2.get_mpl_patches_texts()

for p in patch_list1 + patch_list2:
    ax.add_patch(p)
for t in artist_list1 + artist_list2:
    ax.add_artist(t)

plt.show()
```



2.3 Use Regions for Spatial Filtering

pyregion includes some basic spatial filter support.

```
>>> import pyregion._region_filter as filter
>>> myfilter = filter.Circle(0, 0, 10) & filter.Box(15, 0, 10, 10)
>>> myfilter.inside1(0, 0)
0
>>> myfilter.inside1(10, 0)
1
>>> myfilter.inside([0, 10], [0, 0])
array([False,  True], dtype=bool)
```

The ShapeList.get_filter method returns the filter from the parsed region. The filter is meant to be used in the image coordinate, thus you need to convert the region to the image coordinate before calling get_filter.

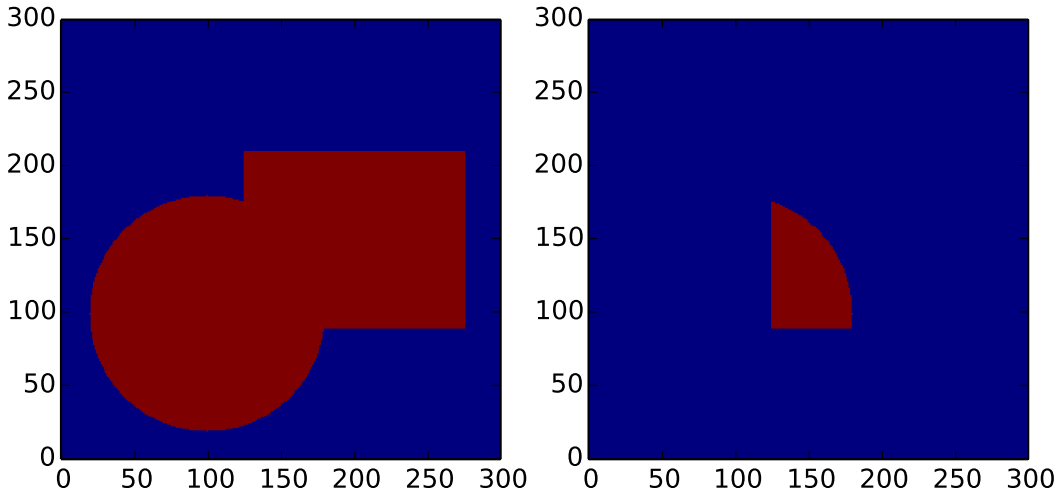
```
r2 = pyregion.parse(region_string).as_imagecoord(f[0].header)
myfilter = r2.get_filter()
myfilter.inside1(50, 30)
```

The returned filter has a mask method that creates a 2d mask. You can create the mask directly from the ShapeList object.

```
r2 = pyregion.parse(region_string)
mymask = r2.get_mask(hdu=f[0])
```

It will creates an mask in the shape of the given hdu image (the mask will be created after transforming the region to the image coordinate if necessary).

```
import matplotlib.pyplot as plt
import pyregion

region = """
image
circle(100, 100, 80)
box(200, 150, 150, 120, 0)
"""

r = pyregion.parse(region)
mask_1or2 = r.get_mask(shape=(300, 300))

myfilter = r.get_filter()
mask_1and2 = (myfilter[0] & myfilter[1]).mask((300, 300))

plt.subplot(121).imshow(mask_1or2, origin="lower", interpolation="nearest")
plt.subplot(122).imshow(mask_1and2, origin="lower", interpolation="nearest")
plt.show()
```



Note that this will fail if your template image is not a simple 2D image. To work around this you may use the shape

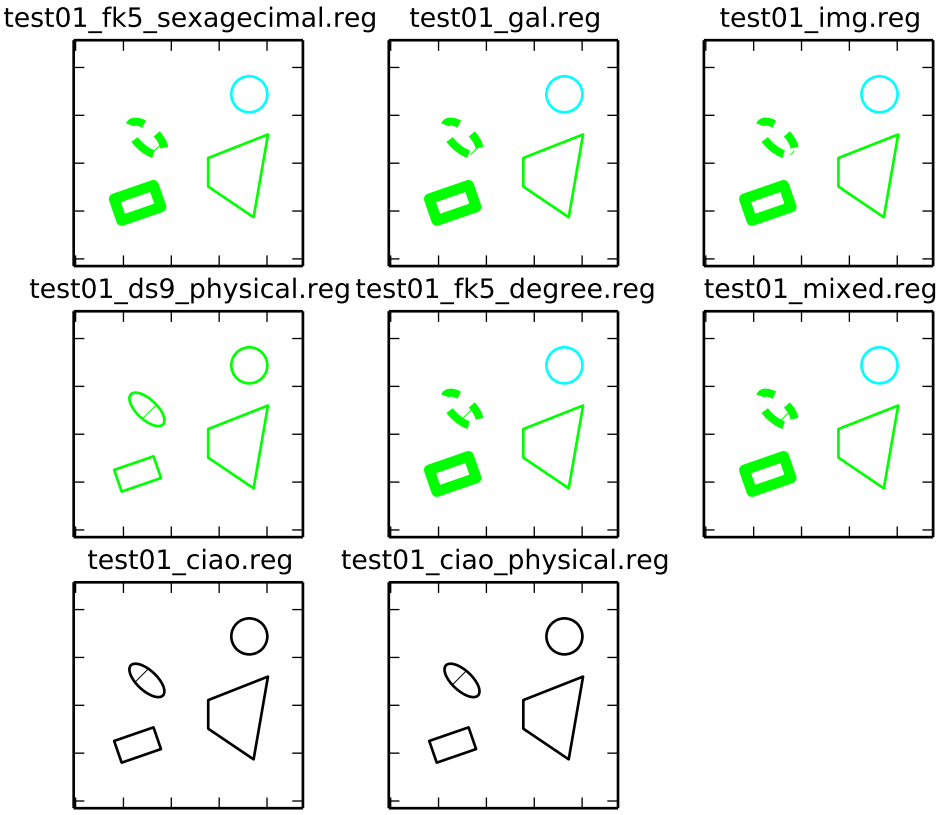
optional argument of `get_mask`:

```
mymask = r2.get_mask(hdu=f[0],shape=(1024,1024))
```

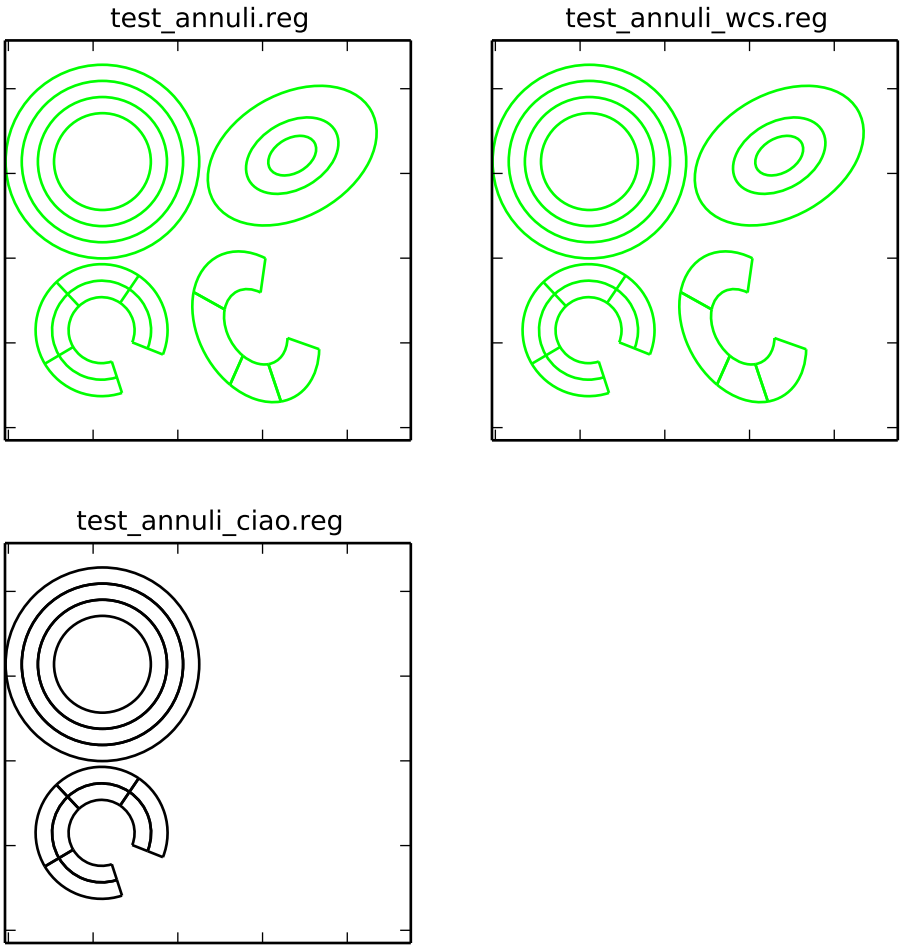
CHAPTER 3

Examples

3.1 demo_region01.py

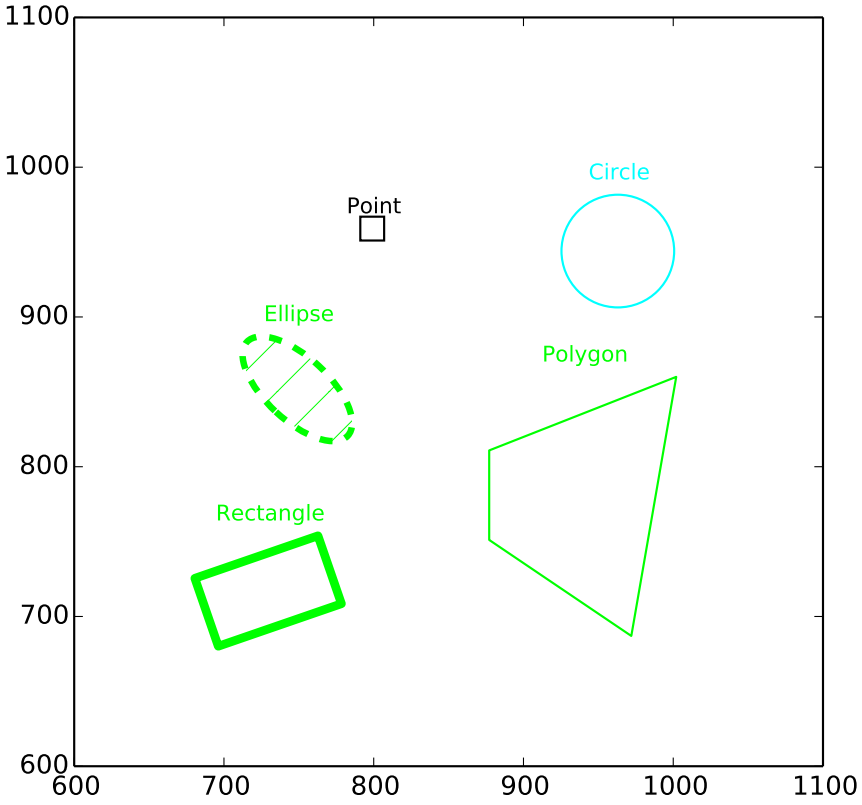


3.2 demo_region02.py

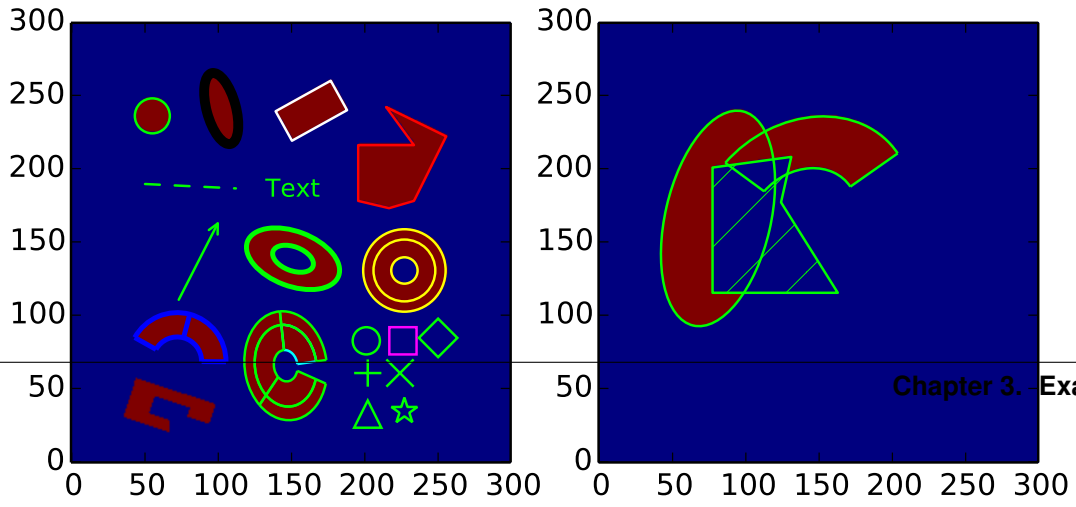


3.3 demo_region03.py

3.4 demo_region04.py



3.5 demo_region_filter01.py



CHAPTER 4

Reference/API

pyregion: a Python parser for ds9 region files

- Code : https://github.com/astropy/pyregion
- Docs : http://pyregion.readthedocs.io/

See also the in-development `regions` package at https://github.com/astropy/regions a new astronomy package for regions based on Astropy.

4.1 Functions

`get_mask`(region, hdu[, origin])	Get mask.
`open`(fname)	Open, read and parse DS9 region file.
`parse`(region_string)	Parse DS9 region string into a ShapeList.
`test`([package, test_path, args, plugins, ...])	Run the tests using py.test.

4.1.1 get_mask

pyregion.**get_mask**(*region*, *hdu*, *origin=1*)

Get mask.

Parameters

region : `ShapeList`

List of `Shape`

hdu : `ImageHDU`

FITS image HDU

origin : float

TODO: document me

Returns

mask : `array`

Boolean mask

Examples

```
>>> from astropy.io import fits
>>> from pyregion import read_region_as_imagecoord, get_mask
>>> f = fits.read("test.fits")
>>> region = "test01.reg"
>>> reg = read_region_as_imagecoord(open(region), f[0].header)
>>> mask = get_mask(reg, f[0])
```

4.1.2 open

pyregion.**open**(*fname*)

Open, read and parse DS9 region file.

Parameters

fname : str

Filename

Returns

shapes : ShapeList

List of Shape

4.1.3 parse

pyregion.**parse**(*region_string*)

Parse DS9 region string into a ShapeList.

Parameters

region_string : str

Region string

Returns

shapes : ShapeList

List of Shape

4.1.4 test

pyregion.**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 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.

4.2 Classes

Shape(shape_name, shape_params)	Shape.
ShapeList(shape_list[, comment_list])	A list of Shape objects.

4.2.1 Shape

class pyregion.**Shape**(*shape_name*, *shape_params*)

Bases: object

Shape.

Parameters

shape_name : str

Shape name

params : list

List of parameters

Examples

```
>>> region_string = 'fk5;circle(290.96388,14.019167,843.31194")'
>>> shape_list = pyregion.parse(region_string)
>>> shape = shape_list[0]
>>> print(shape.__dict__)
{'attr': ([], {}),
 'comment': None,
 'continued': None,
 'coord_format': 'fk5',
 'coord_list': [290.96388, 14.019167, 0.23425331666666666],
 'exclude': False,
 'name': 'circle',
 'params': [Number(290.96388), Number(14.019167), Ang(843.31194")]}
```

Methods Summary

set_exclude()

Methods Documentation

set_exclude()

4.2.2 ShapeList

class pyregion.**ShapeList**(*shape_list*, *comment_list=None*)

Bases: list

A list of Shape objects.

Parameters

shape_list : list

List of pyregion.Shape objects

comment_list : list, None

List of comment strings for each argument

Methods Summary

as_imagecoord(header[, rot_wrt_axis])	New shape list in image coordinates.
check_imagecoord()	Are all shapes in image coordinates?
get_filter([header, origin, rot_wrt_axis])	Get filter.
get_mask([hdu, header, shape, rot_wrt_axis])	Create a 2-d mask.
get_mpl_patches_texts([properties_func, ...])	Often, the regions files implicitly assume the lower-left corner of the image as a co
write(outfile)	Write this shape list to a region file.

Methods Documentation

as_imagecoord(*header*, *rot_wrt_axis=1*)

New shape list in image coordinates.

Parameters

header : Header

FITS header

rot_wrt_axis : {1, 2}

Use rotation with respect to axis 1 (X-axis) or axis 2 (Y-axis) and north.

Returns

shape_list : ShapeList

New shape list, with coordinates of the each shape converted to the image coordinate using the given header information.

check_imagecoord()

Are all shapes in image coordinates?

Returns `True` if yes, and `False` if not.

get_filter(*header=None*, *origin=1*, *rot_wrt_axis=1*)

Get filter.

Often, the regions files implicitly assume the lower-left corner of the image as a coordinate (1,1). However, the python convetion is that the array index starts from 0. By default (`origin=1`), coordinates of the returned mpl artists have coordinate shifted by (1, 1). If you do not want this shift, use `origin=0`.

Parameters

header : astropy.io.fits.Header

FITS header

origin : {0, 1}

Pixel coordinate origin

rot_wrt_axis : {1, 2}

Use rotation with respect to axis 1 (X-axis) or axis 2 (Y-axis) and north.

Returns

filter : TODO

Filter object

get_mask(*hdu=None*, *header=None*, *shape=None*, *rot_wrt_axis=1*)

Create a 2-d mask.

Parameters

hdu : astropy.io.fits.ImageHDU

FITS image HDU

header : `Header`

FITS header

shape : tuple

Image shape

rot_wrt_axis : {1, 2}

Use rotation with respect to axis 1 (X-axis) or axis 2 (Y-axis) and north.

Returns

mask : `numpy.array`

Boolean mask

Examples

get_mask(hdu=f[0]) get_mask(shape=(10,10)) get_mask(header=f[0].header, shape=(10,10))

get_mpl_patches_texts(*properties_func=None*, *text_offset=5.0*, *origin=1*)

Often, the regions files implicitly assume the lower-left corner of the image as a coordinate (1,1). However, the python convetion is that the array index starts from 0. By default (`origin=1`), coordinates of the returned mpl artists have coordinate shifted by (1, 1). If you do not want this shift, use `origin=0`.

write(*outfile*)

Write this shape list to a region file.

Parameters

outfile : str

File name

CHAPTER 5

Changelog

5.1 2.0 (unreleased)

We plan to release pyregion v2.0 shortly (weeks) after v1.2.

5.2 1.2 (Aug 11, 2016)

- https://pypi.org/project/pyregion/1.2/
- The changelog for this release is incomplete.
- We'll start collecting a complete changelog starting after this release.
- This release brings major changes to the code, docs and test setup, the package was converted to an Astropy affiliated package.
- There are only a few bugfixes and there should be no changes that break scripts or change results for pyregion users.

5.3 1.1.4 (Oct 26, 2014)

- https://pypi.org/project/pyregion/1.1.4/
- The changelog for this release is incomplete.
- Change tag attribute from string to list of strings. [#26]

5.4 1.1 (March 15, 2013)

- https://pypi.org/project/pyregion/1.1/
- No changelog available

5.5 1.0 (Sep 14, 2010)

- https://pypi.org/project/pyregion/1.0/

- First stable release

Note: *pyregion* is rather slow, likely due to a inefficient parser. Any contribution will be welcome.

p

A

C

G

O

P

S

T

W