pyregion Documentation

Release 2.1.dev159

Jae-Joon Lee

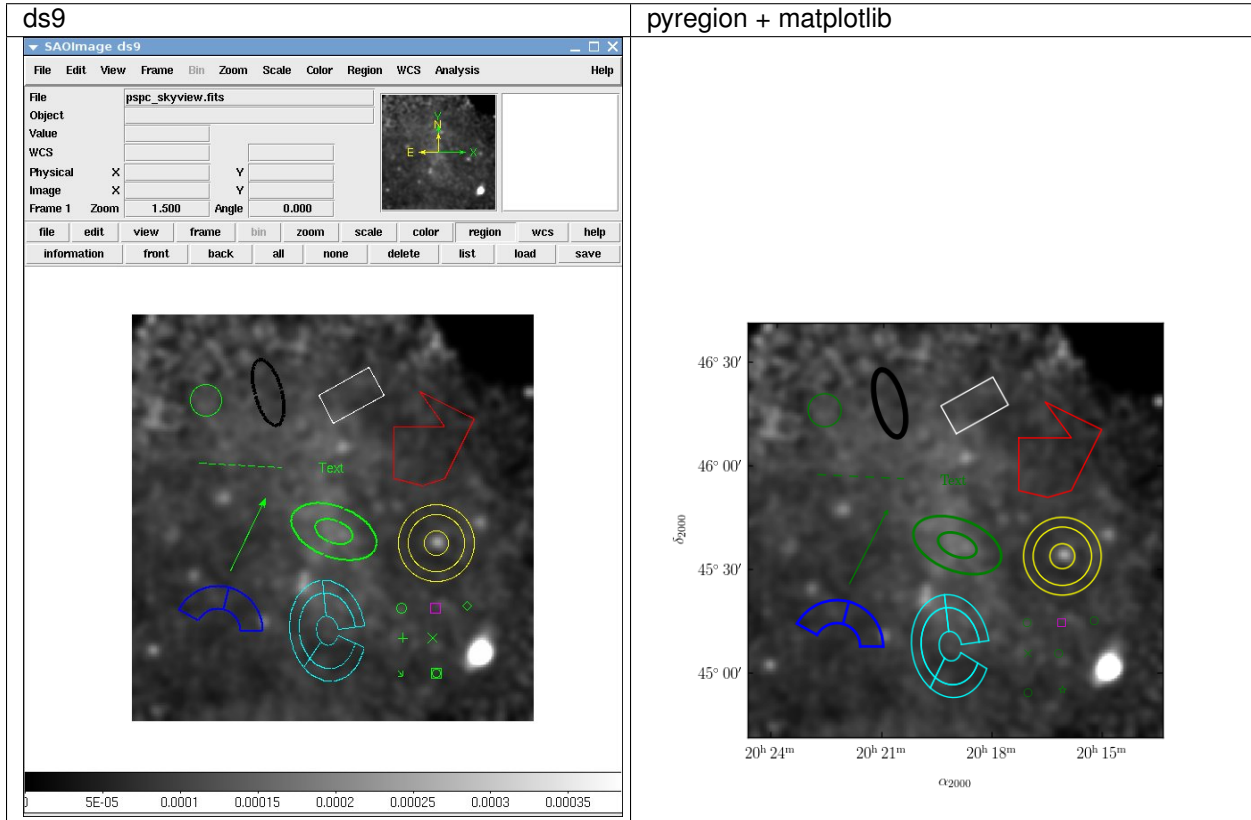**Jul 23, 2019**

Contents

Release
> 2.1.dev159

Date
> Jul 23, 2019

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.

Part I

Documentation

Installation

1.1 Stable version

Installing the latest stable version is possible either using pip or conda.

1.1.1 Using pip

To install pyregion with pip from PyPI simply run:

```
pip install --no-deps pyregion
```

Note: The `--no-deps` flag is optional, but highly recommended if you already have Numpy installed, since otherwise pip will sometimes try to "help" you by upgrading your Numpy installation, which may not always be desired.

1.1.2 Using conda

To install regions with Anaconda from the conda-forge channel on anaconda.org simply run:

```
conda install -c conda-forge pyregion
```

1.1.3 Testing installation

To check if your install is OK, run the tests:

```
python -c 'import pyregion; pyregion.test()'
```

1.2 Development version

Install the latest development version from https://github.com/astropy/pyregion :

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

1.3 Dependencies

Python 2.7 and 3.4+ are supported.

pyregion has the following required dependencies:

- Astropy version 1.0 or later (which requires Numpy)
- **pyparsing version 2.0 or later for parsing the DS9 region files**

  - Homepage
  - PyPI page

pyregion has the following optional dependencies for plotting:

- matplotlib

If you are using Astropy version 1.3 or later, then you have `astropy.visualization.wcsaxes`. For older versions of Astropy, you have to install the separate package: wcsaxes

To work with the development version, you'll need Cython and a C compiler, because the code to generate masks from regions is written in Cython.

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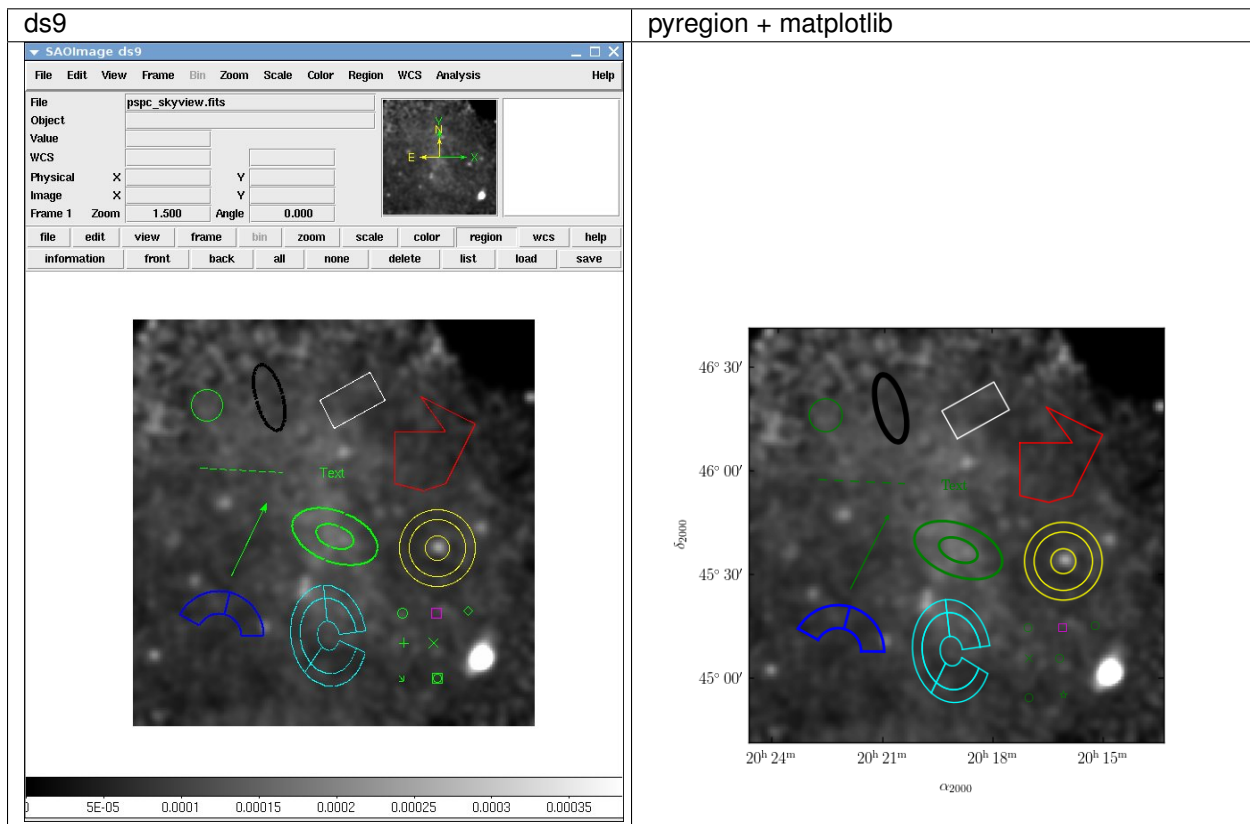
- *Read Region Files*
- *Draw Regions with Matplotlib*
- *Use Regions for Spatial Filtering*

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).

```python
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

```python
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,:

```python
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 delete=1 include=1 source=1
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:

```python
>>> 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..

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

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

  ```python
  >>> 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 will use the subset of the header defining a celestial coordinate system, ignoring any velocity or channel components.

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.

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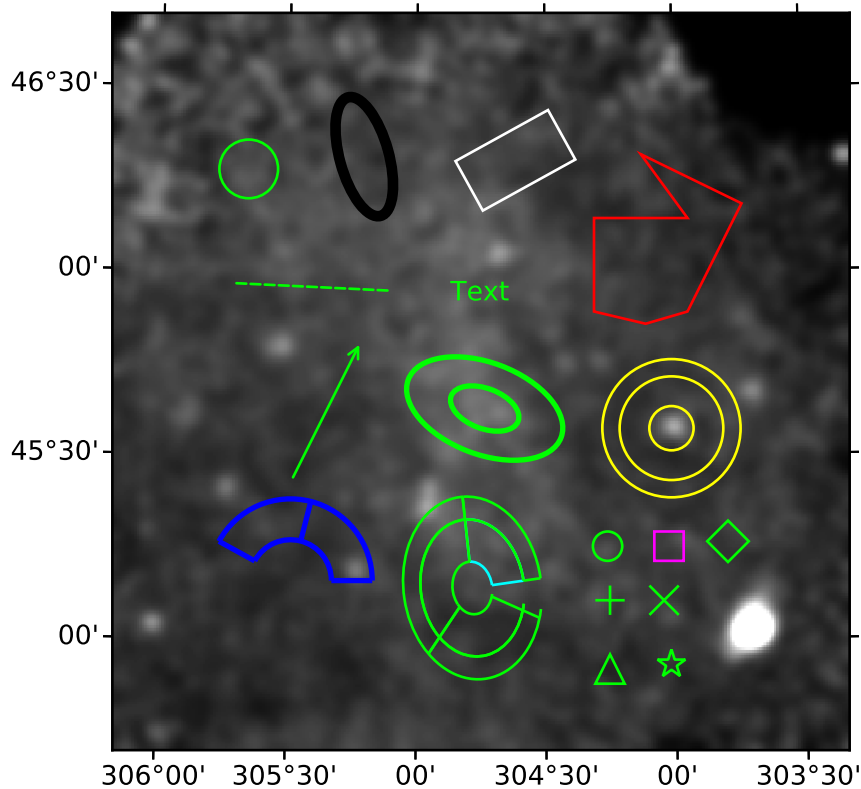

The (optional) argument of the `get_mpl_patches_texts` method is a callable object that takes the shape object as an argument and returns a dictionary object that will be used as a keyword arguments (e.g., colors and line width) for creating the mpl artists. By default, it uses `pyregion.mpl_helper.properties_func_default`, 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.

```python
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 astropy.visualization.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()
```
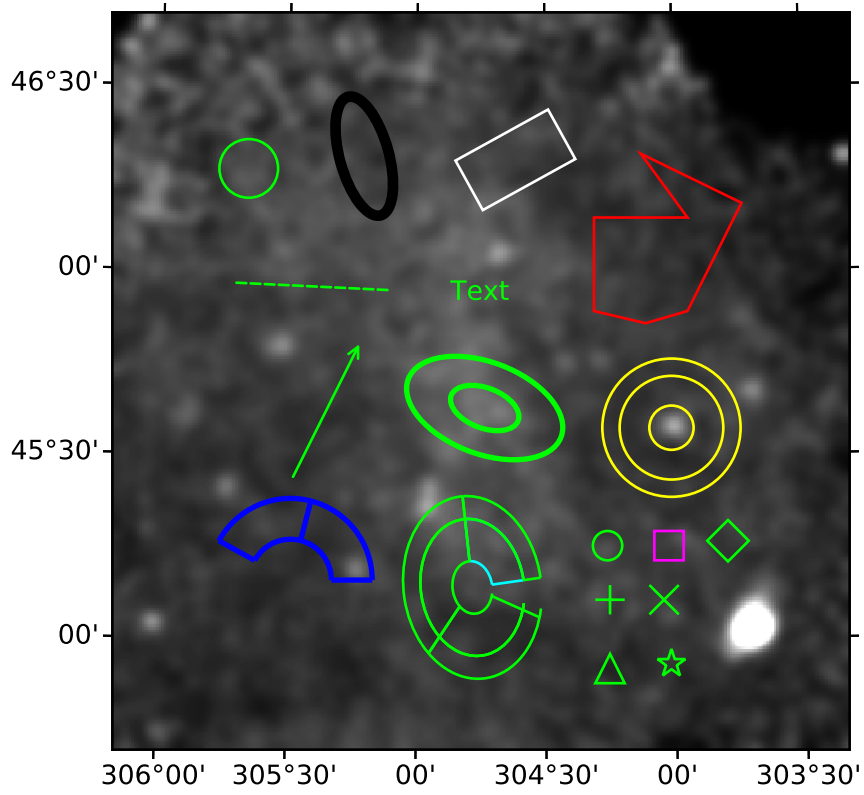
2.3 Use Regions for Spatial Filtering

pyregion includes some basic spatial filter support.

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).

```python
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()
```
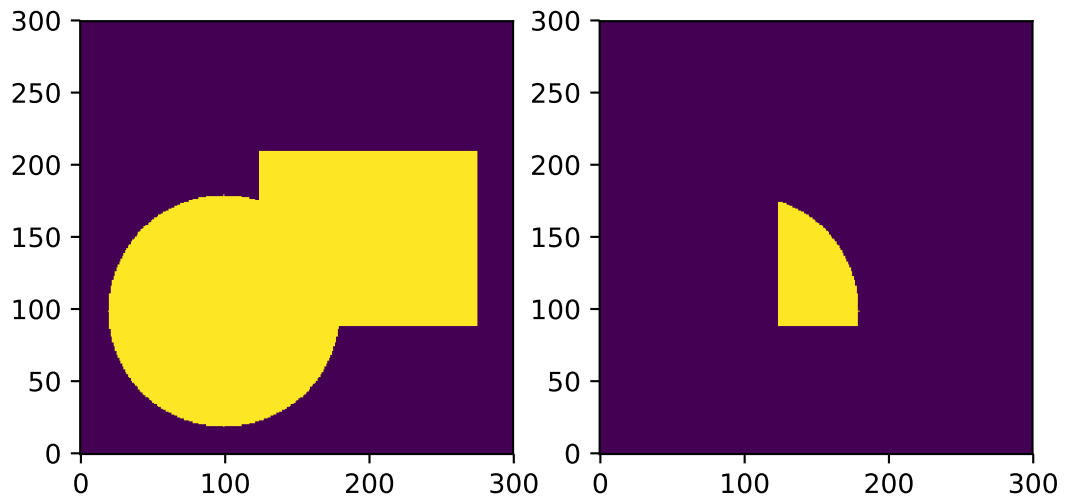
Note that this will fail if your template image is not a simple 2D image. To work around this you may use the shape optional argument of ShapeList.get_mask:
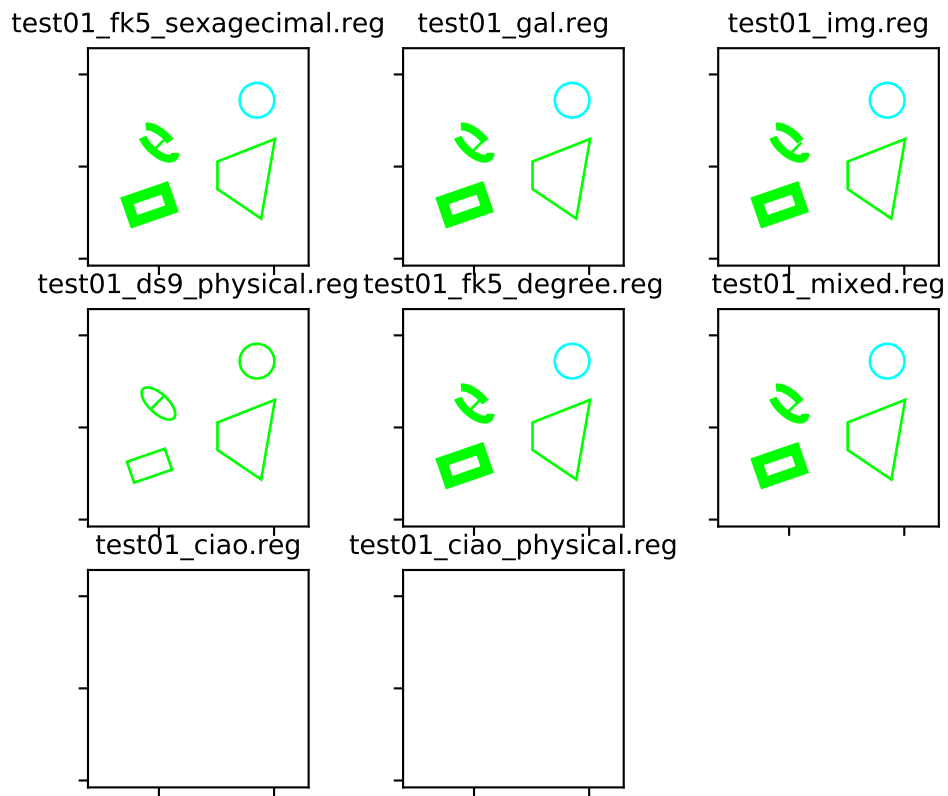
```
mymask = r2.get_mask(hdu=f[0],shape=(1024,1024))
```

CHAPTER 3

Examples

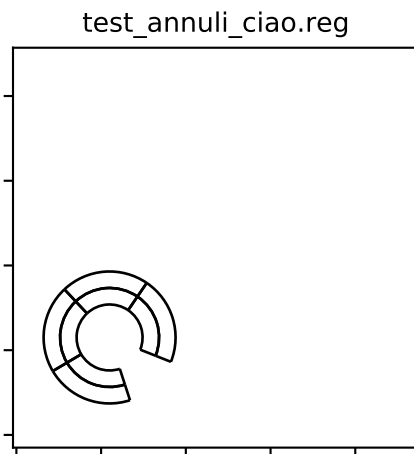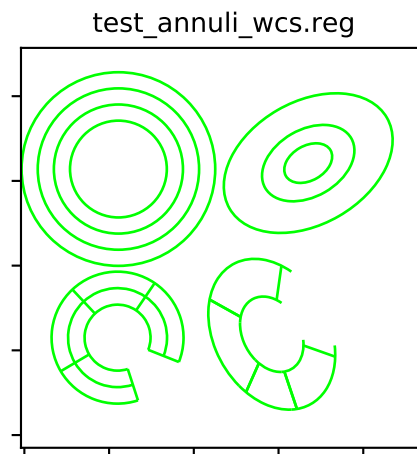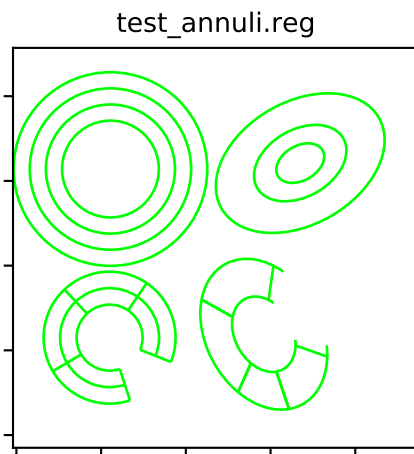## 3.1 demo_region01.py

3.2 demo_region02.py

3.3 demo_region03.py

3.4 demo_region04.py

3.5 demo_region_filter01.py

test01_fk5_sexagecimal.reg   test01_gal.reg   test01_img.reg

test01_ds9_physical.reg test01_fk5_degree.reg   test01_mixed.reg

test01_ciao.reg   test01_ciao_physical.reg

test_annuli.reg

test_annuli_wcs.reg

test_annuli_ciao.reg

Reference/API
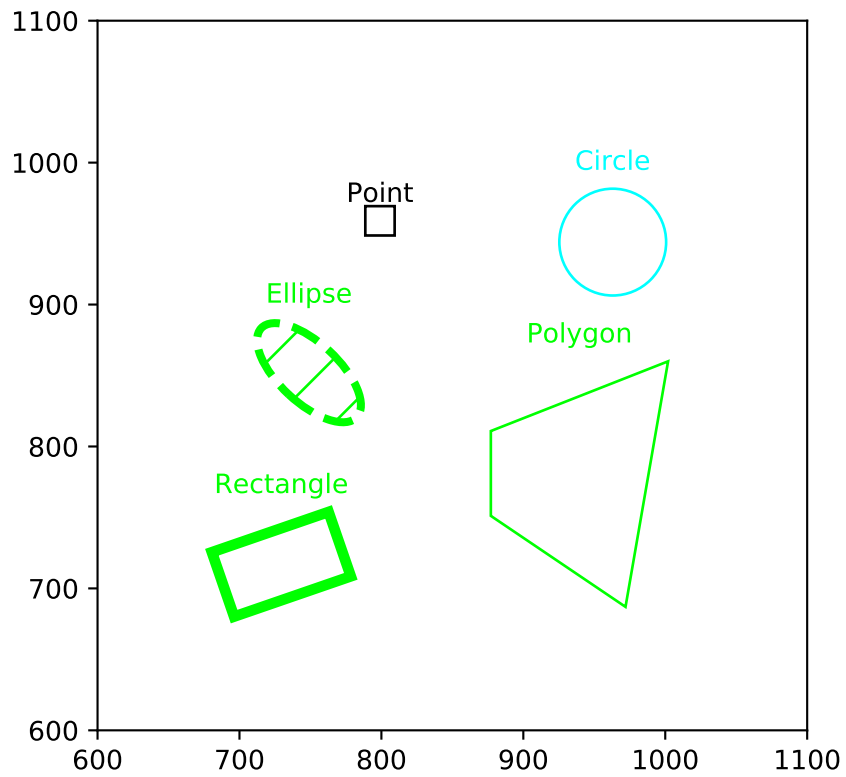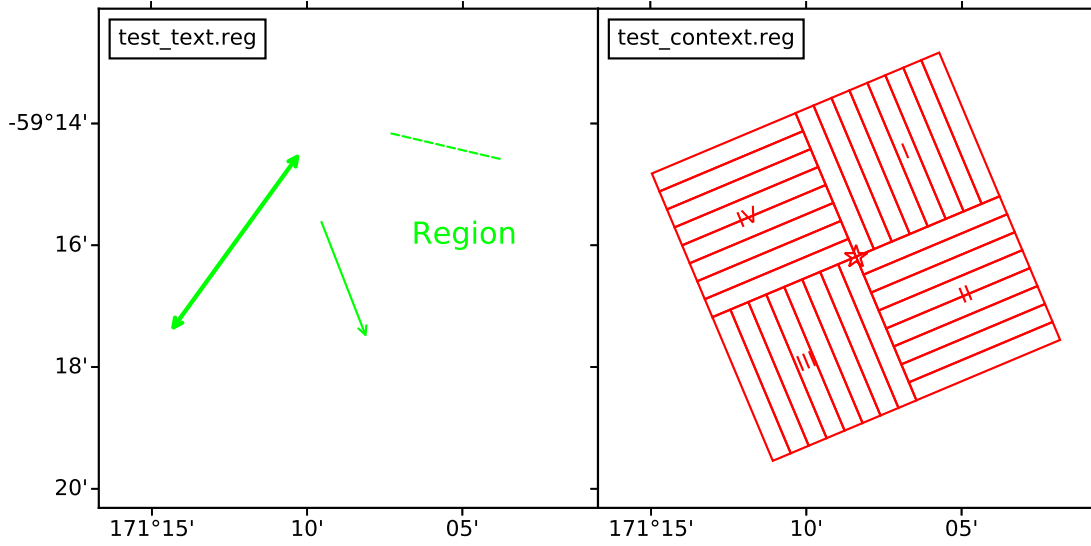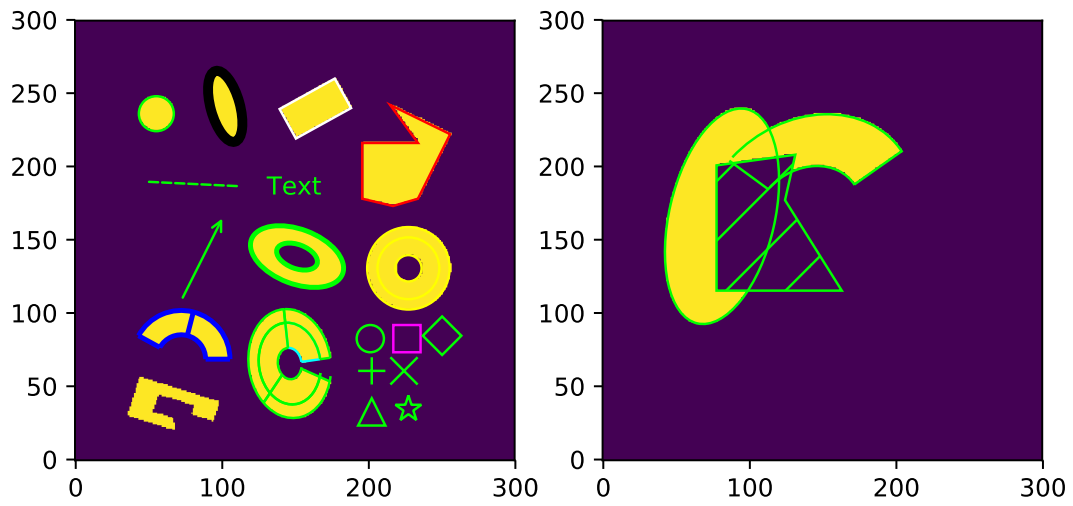
pyregion: a Python parser for ds9 region files

- 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(**kwargs)	Run the tests for the package.

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
>>> hdu = fits.open("test.fits")[0]
>>> region = "test01.reg"
>>> reg = read_region_as_imagecoord(open(region), f[0].header)
>>> mask = get_mask(reg, hdu)
```

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**(***kwargs*)

Run the tests for the package.

This method builds arguments for and then calls 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 Astropy tests are run.

args
[str, optional] Additional arguments to be passed to `pytest.main` in the `args` keyword argument.

docs_path
[str, optional] The path to the documentation .rst files.

open_files
[bool, optional] Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Requires the `psutil` package.

parallel
[int or 'auto', optional] When provided, run the tests in parallel on the specified number of CPUs. If parallel is `'auto'`, it will use the all the cores on the machine. Requires the `pytest-xdist` plugin.

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.

pdb
[bool, optional] Turn on PDB post-mortem analysis for failing tests. Same as specifying `--pdb` in `args`.

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

plugins
[list, optional] Plugins to be passed to `pytest.main` in the `plugins` keyword argument.

remote_data
[{'none', 'astropy', 'any'}, optional] Controls whether to run tests marked with @remote_data. This can be set to run no tests with remote data (none), only ones that use data from http://data.astropy.org (astropy), or all tests that use remote data (any). The default is none.

repeat
[int, optional] If set, specifies how many times each test should be run. This is useful for diagnosing sporadic failures.

skip_docs
[bool, optional] When True, skips running the doctests in the .rst files.

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.

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

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

> > > **shape_params**
> > > > [list] List of parameters

> **Examples**

```
>>> import pyregion
>>> 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)	New shape list in image coordinates.
`check_imagecoord`()	Are all shapes in image coordinates?
`get_filter`([header, origin])	Get filter.
`get_mask`([hdu, header, shape])	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 coordinate (1,1).
`write`(outfile)	Write this shape list to a region file.

Methods Documentation

as_imagecoord(*header*)

New shape list in image coordinates.

> **Parameters**
>
> > **header**
> > > [Header] FITS header
>
> **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*)

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

Returns

filter
[TODO] Filter object

get_mask(*hdu=None*, *header=None*, *shape=None*)
Create a 2-d mask.

Parameters

hdu
[astropy.io.fits.ImageHDU] FITS image HDU

header
[Header] FITS header

shape
[tuple] Image shape

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

Changelog

5.1 2.1 (unreleased)

5.1.1 Other Changes and Additions

- Support for angular units of 'd' and 'r' added.

5.2 2.0 (Oct 14, 2017)

This is a major new release of **pyregion**. There are some API changes (listed below), but overall our goal was to keep backwards-compatibility as much as possible while fixing code and installation issues and refactor the internals to use Astropy more.

We note that we are developing a new **regions** package that is supposed to become a superset of the functionality that is now in **pyregion** and might be moved in the Astropy core package as **astropy.regions** in the future. The main difference is that it represents regions as classes and uses Astropy angle and coordinate objects, allowing for easier region-based analysis. It is not feature complete, especially the DS9 region file parser is not a complete replacement for **pyregion** yet. Still, you are encouraged to try it out (http://astropy-regions.readthedocs.io/), give feedback or even contribute.

For **pyregion**, the plan is to continue to do bugfixes and releases, but to keep API changes to a minimum to avoid breaking existing scripts or pipelines. If you have any questions or issues or requests, please open an issue in the **pyregion** issue tracker on Github.

5.2.1 API Changes

- Removed `rot_wrt_axis` parameter from `ShapeList` and internal methods.

- `ShapeList.as_imagecoord` no longer accepts a `asropy.wcs.WCS` object. The conversion from pixel to image coordinates depends on the center of the image defined in `astropy.io.fits.Header` in order to agree with DS9.

- `pyregion.ds9_region_parser`

  - `RegionParser.sky_to_image` now calls its first parameter `shape_list` instead of `l`.

- `pyregion.extern`

  - `kapteyn_celestial` removed.

- `pyregion.wcs_converter`

  - `convert_to_imagecoord` changed signature with the switch to Astropy and takes a Shape object.

  - `convert_physical_to_imagecoord` changed signature to accept a Shape object.

- `pyregion.wcs_helper`

  - All public methods and constants removed. They are replaced by Astropy, or replaced by private methods.

5.2.2 Other Changes and Additions

- Astropy is used for all sky to image coordinate conversions. Science results may change, as SIP and distortion paper corrections are now used if present in the FITS file.

- Headers with more then 2 axes are now supported; only the celestial axes are used.

- Rotation angles are measured from the Y-axis instead of the X-axis, in order to agree with DS9 and potentially other astronomy software. This is a change from previous behavior, but only affects images with non-orthogonal axes. Previously, this behavior was controlled by the `rot_wrt_axis` parameter.

- Astropy 1.0 is now required.

- Shape conversion for multi-dimenstional HDU does not raise exceptions.

- Parser supports hex color in attributes

5.3 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.

5.4 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]

5.5 1.1 (March 15, 2013)

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

5.6 1.0 (Sep 14, 2010)

- https://pypi.org/project/pyregion/1.0/
- First stable release

Python Module Index

p

A

C

G

O

P

S

T

W