dendrocat Documentation

Release 0.0.dev079

B. Connor McClellan

Jan 23, 2019

Contents

Part I

Introduction

This is the documentation for `dendrocat`, a package for detecting and processing sources in radio images. `dendrocat` provides classes and methods to do the following core functions:

- Pipeline radio images to source catalogs
- Match sources between multiple catalogs
- Aperture photometry on source catalogs

Note that this package relies on other astronomy-related Python packages to function, which themselves may be in development.

- Code: Github repository
- Docs: *dendrocat*
- Contributors: https://github.com/cmcclellan1010/dendrocat/graphs/contributors

Part II

Getting Started

The procedure for creating radio source objects, generating dendrograms, and creating source catalogs is demonstrated below. This example uses default settings. In-depth documentation is located in the *Using dendrocat* section.

`RadioSource` is the starting place for any analysis to be done using dendrocat. It takes a radio image, extracts information from the FITS header, and sets attributes that are necessary for further processing.

```
>>> from dendrocat import RadioSource
>>> from astropy.io import fits
>>> source_object = RadioSource(fits.open('/path/to/file.fits'))
>>> source_object
<dendrocat.radiosource.RadioSource at 0x7fda2f851080>
```

Note: FITS header extraction is undergoing development. Currently, only specific headers from EVLA and ALMA are supported, but this will change soon. See *The RadioSource Class* for details.

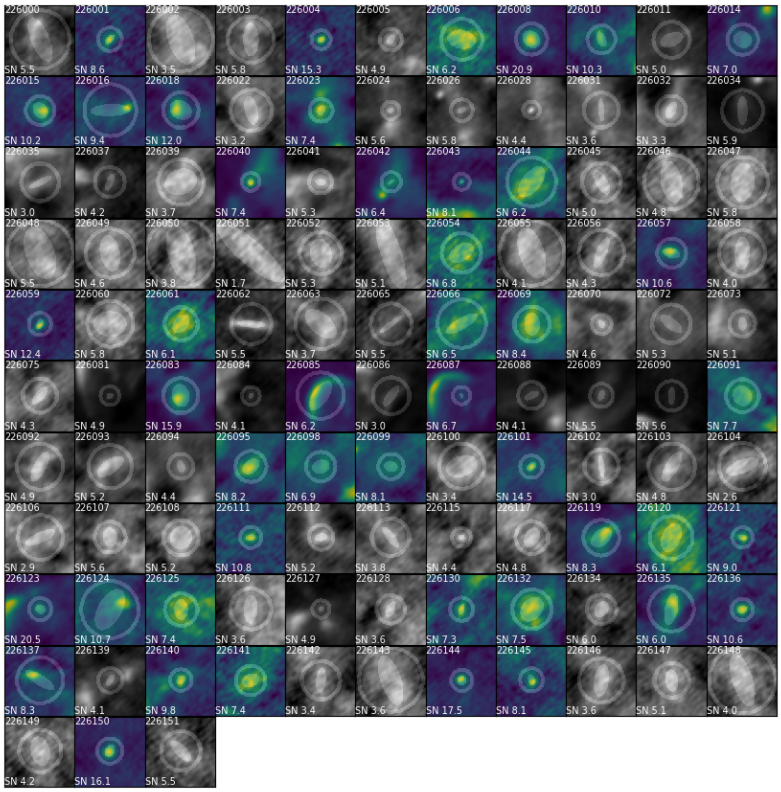
`RadioSource` objects should be named to distinguish them from others that may be combined later. This can be done using the keyword argument `name` when initializing the `RadioSource`, or afterwards by setting the instance attribute manually.

A catalog of sources can then be created using dendrograms.

```
>>> source_object.to_catalog()
Generating dendrogram using 738,094 of 49,787,136 pixels (1.4824994151099593% of data)
[========================================>] 100%
Computing catalog for 113 structures
[========================================>] 100%
<Table masked=True length=113>
_idx _index _name  ... rejected 226.1GHz_detected
 ...    ...   ...  ...      ...              ...
```

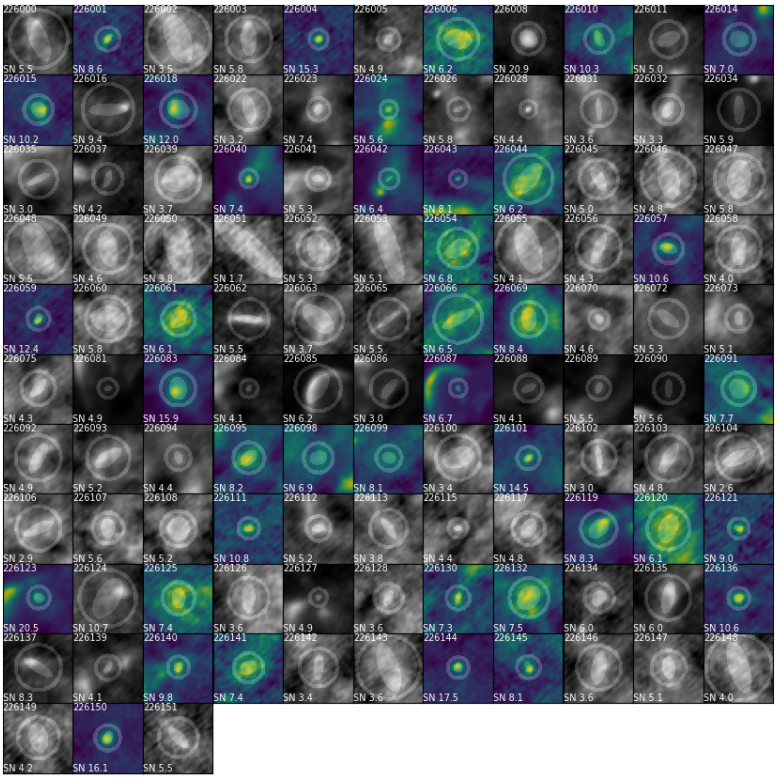
Most false detections (i.e., noise) can be filtered out using the `autoreject` method, with the signal-to-noise requirement specified as a keyword argument `threshold`. To view all the image cutouts around each source in a grid, call the `plot_grid` method. Rejected sources are greyed out, and the source apertures used to calculate the signal to noise are overplotted on top of the image.

```
>>> source_object.autoreject(threshold=6.)
>>> source_object.plot_grid(skip_rejects=False)
```



Sources can then be manually accepted or rejected using `accept` or `reject` (and reset entirely using `reset`).

```
>>> source_object.reject([226000, 226008, 226016, 226023, 226028, 226085, 226124, 226135, 226137])
>>> source_object.accept([226024, 226043, 226123])
>>> source_object.plot_grid(skip_rejects=False)
```



The identifiers used to accept and reject sources are those stored under the `_name` column in the `RadioSource` object's catalog. More information about accessing Astropy tables can be found in the Accessing a table section of the Astropy documentation.

`dendrocat.utils.saveregions` can be used to save a DS9 region file of all the apertures in a catalog. With an image open in DS9, overlay the region file to check each aperture and find the label (saved as `_name` in the source catalog) to use for manual acceptance or rejection.

```
>>> dendrocat.utils.saveregions(source_object.catalog, '/path/to/outputfile.reg')
```

The `RadioSource` object is (more or less) complete after source rejection is finished. It can be combined with other `RadioSource` objects to form a `MasterCatalog`. The catalogs of all the `RadioSource` objects are combined using `dendrocat.utils.match`.

```
from astropy.io import fits
from dendrocat import RadioSource, MasterCatalog
from dendrocat.utils import match

source_object1 = RadioSource(fits.open('/path/to/file1.fits'), name='so1')
source_object2 = RadioSource(fits.open('/path/to/file2.fits'), name='so2')

source_object1.autoreject()
source_object2.autoreject()

combined_catalog = match(source_object1.catalog, source_object2.catalog)

mastercatalog = MasterCatalog(source_object1, source_object2, catalog=combined_catalog)
```

Note: `match` operates on `RadioSource` and `MasterCatalog` objects only, and will not take plain catalogs as arguments. To match catalogs that have been manually edited, filtered, etc., make customizations to the `RadioSource` and `MasterCatalog` catalogs before matching.

The `MasterCatalog` stores each of the `RadioSource` objects as instance attributes. These can be accessed using the `__name__` of each `RadioSource`. It also has its own catalog, which is usually the matched catalog of its constituent `RadioSource` objects.

```
>>> mastercatalog.__dict__.keys()
dict_keys(['catalog', 'accepted', 'so1', 'so2'])

>>> mastercatalog.so1
<dendrocat.radiosource.RadioSource at 0x7f0c25f29fd0>
```

The main purpose of the `MasterCatalog` is as a framework for photometry. Photometry is done using `Aperture` objects, which define the shape and behavior of the apertures used to photometer the sources in a catalog.

The `MasterCatalog` class performs photometry by taking an aperture and placing it over the locations of all the sources in its catalog. The apertures may also be scaled according to the FWHM of the source.

An `Aperture` should be made into an instance if you want to use fixed-dimension apertures—for example, a circular aperture with a constant radius of 15 pixels.

```
import astropy.units as u
from dendrocat.aperture import Circle, Annulus

# Define a fixed-radius circular aperture in pixels
fixed_circle = Circle([0, 0], 15*u.pix, name='fixedcirc')

# Define a fixed-dimension annular aperture in pixels
fixed_annulus = Annulus([0, 0], 30, 40, unit=u.pix, name='fixedannulus')
```

Now, photometry can be done on the `MasterCatalog` object.

```
mastercatalog.photometer(fixed_circle, fixed_annulus)
```

For apertures that change shape according to the major and minor FWHM of each source, simply use the class itself instead of creating an instance.

```
mastercatalog.photometer(Circle, Annulus)
```

The source catalog is updated to include photometry data for all available frequencies, in each of the specified apertures.

```
>>> mastercatalog.catalog.colnames
 [...
 '226.1GHz_fixedcirc_peak',
 '226.1GHz_fixedcirc_sum',
 '226.1GHz_fixedcirc_rms',
 '226.1GHz_fixedcirc_median',
 '226.1GHz_fixedcirc_npix',
 '93.0GHz_fixedcirc_peak',
 '93.0GHz_fixedcirc_sum',
 '93.0GHz_fixedcirc_rms',
 '93.0GHz_fixedcirc_median',
 '93.0GHz_fixedcirc_npix',
```

(continues on next page)

(continued from previous page)

```
 '226.1GHz_fixedannulus_peak',
 '226.1GHz_fixedannulus_sum',
 '226.1GHz_fixedannulus_rms',
 '226.1GHz_fixedannulus_median',
 '226.1GHz_fixedannulus_npix',
 '93.0GHz_fixedannulus_peak',
 '93.0GHz_fixedannulus_sum',
 '93.0GHz_fixedannulus_rms',
 '93.0GHz_fixedannulus_median',
 '93.0GHz_fixedannulus_npix',
 '226.1GHz_Circle_peak',
 '226.1GHz_Circle_sum',
 '226.1GHz_Circle_rms',
 '226.1GHz_Circle_median',
 '226.1GHz_Circle_npix',
 '93.0GHz_Circle_peak',
 '93.0GHz_Circle_sum',
 '93.0GHz_Circle_rms',
 '93.0GHz_Circle_median',
 '93.0GHz_Circle_npix',
 '226.1GHz_Annulus_peak',
 '226.1GHz_Annulus_sum',
 '226.1GHz_Annulus_rms',
 '226.1GHz_Annulus_median',
 '226.1GHz_Annulus_npix',
 '93.0GHz_Annulus_peak',
 '93.0GHz_Annulus_sum',
 '93.0GHz_Annulus_rms',
 '93.0GHz_Annulus_median',
 '93.0GHz_Annulus_npix']
```

Photometry data for each of the sources can then be accessed using each of these column names.

To save any catalog for later use, use `write`.

```
>>> mastercatalog.catalog.write('/path/to/outfile.dat', format='ascii', overwrite=True)
```

Part III

Using dendrocat

In-depth documentation is linked below.

CHAPTER 1

The RadioSource Class

1.1 The RadioSource Class

Initializing a `RadioSource` object will attempt to extract data from the FITS file header of the image. Since FITS headers vary between telescopes and observations, support for new header formats is added as needed.

Currently, `dendrocat` supports FITS header information extraction from the following telescopes:

- EVLA
- ALMA

Note: In a future update, this functionality will be replaced by a `fitsconfig` file that will tell the program where to find the information it needs. Users can add to their `fitsconfig` to handle unsupported formats.

1.1.1 Manually Setting FITS Header Information

If the FITS header is not recognized, the missing attributes will be printed to the console so they can be set manually. Typically, this will include

- `nu` : The central frequency of the observation. Specified as a `astropy.units.quantity.Quantity`.
- `freq_id` : A string used to identify different images at a glance. For a 226 GHz observation, for example, the `freq_id` could be `'226.1GHz'`.
- `metadata` : Image metadata required for generating a dendrogram.

```
import dendrocat
from astropy.io import fits
import astropy.units as u

source_object = dendrocat.RadioSource(fits.open('/path/to/file.fits'))
```

(continues on next page)

(continued from previous page)

```
# Manually set RadioSource attributes
source_object.nu = 226.1*u.GHz
source_object.freq_id = '226.1GHz'
source_object.set_metadata()
```

If the wcs, beam, and pixel scale are successfully extracted from the FITS header, after manually setting nu the method `set_metadata` can be called to fill in the rest of the metadata.

Warning: FITS header formats from telescopes other than EVLA and ALMA are likely not supported at this time. The ability to load data with custom headers is in development.

1.1.2 Custom Dendrograms

The method `to_dendrogram` returns a `astrodendro.dendrogram.Dendrogram` object and adds it as an instance attribute of the `RadioSource` object.

```
>>> source_object.to_dendrogram()
Generating dendrogram using 738,094 of 49,787,136 pixels (1.4824994151099593% of data)
[========================================>] 100%
<astrodendro.dendrogram.Dendrogram at 0x7f00edfe7ac8>

>>> source_object.dendrogram
<astrodendro.dendrogram.Dendrogram at 0x7f00edfe7ac8>
```

Default dendrogram parameters are produced when the `RadioSource` object is first initiated. They are based on the standard deviation of pixel values across the whole image, and may need to be adjusted depending on how noisy the image is.

```
# Option 1 - Set dendrogram parameters as instance attributes
import numpy as np
source_object.min_value = 1.4*np.nanstd(source_object.data)
source_object.min_delta = 1.2*source_object.min_value
source_object.min_npix = 10
custom_dendro = source_object.to_dendrogram()

# Option 2 - Set dendrogram parameters as keyword arguments
custom_dendro = source_object.to_dendrogram(min_value=1e-4, min_delta=1.5e-4, min_npix=10)
```

1.1.3 Making A Source Catalog

`to_catalog` returns a source catalog as an `astropy.table.Table` object. It also saves the catalog as an instance attribute. If this method is called and the `RadioSource` object has no existing dendrogram, one will be generated using default parameters.

```
>>> source_object.to_catalog()
Computing catalog for 113 structures
[========================================>] 100%
<Table masked=True length=113>
_idx _index _name  ... rejected 226.1GHz_detected
 ...    ...   ...  ...      ...              ...
```

(continues on next page)

(continued from previous page)

```
>>> source_object.catalog
<Table masked=True length=113>
_idx _index _name  ... rejected 226.1GHz_detected
 ...    ...   ...  ...      ...              ...
```

To generate a catalog from a specific `astrodendro.dendrogram.Dendrogram` object, use the `dendrogram` keyword argument.

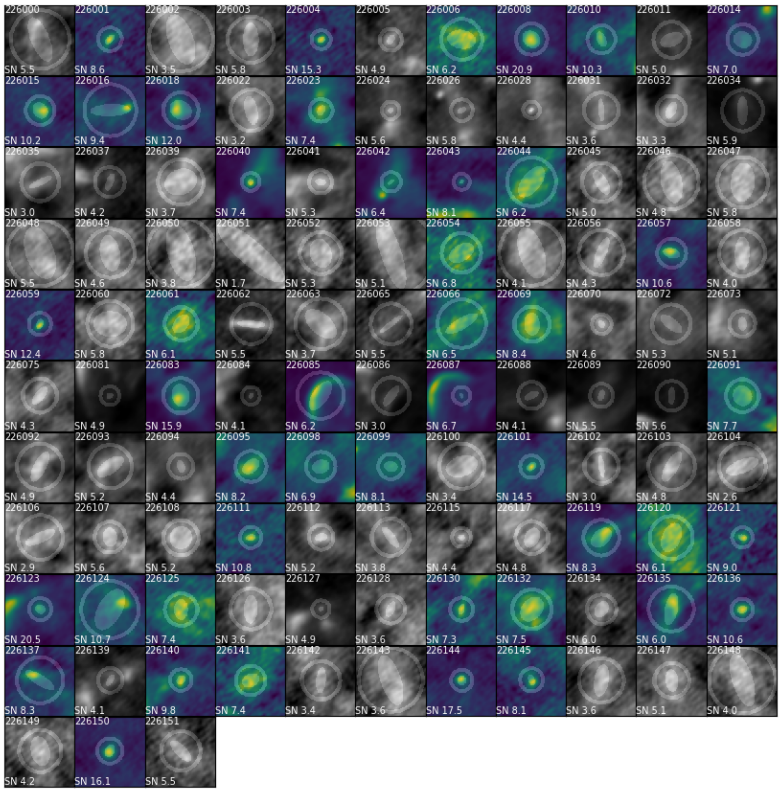
```
>>> custom_dendro = source_object.to_dendrogram(min_value=1e-4, min_delta=1.5e-4, min_npix=10)
Generating dendrogram using 738,094 of 49,787,136 pixels (1.4824994151099593% of data)
[========================================>] 100%

>>> source_object.to_catalog(dendrogram=custom_dendro)
Computing catalog for 113 structures
[========================================>] 100%
 <Table masked=True length=113>
_idx _index _name  ... rejected 226.1GHz_detected
 ...    ...   ...  ...      ...              ...
```

1.1.4 Rejection and Plotting Grids

To flag false detections, the `autoreject` method can be used.

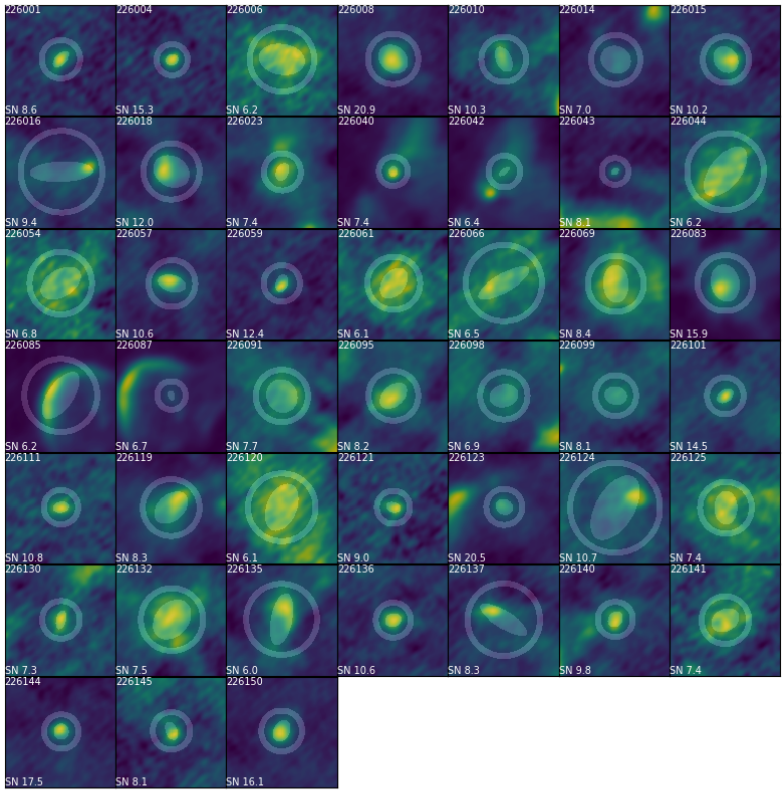
`plot_grid` displays cutout regions around each of the detected sources, as well as the apertures used to calculate signal-to-noise. Rejected sources show up in grey.

```
>>> source_object.autoreject(threshold=6.)
>>> source_object.plot_grid(skip_rejects=False)
```



To display only accepted sources, use the `skip_rejects` keyword argument.

```
>>> source_object.plot_grid(skip_rejects=True)
```



In practice, it is often useful to examine the sources in the context of the original image. In this case, `dendrocat.utils.saveregions` can be used to save a DS9 region file of all the apertures in a catalog. With an image open in DS9, the region file can be loaded in to check each aperture and find the proper identifier (saved as `_name` in the source catalog) to use for manual acceptance or rejection.

```
>>> dendrocat.utils.saveregions(source_object.catalog, '/path/to/outputfile.reg')
```

Sources can be manually accepted and rejected using the `accept` and `reject` methods.

Accepted sources and rejected sources, including those set by `accept` and `reject`, are stored in their own catalogs and are accessible through the `RadioSource` object.

```
>>> source_object.accepted

<Table masked=True length=45>
 _idx _index _name  ... rejected 226.1GHz_detected
                    ...
int64 int64  str20  ...  int64         int64
----- ------ ------ ... -------- -----------------
    1      1 226001 ...        0                 1
    4      4 226004 ...        0                 1
    6      6 226006 ...        0                 1
    8      7 226008 ...        0                 1
  ...    ...    ... ...      ...               ...

>>> source_object.reject([226000, 226008])
```

(continues on next page)

(continued from previous page)

```
>>> source_object.accepted

<Table masked=True length=43>
 _idx _index _name  ... rejected 226.1GHz_detected
                    ...
int64 int64  str20  ...  int64         int64
----- ------ ------ ... -------- -----------------
    1      1 226001 ...        0                 1
    4      4 226004 ...        0                 1
    6      6 226006 ...        0                 1
   10      8 226010 ...        0                 1
  ...    ...    ... ...      ...               ...
```

All rejections can be cleared using `reset`.

```
>>> source_object.reset()
>>> source_object.rejected

<Table masked=True length=0>
 _idx _index _name ...  y_cen  rejected 226.1GHz_detected
                   ...
int64 int64  str20 ... float64  int64         int64
----- ------ ----- ... ------- -------- -----------------
```

1.1.5 Adding Sources

If a single dendrogram can't detect all the sources you're interested in, you can create separate dendrograms with different parameters and add sources of interest from their catalogs to the catalog associated with the `RadioSource` object.

```
import dendrocat
from astropy.io import fits

source_object = dendrocat.RadioSource(fits.open('/path/to/file.fits'))

# Generate two separate dendrograms
default_dend = source_object.to_dendrogram()
custom_dend = source.object.to_dendrogram(min_delta=0.0002, save=False)

# Make catalogs
default_catalog = source_object.to_catalog()
custom_catalog = source_object.to_catalog(dendrogram=custom_dend)
```

Sources of interest not included in the default catalog are added using `add_sources`.

```
# Grab sources of interest, rename them to avoid name overlap with the other catalog
sources_of_interest = custom_catalog.grab([226000, 226001, 226002])
sources_of_interest['_name'] = ['custom1', 'custom2', 'custom3']

source_object.add_sources(sources_of_interest)
```

CHAPTER 2

The MasterCatalog Class

2.1 The MasterCatalog Class

A `MasterCatalog` object combines multiple `RadioSource` objects, retaining each object's attributes while creating a new, combined catalog from the sources contained in their individual catalogs.

The `MasterCatalog` enables analysis not possible with a single `RadioSource` object, such as photometry and catalog cross-matching.

2.1.1 Adding Objects

Additional `RadioSource` or `MasterCatalog` objects can be added to an existing `MasterCatalog` using `add_objects`.

```
from dendrocat import RadioSource, MasterCatalog
from dendrocat.utils import match
from astropy.io import fits

source_object1 = RadioSource(fits.open('file1.fits'), name='so1')
source_object2 = RadioSource(fits.open('file2.fits'), name='so2')
source_object3 = RadioSource(fits.open('file3.fits'), name='so3')
```

If `source_object3` is a much lower-resolution image, you may want to forgo generating a dendrogram for it. The source regions from the other two higher-resolution images may be used instead.

```
source_object1.autoreject()
source_object2.autoreject()

mastercatalog = match(source_object1, source_object2)
```

Adding other `RadioSource` objects or `MasterCatalog` objects will preserve the existing `MasterCatalog`'s source catalog.

```
>>> mastercatalog.add_objects(source_object3)
>>> mastercatalog.__dict__.keys()
dict_keys(['catalog', 'accepted', 'so1', 'so2', 'so3'])
```

At this point, performing photometry yields photometry data for all three images, though only two images were used to detect the sources in the first place.

Note that the `MasterCatalog` which calls `add_objects` will always have its catalog preserved, and will take `RadioSource` objects from whatever is added to it.

```
>>> mastercatalog1 = MasterCatalog(so1, so2, catalog=cat_A)
>>> mastercatalog2 = MasterCatalog(so3, so4, catalog=cat_B)
>>> mastercatalog1.add_objects(mastercatalog2)
>>> mastercatalog1.catalog == cat_A
True

>>> mastercatalog1.__dict__.keys()
dict_keys(['catalog', 'accepted', 'so1', 'so2', 'so3', 'so4'])
```

2.1.2 Renaming Sources

Any source can have its _name set for distinguishability. Suppose we have a `MasterCatalog` object named `mc`.

```
#Index of the row in the source catalog containing the old name
index_of_old_name = [mc.catalog['_name'] == '226007']

# Assign new name to the row
mc.catalog['_name'][index_of_old_name] = 'w51d2'
```

This can also be done in one line.

```
>>> mc.catalog['_name'][mc.catalog['_name'] == '226007'] = 'w51d2'
```

Here, we first access the `_name` column of the catalog. Then, we index the column with the location in the catalog the `_name` is equal to the old name, `'226007'`. Then the entry in the catalog is set to be equal to `'w51d2'`

CHAPTER 3

Apertures

3.1 Apertures

3.1.1 Using Preset Apertures

The presets for an `Aperture` are `Ellipse`, `Circle`, and `Annulus`. Each of these subclasses inherit properties from the base class, `Aperture`.

When using `photometer`, an aperture must be specified. To use variable-width apertures that change with the FWHM of the sources in the catalog, use any of the `Aperture` subclasses as the `aperture` argument in `photometer`.

For this example, suppose we have a `MasterCatalog` object called `mc` (For directions on how to create this object, see the *Getting Started* section) and we want to do some photometry.

```
from dendrocat.aperture import Ellipse, Circle, Annulus

mc.photometer(Ellipse, Circle, Annulus)
```

This will take the `MasterCatalog`'s catalog source entries and use their major and minor FWHM as aperture radii, where appli

- For a `Circle`, the radius is the source's major FWHM.
- For a `Ellipse`, the major and minor FWHM of the source, as well as its position angle, are used directly as the parameters of the elliptical aperture.
- For a `Annulus`, the inner and outer radii of the aperture are determined by the major FWHM of the source, as well as the `annulus_padding` and `annulus_width` attributes of the `RadioSource` object storing the image data.

Annulus Inner Radius = Source Major FWHM + Annulus Padding

Annulus Outer Radius = Source Major FWHM + Annulus Padding + Annulus Width

These two parameters ensure that annular apertures don't overlap with source apertures, and can be tuned within each `RadioSource` object.

3.1.2 Defining Custom Apertures

To use fixed apertures instead, create an instance of any of the `Aperture` subclasses with the desired parameters. Parameters can be specified in pixel or degree coordinates.

```
import astropy.units as u

# Define a fixed-radius elliptical aperture in pixels
fixed_ellipse_pix = Ellipse([0,0], 15*u.pix, 10*u.pix, 30*u.deg, name=ellipsepix)

# Define a fixed-radius elliptical aperture in degrees
fixed_ellipse_deg = Ellipse([0,0], 15*u.arcsec, 10*u.arcsec, 30*u.deg, name=ellipsedeg)
```

Photometry can then be performed exactly as if these were new aperture presets.

```
mc. photometer(fixed_ellipse_pix, fixed_ellipse_deg)
```

Note: The first argument of any `Aperture` subclass is always `center`. When creating an instance of the `Aperture` subclasses, this argument can be filled with any two coordinates—they will be overwritten with the source objects' center coordinates when photometry is performed.

CHAPTER 4

Reference/API

4.1 Reference/API

4.1.1 dendrocat Package

Functions

test(**kwargs)	Run the tests for the package.
ucheck(quantity, unit)	Check if a quantity already has units, and attempt conversion if so.

test

dendrocat.**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'. Accepts comma separated string to specify multiple packages. If nothing is specified all default 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 @pytest.mark.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`.

ucheck

dendrocat.**ucheck**(*quantity*, *unit*)
Check if a quantity already has units, and attempt conversion if so.

Parameters

quantity
[scalar, array, or `Unit`] The quantity to check for units. If scalar, units will assumed to be the same as in the "unit" argument.

unit
[`Unit`] The unit to check against. If the "quantity" argument already has an associated unit, a conversion will be attempted.

Classes

MasterCatalog(*args[, catalog])	An object to store combined data from two or more RadioSource objects.
RadioSource(hdu[, name, freq_id])	An object to store radio image data.
UnsupportedPythonError	

MasterCatalog

class dendrocat.**MasterCatalog**(**args*, *catalog=None*)

Bases: object

An object to store combined data from two or more RadioSource objects.

Create a new master catalog object.

Parameters

catalog
[astropy.table.Table object] The master table from which to build the catalog object.

***args**
[radiosource.RadioSource objects] RadioSource objects from which the master table was built.

Methods Summary

add_objects(*args)	Add a new RadioSource object to the existing master catalog.
add_sources(*args)	Add source entries from another catalog.
ffplot(rsobj1, rsobj2[, apertures, ...])	Produce a flux-flux plot for two RadioSource objects.
grab(name[, skip_rejects])	Grab a source or sources by name.
photometer(*args[, catalog])	Add photometry data columns to the master catalog.

Methods Documentation

add_objects(**args*)

Add a new RadioSource object to the existing master catalog.

Parameters

***args**
[‘~dendrocat.RadioSource‘ objects] RadioSource objects to add to the master catalog.

add_sources(**args*)

Add source entries from another catalog.

Parameters

***args**
[astropy.table.Table objects] Source tables to vertically stack with the existing master cat-

alog.

ffplot(*rsobj1*, *rsobj2*, *apertures=[]*, *bkg_apertures=[]*, *alphas=None*, *peak=False*, *label=False*, *log=True*, *outfile=None*)

Produce a flux-flux plot for two `RadioSource` objects.

Parameters

rsobj1
[`RadioSource` object] One of two radio source objects from which to make a flux-flux plot.

rsobj2
[`RadioSource` object] The other of two radio source objects from which to make a flux-flux plot.

apertures
[list] List of `Aperture` objects to use for source apertures.

bkg_apertures
[list] List of `Aperture` objects to use for background apertures.

alphas
[list, optional] Spectral indices to overplot on top of the flux-flux data. 1, 2, and 3 will be used by default.

peak
[bool, optional] If enabled, peak flux inside the aperture is used instead of aperture sum. Disabled by default.

label
[bool, optional] If enabled, labels will be printed on the plot to identify sources. Disabled by default.

log
[bool, optional] If enabled, results will be shown on log-log axes. Enabled by default.

outfile
[str, optional] If provided, output plot will be saved to this file path.

grab(*name*, *skip_rejects=False*)

Grab a source or sources by name.

Parameters

name
[str or list] String or list of strings to search the catalog "_name" header for.

skip_rejects
[bool, optional] If enabled, rejected sources will not be queried. Disabled by default.

photometer(**args*, *catalog=None*)

Add photometry data columns to the master catalog.

Parameters

args
[`Aperture` objects] The apertures to use for photometry. Can be given as either instances or objects, to use fixed or variable aperture widths, respectively.

catalog
[astropy.table.Table object] The catalog from which to extract source coordinates and ellipse parameters.

RadioSource

class dendrocat.**RadioSource**(*hdu*, *name=None*, *freq_id=None*)
Bases: object

An object to store radio image data.

Parameters

hdu
[PrimaryHDU] An astropy FITS HDU object containing the radio image data and header.

region_id
[str, optional] An identifier specifying what sky object the radio image contains.

freq_id
[str, optional] An identifier specifying the observation frequency (Ex: 226.0GHz). If not specified, it will be generated from the FITS image header.

Methods Summary

accept(accepted_list)	Accept specific sources in the catalog.
add_sources(*args)	Adds external source entries to the existing catalog.
autoreject([threshold])	Reject noisy detections.
dump(outfile)	Dump the RadioSource object via pickle.
get_pixels(aperture[, catalog, data, ...])	Get pixels within an aperture for each entry in the specified catalog.
get_snr([source, background, catalog, data, ...])	Return the SNR of all sources in the catalog.
grab(name[, skip_rejects])	Search the catalog for an entry matching a specific name, and return
plot_grid([catalog, data, cutouts, ...])	Plot sources in a grid.
reject(rejected_list)	Reject specific sources in the catalog.
reset()	Reset all sources' rejection flags to 0 (all accepted).
set_metadata()	Sets RadioSource metadata using nu, WCS, and other FITS header data.
to_catalog([dendrogram])	Creates a new position-position catalog of leaves in a dendrogram.
to_dendrogram([min_value, min_delta, ...])	Calculates a dendrogram for the image.

Methods Documentation

accept(*accepted_list*)
Accept specific sources in the catalog.

Parameters

accpeted_list: list
A list of ‘‘_name‘‘s, for which each corresponding entry will be marked accepted.

add_sources(**args*)
Adds external source entries to the existing catalog.

Parameters

***args: '~astropy.table.Table'**
A source catalog containing the sources you wish to add to the existing catalog.

autoreject(*threshold=None*)
Reject noisy detections.

Parameters

threshold
[float, optional] The signal-to-noise threshold below which sources are rejected

dump(*outfile*)
Dump the RadioSource object via pickle.

Parameters

outfile
[str] Desired output file path.

get_pixels(*aperture*, *catalog=None*, *data=None*, *cutouts=None*, *save=True*)
Get pixels within an aperture for each entry in the specified catalog.

Parameters

aperture: '~dendrocat.aperture.Aperture'
The aperture determining which pixels to grab.

catalog: '~astropy.table.Table', optional
A source catalog containing the center positions of each source.

data: array-like
Image data for the sources in the catalog.

cutouts:
For developer use

Returns

pixels, masks

'~numpy.ndarray', '~numpy.ndarray'

get_snr(*source=None*, *background=None*, *catalog=None*, *data=None*, *cutouts=None*, *cutout_data=None*, *peak=True*, *save=True*)
Return the SNR of all sources in the catalog.

Parameters

source: array-like
Array of source fluxes to use in SNR calculation.

background: array-like
Array of background fluxes to use in SNR calculation.

catalog: ‘~astropy.table.Table‘
The catalog of sources for which to calculate the SNR.

data: array-like
Image data for the sources in the catalog.

cutouts:
For debugging. Provides a specific set of cutouts instead of letting the function generate them.

cutout_data:
For debugging. Provides a specific set of cutout data instead of letting the function generate it.

peak
[bool, optional] Use peak flux of source pixels as ‘signal’. Default is True.

save
[bool, optional] If enabled, the snr will be saved as a column in the source catalog and as an instance attribute. Default is True.

Returns

‘~numpy.ndarray‘

`grab`(*name*, *skip_rejects=False*)

Search the catalog for an entry matching a specific name, and return
it.

Parameters

name: tuple, list, or str
The name or names of the sources to search for.

skip_rejects: bool, optional
If enabled, will only search accepted sources.

`plot_grid`(*catalog=None*, *data=None*, *cutouts=None*, *cutout_data=None*, *source_aperture=None*, *bkg_aperture=None*, *skip_rejects=True*, *outfile=None*, *figurekwargs={}*)
Plot sources in a grid.

Parameters

catalog
[astropy.table.Table object, optional] The catalog used to extract source positions.

data
[numpy.ndarray, optional] The image data displayed and used to make cutouts.

cutouts
[list of astropy.nddata.utils.Cutout2D objects, optional] Image cutout regions to save computation time, if they have already been calculated.

cutout_data
[list of numpy.ndarrays, optional] Image cutout region data to save on computation time, if it has already been calculated.

apertures
[list of dendrocat.aperture functions, optional] Apertures to plot over the image cutouts.

skip_rejects
[bool, optional] If enabled, don't plot rejected sources. Default is True.

reject(*rejected_list*)
Reject specific sources in the catalog.

Parameters

rejected_list: list
A list of ‘‘_name‘‘s, for which each corresponding entry will be marked rejected.

reset()
Reset all sources' rejection flags to 0 (all accepted).

set_metadata()
Sets RadioSource metadata using nu, WCS, and other FITS header data.

to_catalog(*dendrogram=None*)
Creates a new position-position catalog of leaves in a dendrogram. This task will overwrite the existing catalog if there is one.

Parameters

dendrogram
[Dendrogram object, optional] The dendrogram object to extract sources from.

Returns

‘~astropy.table.Table‘

to_dendrogram(*min_value=None*, *min_delta=None*, *min_npix=None*, *save=True*)
Calculates a dendrogram for the image.

Parameters

min_value
[float, optional] Minimum detection level to be considered in the dendrogram.

min_delta
[float, optional] How significant a dendrogram structure has to be in order to be considered a separate entity.

min_npix
[float, optional] Minimum number of pixel needed for a dendrogram structure to be considered a separate entity.

save
[bool, optional] If enabled, the resulting dendrogram will be saved as an instance attribute. Default is True.

Returns

‘~astrodendro.dendrogram.Dendrogram‘ object
A dendrogram object calculated from the radio image.

UnsupportedPythonError

exception dendrocat.**UnsupportedPythonError**

Python Module Index

d

Index