What is Astropy

- http://www.astropy.org
- Well-organized community effort to create a vetted, well-documented set of astronomical libraries for python
  - Will ultimately be like idlastro for IDL on steroids
  - Currently at version 0.2.3
  - Python + numpy + matplotlib + astrolib is a fairly complete, free alternative to IDL
- Main leaders are Erik Tollerud (Yale), Thomas Robitaille (MPIA), Perry Greenfield (STSci)
  - Perry Greenfield is the software lead at STSci, has STSci programmers contributing
  - Paper on AstroPy submitted two days ago (to A&A)!

Installation

- Easiest with pip (see http://www.pip-installer.org/en/latest/installing.html#python-os-support)
  - `pip install astropy`
  - To upgrade: `pip install astropy –upgrade`
- Manual install:
  - Download tarball from http:///astropy.org
  - Or use git for bleeding-edge version (requires Cython also):
    - git clone git://github.com/astropy/astropy.git
  - `python setup.py build`
  - `python setup.py install`

Packages - Core

- Constants (astropy.constants)
- Units (astropy.units)
- N-dimensional datasets (astropy.nddata)
- Data Tables (astropy.table)
- Time and Dates (astropy.time)
- Astronomical Coordinate Systems (astropy.coordinates)
- World Coordinate System (astropy.wcs)

Packages – Data I/O & Computation

- FITS File handling (astropy.io.fits)
- ASCII Tables (astropy.io.ascii)
- VOTable XML handling (astropy.io.votable)
- Miscellaneous Input/Output (astropy.io.misc)
- Cosmological Calculations (astropy.cosmology)
- Astrostatistics Tools (astropy.stats)

Heritage

Several packages are direct ports from established libraries:

```
import pyfits           →    from astropy.io import fits as pyfits
import vo                    from astropy.io import vo
import pywcs                 from astropy import wcs as pywcs
```

Examples taken from astropy paper…

Tables

Create an empty table and add columns

```
>>> from astropy.table import Table, Column
>>> t = Table()
>>> t.add_column(Column(data=["a", "b", "c"], name="source"))
>>> t.add_column(Column(data=[1.2, 3.3, 5.3], name="flux"))
>>> print t
source flux
------ ----
a 1.2
b 3.3
c 5.3
```

Read a table from a file

```
>>> t1 = Table.read("catalog.vot")
>>> t1 = Table.read("catalog.tbl", format="ipac”)
```

Select all rows from t1 where the flux column is greater than 5

```
>>> t2 = t1[t1["flux"] > 5.0]
```

Manipulate columns

```
>>> t2.remove_column("J_mag")
>>> t2.rename_column("Source", "sources")
```

Write a table to a file

```
>>> t2.write("new_catalog.hdf5")
>>> t2.write("new_catalog.tex")
```

FITS

Read in a FITS file from disk

```
>>> from astropy.io import fits
>>> hdus = fits.open("sample.fits")
```

Access the header of the first HDU:

```
>>> hdus[0].header
```

SIMPLE = T

BITPIX = -32

NAXIS = 3

NAXIS1 = 200

NAXIS2 = 200

NAXIS2 = 10

EXTEND = T

...

Access the shape of the data in the first HDU:

```
>>> hdus[0].data.shape
```

(10, 200, 200)

Update/add header keywords

```
>>> hdus[0].header["TELESCOP"] = "Mt Wilson"
>>> hdus[0].header["OBSERVER"] = "Edwin Hubble"
```

Multiply data by 1.2

```
>>> hdus[0].data *= 1.2
```

Write out to disk

```
>>> hdus.writeto("new_file.fits")
```

Constants

Access physical constants:

```
>>> from astropy import units as u
>>> from astropy import constants as c
>>> print c.G
```

Name = Gravitational constant

Value = 6.67384e-11

Error = 8e-15

Units = m3 / (kg s2)

Reference = CODATA 2010

Combine quantities and constants:

```
>>> F = (c.G * (3 * c.M_sun) * (2 * u.kg) /
... (1.5 * u.au) ** 2)
>>> F.to(u.N)
```

<Quantity 0.01581795428812989 N>

Quantities

Define a quantity from scalars and units:

```
>>> from astropy import units as u
>>> 15.1 * u.m / u.s
<Quantity 15.1 m / s>
```

Convert a distance:

```
>>> (1.15e13 * u.km).to(u.pc)
<Quantity 0.372689618289 pc>
```

Make use of the unit equivalencies:

```
>>> e = 130. * u.eV
>>> e.to(u.Angstrom, equivalencies=u.spectral())
<Quantity 95.37245609234003 Angstrom>
```

Combine quantities:

```
>>> x = 1.4e11 * u.km / (0.7 * u.Myr)
>>> x
<Quantity 2e+11 km / Myr>
```

Convert to SI and CGS units:

```
>>> x.si
<Quantity 6.33761756281 m / s>
>>> x.cgs
<Quantity 633.761756281 cm / s>
```

Use units with NumPy arrays

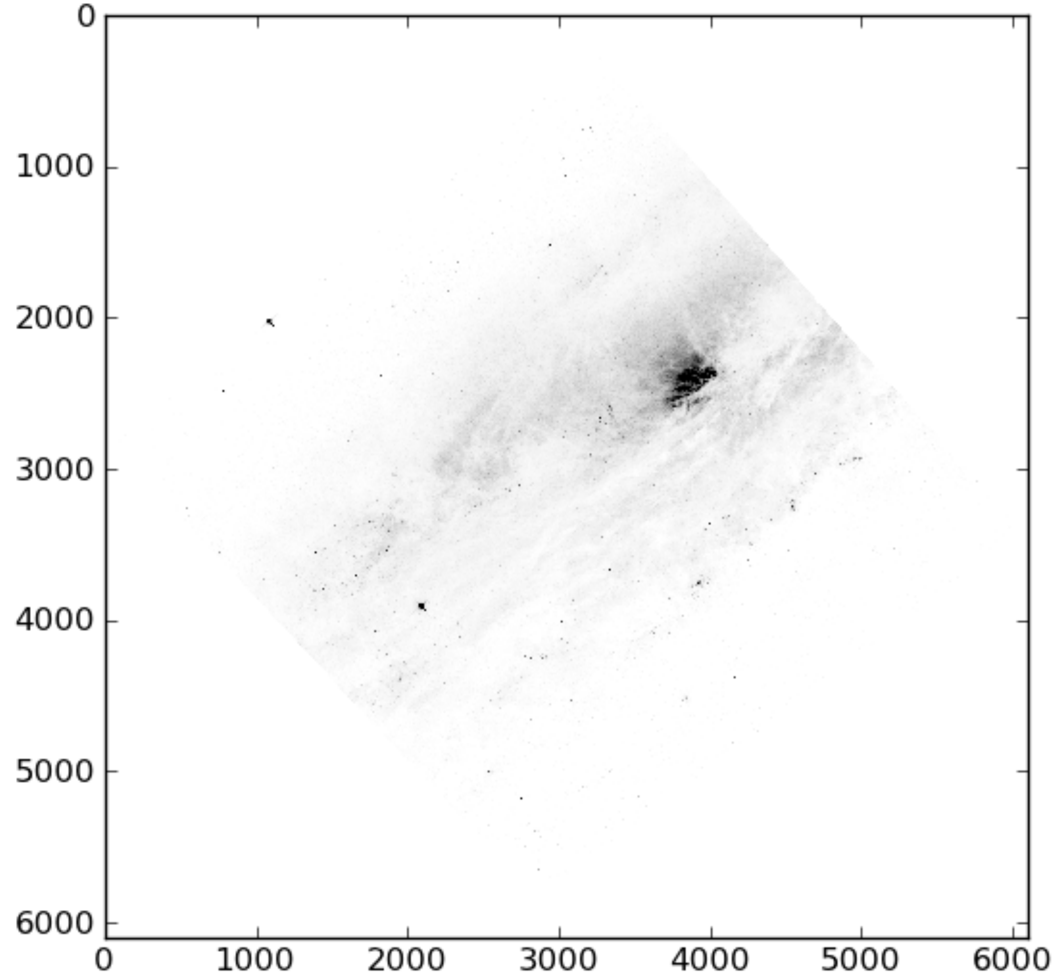
```
>>> import numpy as np
>>> d = np.array([1, 2, 3, 4]) * u.m
>>> d.to(u.cm)
<Quantity [ 100. 200. 300. 400.] cm>
>>> d * 1. / 50. * u.s ** -1
<Quantity [ 2. 4. 6. 8.] cm / s>
```

Example

- Read in an HST FITS image
- Display it
- Convert galaxy center position in RA/Dec. to pixel coordinates
- Display only small region around center
- Plot pixel values in a cut through the center
- Extra (with pyds9 also installed):
  - Send image to DS9
  - Send box region to DS9 corresponding to displayed central region

Read and Display Image

```
from astropy.io import fits as pyfits
import matplotlib.pyplot as plt
f = pyfits.open("HST_10915_98_ACS_WFC_F814W_drz.fits")
im = f[1].data
h = f[1].header
imgplot = plt.imshow(im)
imgplot.set_cmap("Greys")
imgplot.set_clim(0, 10)
plt.show()
```



Note images display upside down (odds are there is a way around this I haven't found yet)

Convert galaxy center position to pixels

Gets name from SIMBAD

```
import astropy.coordinates as coords
m = coords.ICRSCoordinates.from_name("ngc 253")
# <ICRSCoordinates RA=11.88833 deg, Dec=-25.28806 deg>
import astropy.wcs as wcs
pixcrds = w.wcs_world2pix([[m.ra.degrees, m.dec.degrees]], 1)
# 1 = FITS rather than C convention)
x,y = pixcrds[0]
```

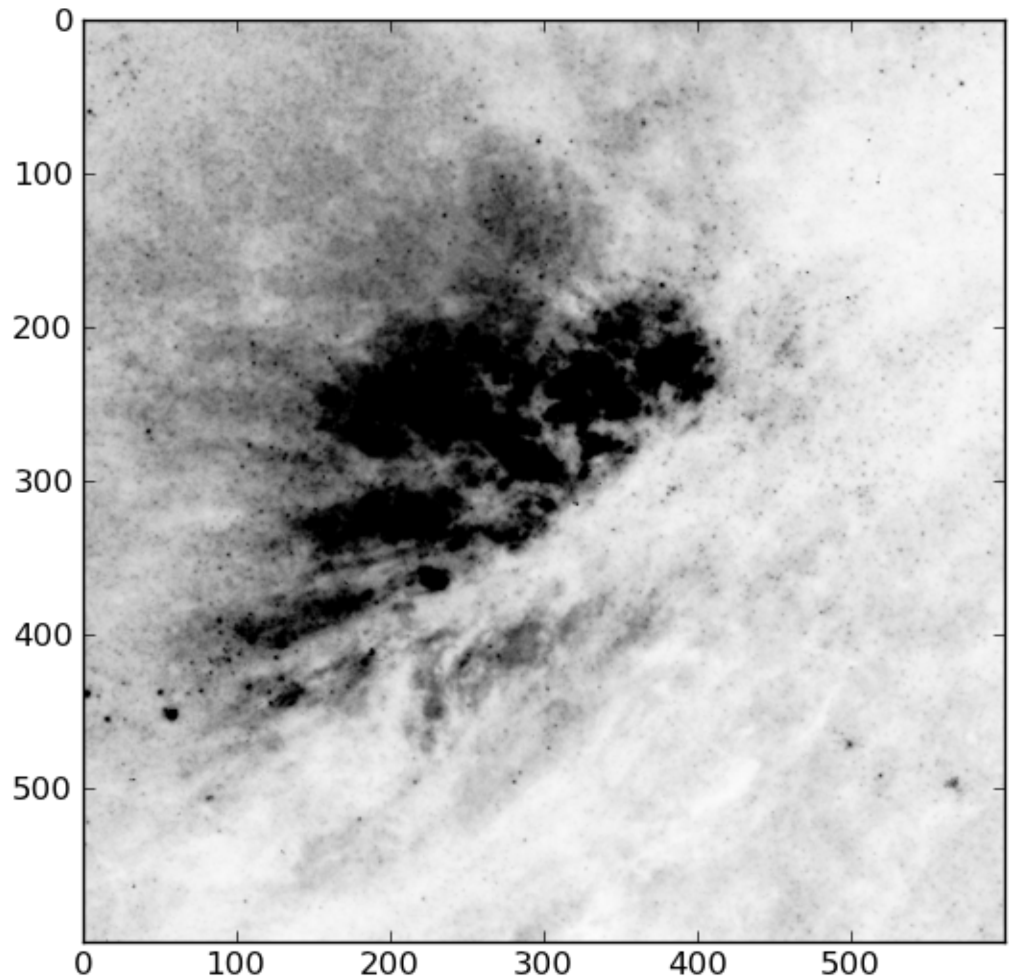
```
In [188]: print x,y
3922.71258601 2450.46998371
```

Right now wcs is based on a c library, makes for a bit of an awkward interface, e.g., position ra, dec in degrees is extracted from coords object as an array with (ra, dec) and returns an array of coordinates (hence x,y = pixcrds[0])
Later wcs will be a “first-class” Python package that will take astropy coordinates

Display only small region around center

```
subim = im[y-300:y+300, x-300:x+300]
imgplot = plt.imshow(subim)
imgplot.set_clim(0,10)
imgplot.set_cmap("Greys")
plt.show()
```

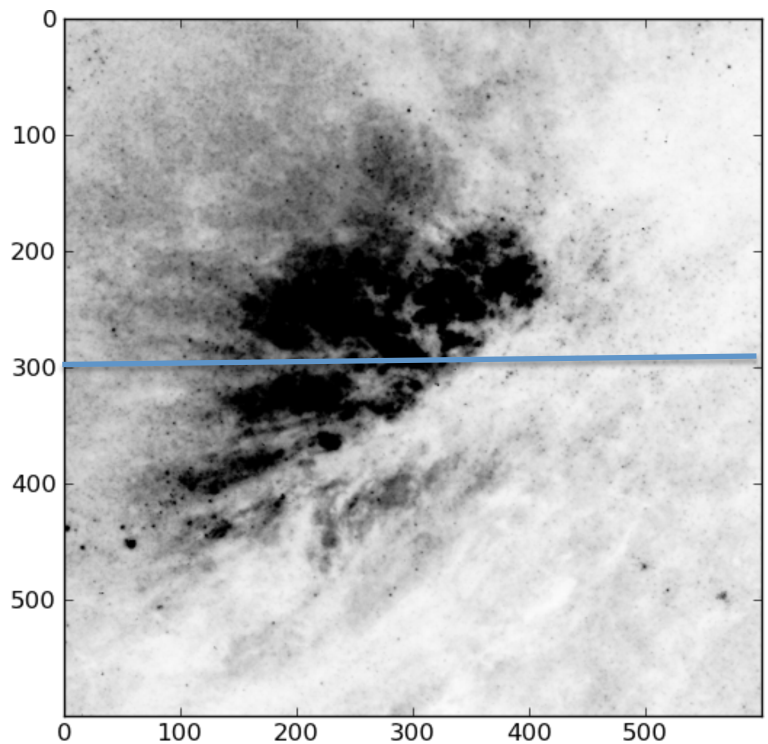
N.B., y index first!

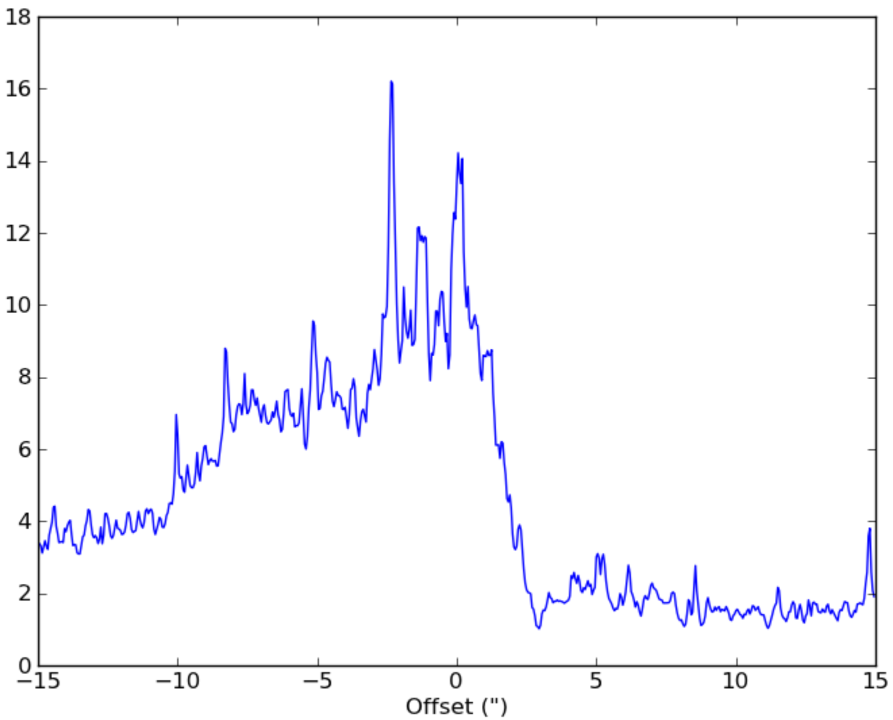


Plot pixel values in a cut through the center

Not a good approach

```
plate_scale = h["CD2_2"]*3600.
offset = (numpy.arange(600)-300.)*plate_scale
plot(offset, subim[300,:])
xlabel("Offset (\")")
```



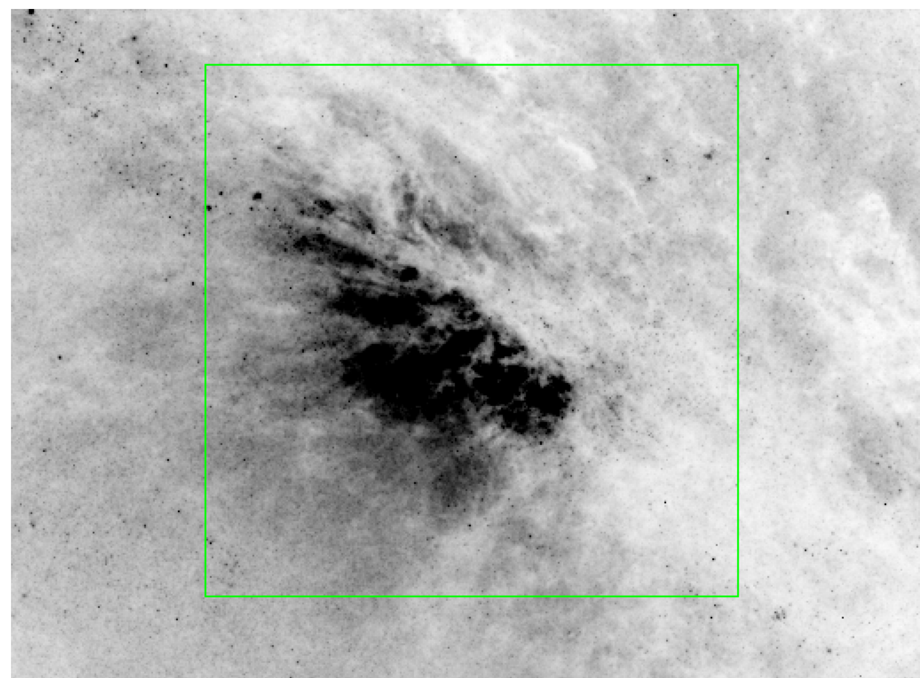


pyds9

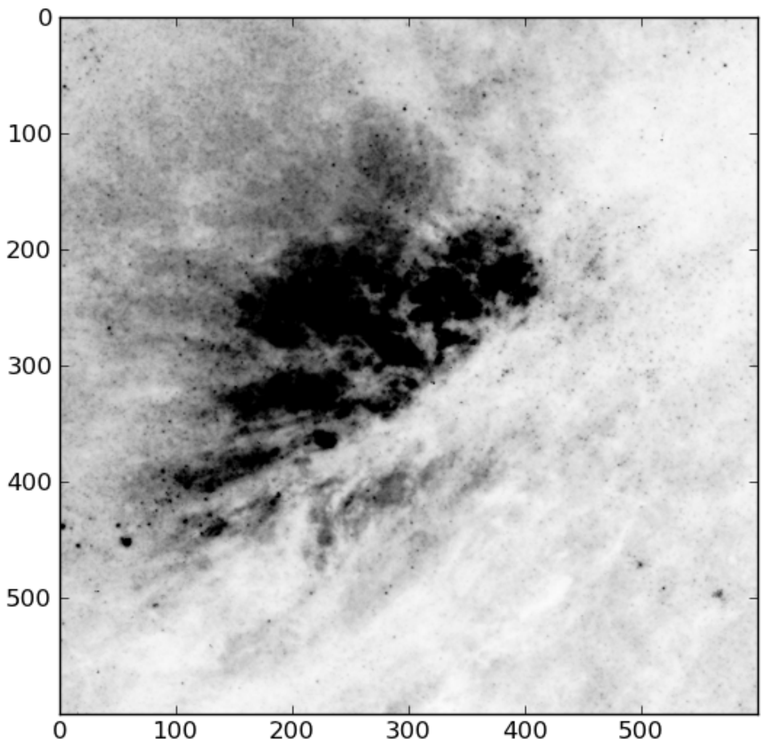
- Communicates with ds9 instance using XPA
- Download
  - http://hea-www.harvard.edu/saord/ds9/site/Download.html
- Seems to have some weird conflict with pylab (import ds9 hangs when tried within "ipython –pylab"), importing ds9 first before pylab works though

Display image using ds9 and draw box

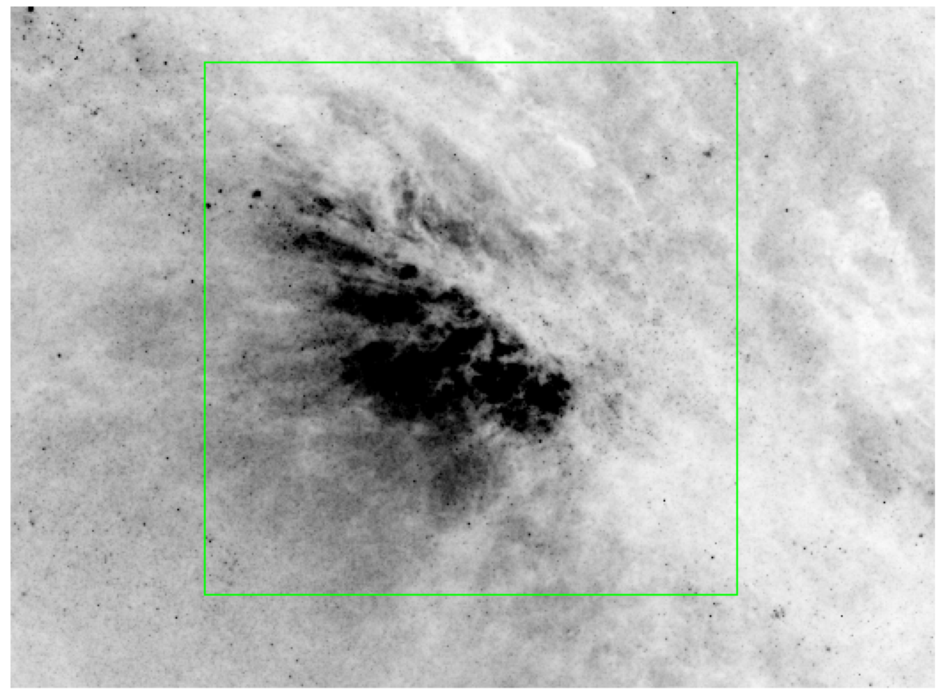
```
from ds9 import *
ds = ds9()
ds.set_pyfits(f)
reg = 'fk5; box(%f,%f,30",30",0.)' % \
   (m.ra.degrees,m.dec.degrees)
# reg =  fk5; box(11.888330,-25.288060,30",30",0.)
ds.set("regions", reg)
```



pyfits



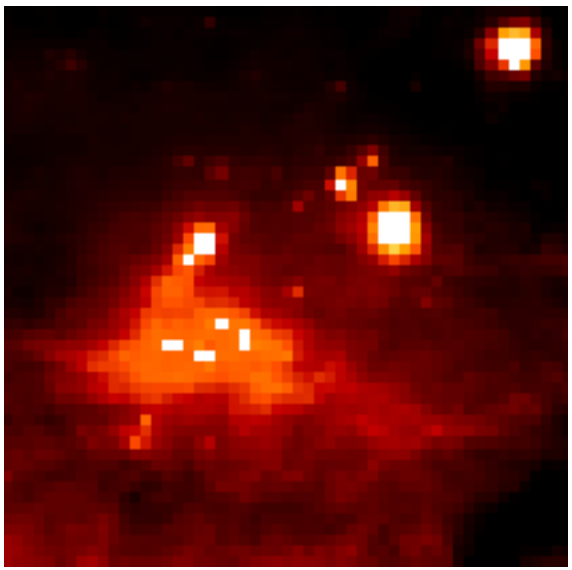
ds9



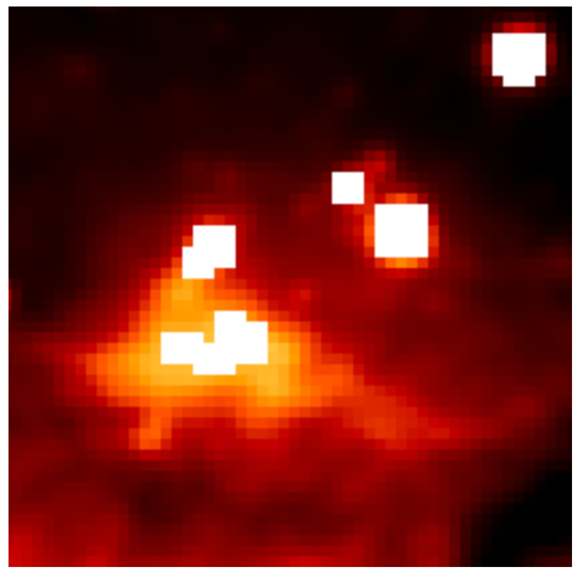
Convolution

Both scipy and astropy have convolution routines, but astropy handles Not-a-Number (NaN):

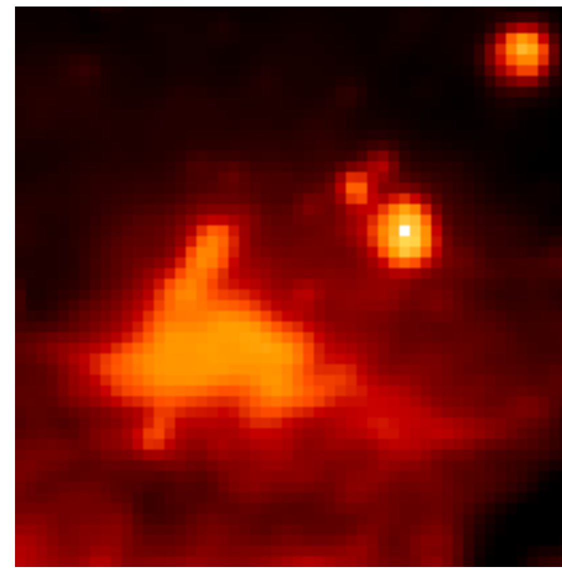
Original



SciPy



AstroPy



http://docs.astropy.org/en/latest/nddata/convolution.html

Potential of AstroPy (once its fully functional and operational)

- Virtual Observatory queries of data (tables, images, spectra)
- Tabular data from VO and/or journal papers, CDS, etc.
  - Automatically recognize coordinates and units
  - Collecting SEDs:
    - Some papers give fluxes, some give luminosities, some sources only have archival data
- Astropy versions of common tasks like aperture photometry
  - Given STSci's buy in of python and astropy, JWST analysis will probably be doable mostly or entirely in python
- Model fitting
- LSST processing software written in C++ with python wrappers using SWIG, some functionality may be added to astropy (maybe just as an "external library")