

7. ASTRONOMICAL SPECIFICS

**JHU Physics & Astronomy
Python Workshop 2015**

Lecturer: Mubdi Rahman

NOW, FOR SOME FUN!

We've been showing you how to do things in Python that you could (for the most part) in many other scripting/programming languages. Let's show you things that make Python great!

NOW, FOR SOME FUN!

We've been showing you how to do things in Python that you could (for the most part) in many other scripting/programming languages. Let's show you things that make Python great!

PRO TIP:

In this section, we'll also be using the `astroquery` package. You should install this now using `pip` or the package manager.

WORLD COORDINATE SYSTEM

Typically, you need to know *where* on an image each pixel is in astronomical coordinates (either RA & Dec, or maybe Galactic Longitude and Latitude). This information (typically referred to as WCS) is typically stored in the header of your FITS file. To use this information, you can use the `astropy.wcs` module:

```
from astropy.wcs import WCS

# If you have a header object named 'head1' from
# either fits.getheader() or fits.open():
w = WCS(head1)

# Or just getting one from a file itself:
w = WCS(filename.fits)
```

FROM PIXELS TO COORDINATES

The wcs object contains functions that conversion from pixel to world coordinates and vice versa:

```
# From pixel => world:
ra, dec = w.all_pix2world(xpx, ypx, 0) # Can be lists

# The third parameter indicates if you're starting
# from 0 (Python-standard) or 1 (FITS-standard)

# From world => pixel:
xpx, ypx = w.all_world2pix(ra, dec, 0)
```

FROM PIXELS TO COORDINATES

The `wcs` object contains functions that conversion from pixel to world coordinates and vice versa:

```
# From pixel => world:
ra, dec = w.all_pix2world(xpx, ypx, 0) # Can be lists

# The third parameter indicates if you're starting
# from 0 (Python-standard) or 1

# From world => pixel:
xpx, ypx = w.all_world2pix(ra,
```

PRO TIP:

Note that the order of the input are in standard Cartesian ordering, and the **opposite** of the FITS image read in.

PLOTTING A FITS IMAGE

It is important to note that most often, the pixels from the FITS image are not perfectly aligned with the coordinate grid, and aren't necessarily the same size on sky throughout the image. **In these cases, it is critical to use `pcolor` (or `pcolormesh`) to get the orientations correct.**

If you want to use `imshow`, remember anything else you'd like to plot should be converted into pixel coordinates through the `w.all_world2pix()` function.

Next, we'll run through plotting an image:

PLOTTING A FITS IMAGE

```
# Getting Data
imfile = fits.open(filename)
header, im = imfile[0].header, imfile[0].data
w = WCS(header)

# Making Indices
xpx = np.arange(im.shape[1]+1)-0.5
ypx = np.arange(im.shape[0]+1)-0.5
xlist, ylist = np.meshgrid(xpx, ypx)
ralist, declist = w.all_pix2world(xlist, ylist, 0)

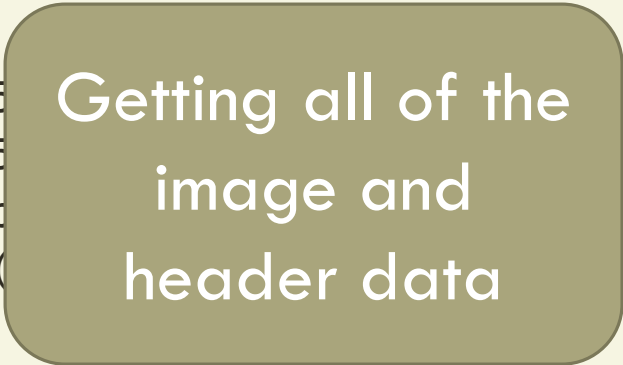
# Plotting
plt.pcolormesh(ralist, declist, im, vmin=min, vmax=max)
```


PLOTTING A FITS IMAGE

```
# Getting Data
imfile = fits.open(filename)
header, im = imfile[0].header, imfile[0].data
w = WCS(header)
```

```
# Making Indices
xpx = np.arange(im.shape[1]+1)-0.5
ypx = np.arange(im.shape[0]+1)-0.5
xlist, ylist = np.meshgrid(xpx, ypx)
ralist, declist = w.all_pix2world(xlist, ylist, 1)
```

```
# Plotting
plt.pcolormesh(ralist, declist, im, vmin=min, vmax=max)
```



Getting all of the
image and
header data

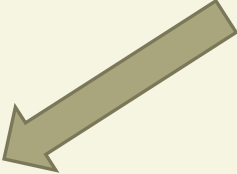
PLOTTING A FITS IMAGE

```
# Getting Data
imfile = fits.open(filename)
header, im = imfile[0].header, imfile[0].data
w = WCS(header)
```

```
# Making Indices
xpx = np.arange(im.shape[1]+1)-0.5
ypx = np.arange(im.shape[0]+1)-0.5
xlist, ylist = np.meshgrid(xpx, ypx)
ralist, declist = w.all_pix2world(xlist, ylist, 0)
```

```
# Plotting
plt.pcolormesh(ralist, declist, im, vmin=min, vmax=max)
```

Making list of bin edges. Remember there are $N+1$ bins. The coordinates are defined on the centre of the pixel, so the first bin edge is at -0.5 .



PLOTTING A FITS IMAGE

```
# Getting Data
imfile = fits.open(f
header, im = imfile[0
w = WCS(header)
```

```
# Making Indices
```

```
xpx = np.arange(im.shape[1]+1)-0.5
ypx = np.arange(im.shape[0]+1)-0.5
```

```
xlist, ylist = np.meshgrid(xpx, ypx)
```

```
ralist, declist = w.all_pix2world(xlist, ylist, 0)
```

```
# Plotting
```

```
plt.pcolormesh(ralist, declist, im, vmin=min, vmax=max)
```

`np.meshgrid()` creates two, 2-D arrays filled with the values in the two 1-D arrays you give it:

```
np.meshgrid([1,2],[3,4]) =
    [1, 2] [3, 3]
    [1, 2] [4, 4]
```



PLOTTING A FITS IMAGE

Getting Data

```
imfile = fits.open(filename)  
header, im = imfile[0].header,  
w = WCS(header)
```

Making Indices

```
xpx = np.arange(im.shape[1]+1)-0.5  
ypx = np.arange(im.shape[0]+1)-0.5  
xlist, ylist = np.meshgrid(xpx, ypx)  
ralist, declist = w.all_pix2world(xlist, ylist, 0)
```

Plotting

```
plt.pcolormesh(ralist, declist, im, vmin=min, vmax=max)
```

Converting the indices
into RA, Dec values for all
values in the lists.



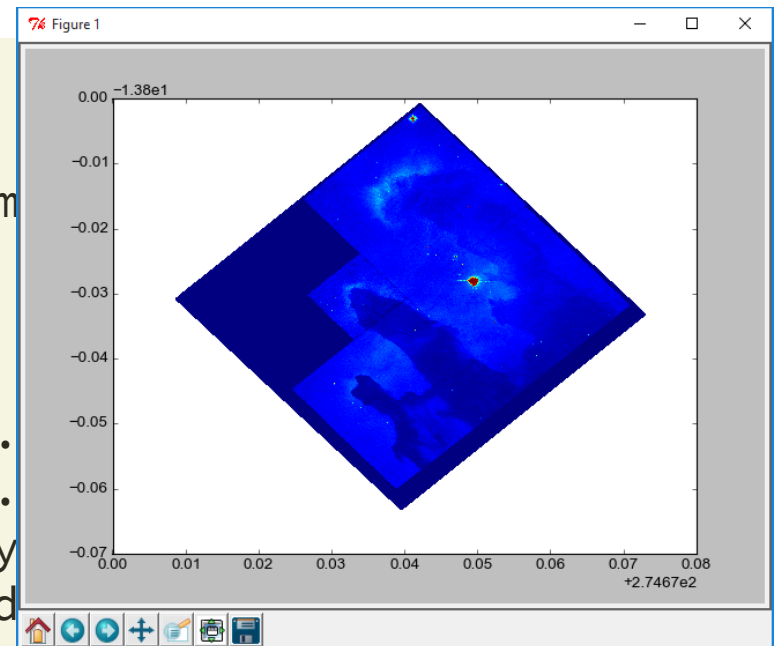
PLOTTING A FITS IMAGE

```
# Getting Data
imfile = fits.open(filename)
header, im = imfile[0].header, im
w = WCS(header)
```

```
# Making Indices
xpx = np.arange(im.shape[1]+1)-0.5
ypx = np.arange(im.shape[0]+1)-0.5
xlist, ylist = np.meshgrid(xpx, ypx)
ralist, declist = w.all_pix2world(xlist, ylist)
```

```
# Plotting
plt.pcolormesh(ralist, declist, im, vmin=min, vmax=max)
```

Final Plot



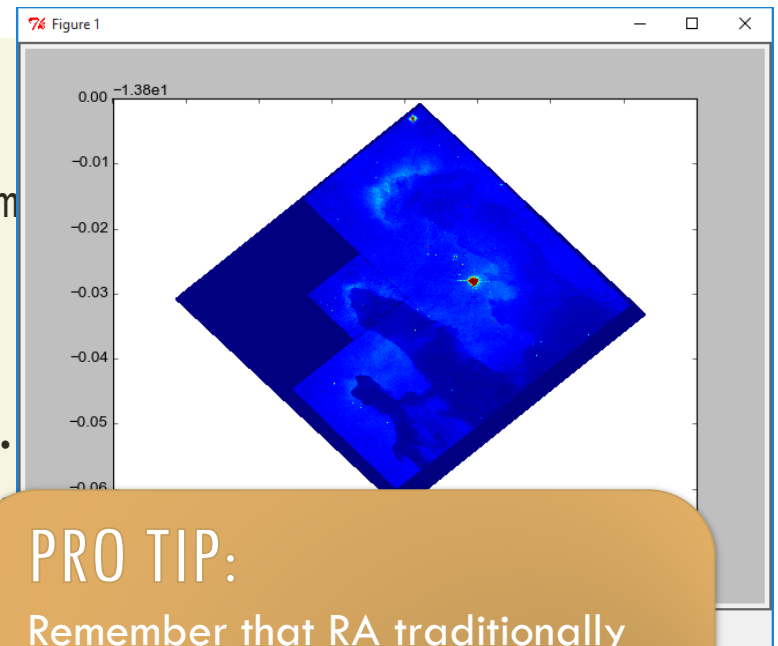
PLOTTING A FITS IMAGE

```
# Getting Data
imfile = fits.open(filename)
header, im = imfile[0].header, im
w = WCS(header)
```

```
# Making Indices
xpx = np.arange(im.shape[1]+1)-0.5
ypx = np.arange(im.shape[0]+1)-0.5
xlist, ylist = np.meshgrid(xpx,
raclist, declist = w.all_pix2world
```

```
# Plotting
plt.pcolormesh(raclist, declist,
```

Final Plot



PRO TIP:

Remember that RA traditionally increases to the left, so you'll have to flip the axis manually through `plt.xlim()`

COORDINATE TRANSFORMATIONS

Astropy provides a way of dealing with coordinates, and automatically deal with conversions:

```
from astropy.coordinates import SkyCoord

# Making Coordinates:
c1 = SkyCoord(ra, dec, frame='icrs', unit='deg')
c2 = SkyCoord(l, b, frame='galactic', unit='deg')
c3 = SkyCoord('00h12m30s', '+42d12m00s')

# Printing and Conversions:
c1.ra, c1.dec, c1.ra.hour, c2.ra.hms, c3.dec.dms
c2.fk5, c1.galactic # Converting Coordinates
c2.to_string('decimal'), c1.to_string('hmsdms')
```

ASTRONOMICAL UNITS

Astropy provides a way to manipulate quantities, automatically taking care of unit conversions automatically:

```
from astropy import units as u

# Defining Quantities with units:
val1, val2 = 30.2 * u.cm, 2.2E4 * u.s
val3 = val1/val2 # Will be units cm / s

# Converting Units
val3km = val3.to(u.km/u.s)

# Simplifying Units
val4 = (10.3 * u.s / (3 * u.Hz)).decompose()
```


ASTRONOMICAL CONSTANTS

Astropy also provides constants (with units):

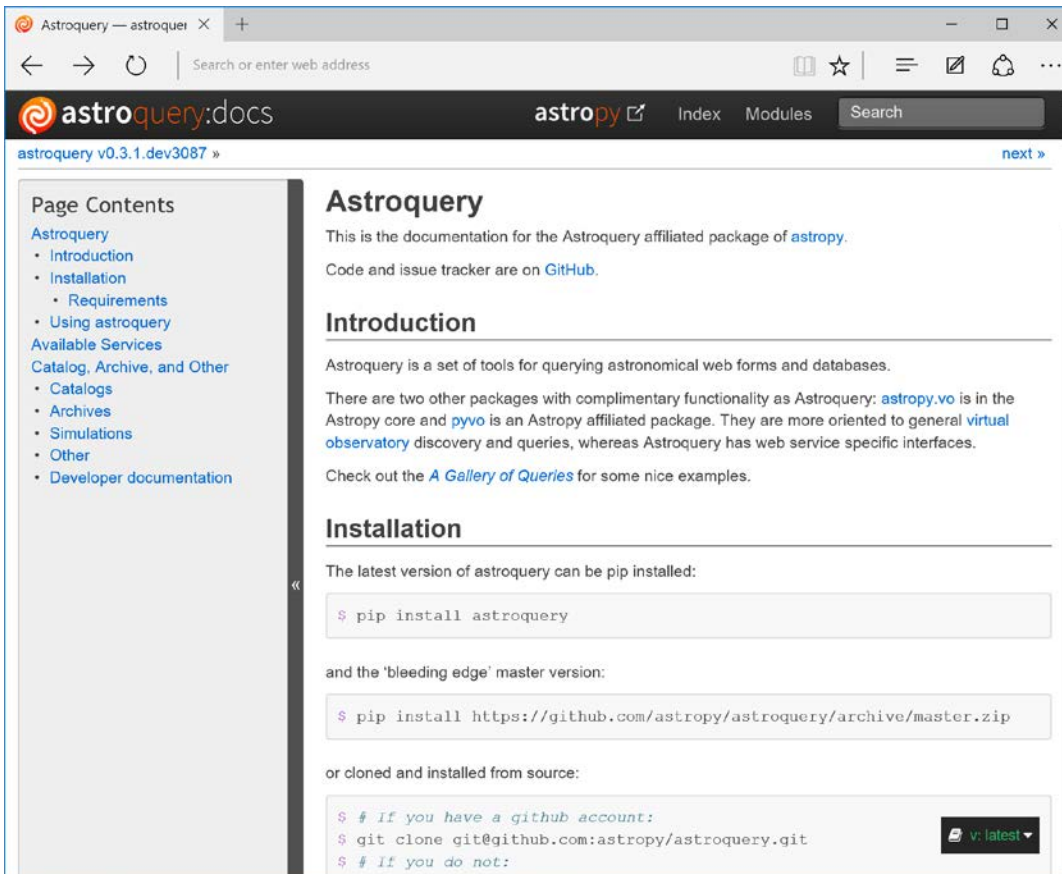
```
from astropy import constants as c

# Some constants
c.k_B, c.c, c.M_sun, c.L_sun

# Can use with units
energy = c.h * 30 * u.GHz

# Can convert units
mass = (3.2E13 * u.kg).to(c.M_sun)
```

ASTRONOMICAL QUERYING



The screenshot shows a web browser window displaying the Astroquery documentation. The browser's address bar shows the URL `astroquery.readthedocs.org/en/latest/`. The page header includes the Astroquery logo, the text "astroquery:docs", and navigation links for "Index" and "Modules". A search bar is also present. The main content area is titled "Astroquery" and includes a brief introduction, a link to the GitHub repository, and sections for "Introduction" and "Installation". The "Installation" section provides instructions on how to install the package using pip, including commands for the latest version and the 'bleeding edge' master version, as well as instructions for cloning and installing from source. A sidebar on the left contains a "Page Contents" section with links to various parts of the documentation, such as "Introduction", "Installation", "Requirements", "Using astroquery", "Available Services", "Catalog, Archive, and Other", "Catalogs", "Archives", "Simulations", "Other", and "Developer documentation".

astroquery v0.3.1.dev3087 »

Astroquery

This is the documentation for the Astroquery affiliated package of [astroquery](#).
Code and issue tracker are on [GitHub](#).

Introduction

Astroquery is a set of tools for querying astronomical web forms and databases.

There are two other packages with complimentary functionality as Astroquery: [astroquery.vo](#) is in the Astroquery core and [pyvo](#) is an Astroquery affiliated package. They are more oriented to general [virtual observatory](#) discovery and queries, whereas Astroquery has web service specific interfaces.

Check out the [A Gallery of Queries](#) for some nice examples.

Installation

The latest version of astroquery can be pip installed:

```
$ pip install astroquery
```

and the 'bleeding edge' master version:

```
$ pip install https://github.com/astroquery/astroquery/archive/master.zip
```

or cloned and installed from source:

```
$ # If you have a github account:  
$ git clone git@github.com:astroquery/astroquery.git  
$ # If you do not:
```

v: latest ▾

Astroquery allows access to online databases of various sources.

The documentation is located:

<http://astroquery.readthedocs.org/en/latest/>

ASTRONOMICAL QUERYING

There are lots of possible databases to query, but as a quick example (from Simbad):

```
from astroquery.simbad import Simbad

# Simbad
s = Simbad()

# Table of Matching Objects
tab1 = s.query_object('M31')

# Printing Table
tab1.pprint()
```

ASTRONOMICAL QUERYING

There are lots of possible databases to query, but as a quick example (from Simbad):

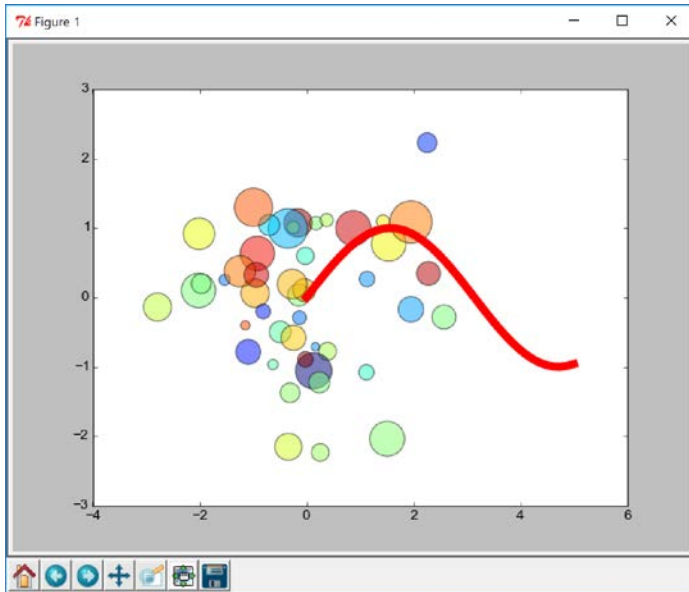
```
from astroquery.simbad import Simbad

# Simbad
s = Simbad()

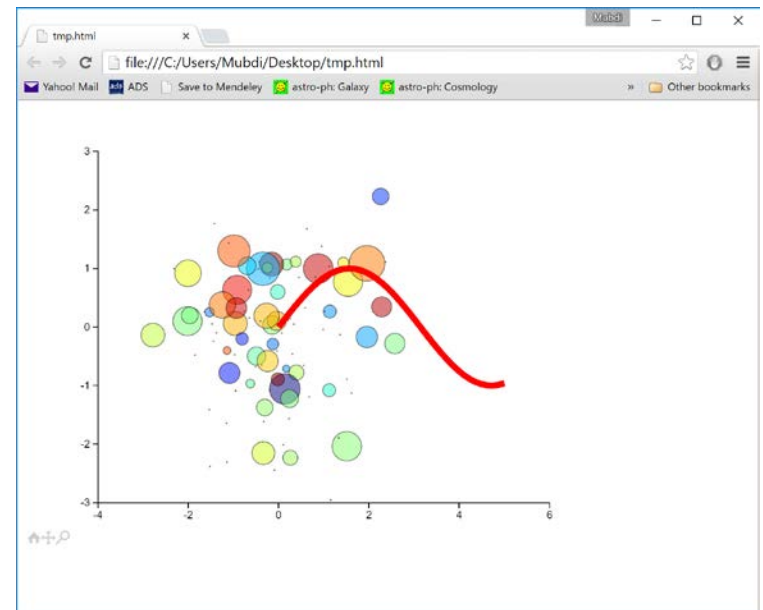
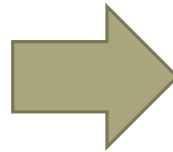
# Searching on Region
c1 = SkyCoord(298.4, -0.4, frame='galactic',
              unit='deg')
tab1 = s.query_region(c1, radius=1*u.deg)

# Printing Table
tab1.pprint()
```

MPLD3: MATPLOTLIB IN YOUR BROWSER



Matplotlib Figure



D3.js Webpage

MPLD3: MATPLOTLIB IN YOUR BROWSER

While not every matplotlib function is supported, it is easy to export your plot into an interactive HTML-based plot:

```
import mpld3

# If you have a figure already defined: fig1
mpld3.save_html(fig1, filename)

# Or if you do not have a variable for your figure
mpld3.save_html(plt.gcf(), filename)
```

PLAY TIME!

C is for cookie. That's good
enough for me.