Week 5

Astronomical Imaging in Python

2D arrays

- The astronomical images we deal with, at a basic level, are 2D arrays with each pixel representing a “photon bin.”
- Once you have extracted this 2D array from a FITS file, you can work with it in the way we talked about last week.

Indexing a 2D array

- When indexing a 2D array, we have a broader set of options for what we may want to index
- For example, we may want to pull an individual value, a whole row or column, a group of rows and columns, or select a small area of image.

Syntax

- The syntax for pulling a single element is the double bracket:
  pull = image_name[456][345] #In row then column format

- To pull a range of elements, we use comma notation:
  pull = image_name[456:500,323:350]

- We specify the row range with a colon, then the column range (separated by a comma). The above would pull a sub-array (2D) that was a rectangle on the original image

Syntax

- We can use our colon shortcuts to index whole rows/columns:
  pull_col = image_name[: , 350] #pulls the 350th column
  pull_row = image_name[40 , :] #pulls 40th row

- We can do the same to specify a range of columns or rows

- Exercise: create a 10x10 array using arange.reshape (length 100), and then pull the second, third, and forth columns. Then pull the 5th, 6th, and 7th row.

Image Transformations

- Sometimes we need to flip our images vertically, horizontally, or both.

- To flip an image vertically we can use either:
  image = image[: : -1] #or
  image = image[: : -1, :]

- The reasoning here is that you are setting the image equal to itself indexed in reversed order along the axis specified (the first version is just a short hand version of the second)

Image Transformations

- Similarly, if we want to reverse an image horizontally (left/right):
  image = image[: , : : -1]

- To do both, we can simply apply these transformations in one ([::-1,::-1]).

- The vertical (up down) flip is especially handy, because if you change matplotlib to put the origin in the lower corner, you need to flip your arrays for your image to plot in the same orientation as before.

- Try flipping your 10x10 array from earlier both vertically and horizontally to see what it looks like.

Pyfits and Astropy

- These are the plotting libraries that allow us to import fits images into our code.
- You’ll need to try to get these installed before Wednesday… on a mac this is trivial but on a PC it might take a bit of work.
- If you really have trouble, we have ways of getting you the array from a fits file raw, so you will have a workaround

Mac/Linux Users

- Try opening terminal and typing “pip pyfits”
- People with canopy should have pip installed, and everything will just work automatically.
- If it says you don’t have pip installed, https://pip.pypa.io/en/latest/installing.html <— go there, download the get-pip.py file, and run it in terminal. pip will install, then you can pip pyfits to install it.

Windows

- Go to http://www.stsci.edu/institute/software_hardware/pyfits/Download and download the version for windows and python 2.7.
- If I’m not mistaken you can open canopy’s terminal (from the bar at the top), and cd to the directory where the file above downloaded, and then type: >> python setup.py install
- Try before Wednesday. If you have trouble, come to tutorial a bit early and we’ll try to work it out. If not, no worries, we will get you the raw data.

What is FITS?

- Stands for Flexible Image Transfer System, and is the preferred "image" format amongst astronomers and physicists.

- I say "image" because a FITS file contains more than just an image, the way a jpeg or png does.

- Once a FITS file is loaded into python, you have access (via dot notation) to a ton of different things called attributes. The image itself is stored in the "data" attribute. The other primarily used attribute is the "header".

Loading a FITS file

- Import pyfits as pf
- hdu = pf.open('path/filename') #Don't need full path if in same directory
- This command opens the fits file and stores everything in the variable hdu (you can call it whatever- HDU stands for header data units)
- Now all the attributes are stored via dot notation in hdu

Accessing the header/image

- We can now set variables for the header and image:
  ```
  head = hdu[0].header
  image = hdu[0].data
  ```

- The reason we index HDU at 0 first is because some actually have multiple images/data stored, so we generally want to make sure we are getting the first. (Just a safety measure).

The Header

- The header is a dictionary with a lot of useful information about the image you are working with, such as the time and date of the exposure, the exposure length, RA/DEC (where the telescope was pointed in the sky), what kind of filter was being used, etc.

- It makes sense to store these as a dictionary, not an array, because we want to be able to index head['RA'] or head['DEC'] rather than some arbitrary index (somehow knowing RA is the 50th element, etc).

- Note- dictionaries are NOT case sensitive, so head['dec] == head['DEC']

The Image

- We now have a variable called image with the raw 2D array. We can immediately plot it using a new plotting command called imshow:

  ```
  plt.imshow(image, origin=‘lower’,cmap=‘gray_r’)
  ```

- image was the only needed argument- the origin command moves the origin to the right corner, and c map choses a color palate (in this case reversed gray). There are a few other commands we will work with to make our plotted images come out with the right contrast.

Manipulation

- Our image is just a 2D array, so we can flip it, index it, pull values from it for analysis, etc., the same way we would for a 2D array!
- For example since I used origin=‘lower’, I first need to set image = image[: : -1]
- Example!