Tutorial Five: Images in Python

Imad Pasha
Chris Agostino

March 4, 2015

1 Introduction

Astronomical images are not typically taken in the familiar jpeg or png format. The preferred file-format currently used in astronomy is known as FITS (or Flexible Image Transfer System). The FITS format lends itself well to astronomy, because it allows for multiple images to be stored (if needed), and for there to be a header file with a lot of relevant information about the image also stored within the single FITS file. If you do the Upper Division labs or research in the Astro and Physics departments, you will likely end up working with FITS files. If you don't, this will be good practice on what we learned about multidimensional arrays.

Our goal today is to have python "locate" the positions of the stars in an astronomical fits image (like in fig. 1).

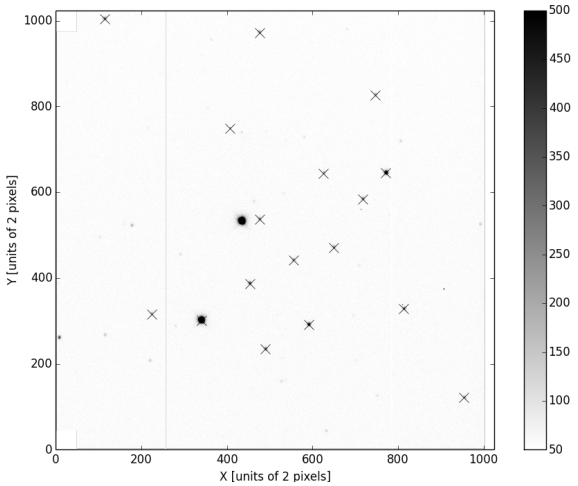

Figure 1: A starfield taken using the 1m telescope at Lick observatory, with star positions marked.

2 Loading Fits Images

In order to work with astronomical images we need to load them into python, which is sightly more complicated than loading a data file. If you are working on your own personal computer, you will need to install the package pyfits. On a mac, this is trivial. Open up your terminal on Mac and type pip install pyfits. If you are on windows... come talk to us, or use the lab computers. If you are working on the lab computers, the package/library you will use is astropy.io.fits (it does the same thing).

2.1 The Header

Open up a new python document in your tutorials directory. Import numpy, matplotlib.pyplot, and the appropriate fits library as pf. To load a fits file, add the line:

hdu = pf.open('filename')

assuming, of course, that the file is in the directory you have the fits file saved (if not include the full path location of the file). The name hdu can be anything. Once the file is open, we want to extract the information we need from the fits file before closing it.

1. set a variable header = hdu[0].header
    - we typically index hdu at 0 in case there are multiple images stored within one fits file. The various data (like headers, data, etc) in a fits file are stored in what are called "attributes" which are accessed with dot notation as above. To see the full list of attributes, you could open ipython, load a fits file using the method above, and then type hdu.(tab key) .
2. set a variable image = hdu[0].data
    - the actual "image" is stored in the "data" attribute. Your variable 'image' is now a two dimensional array of numbers, representing the intensity at that pixel (how many photons hit that ccd pixel during the exposure). Typically, a ccd pixel becomes saturated (maxed out) at around 65,000 ADU (analog to digital units). So the intensity values you see in your image will range between 0 and 65,000.

In order to get information out of the header file, we need to index it, the way we do lists an arrays. However, there's a long list (up to 50) things stored in a header, and it would be almost impossible to remember that the 32nd index was for exposure time, the 6th for filter, etc. (plus it would be different for every telescope). The way around this is to use dictionaries instead of lists/arrays, and a header file is indeed a dictionary. We mentioned dictionaries briefly early in the semester but haven't gotten much practice with them.

Essentially, dictionaries are like lists, but for each entry in the dictionary, you specify a 'key' for it (rather than rely on the position of the element in the dictionary). To define your own dictionary, you use curly bracket notation as follows:

my_dictionary = {'dog':'bill', 'cat', 'whiskers', 'my son':'sam', 'num_pets':2}

Now, you can index your dictionary by typing something like my_dictionary['num_pets'] and the code would return the integer 2. The notation within the brackets is keys and values

separated by colons (keys and values can be strings, numbers, etc), and key/value pairs separated by commas.

If you are in ipython and you have an hdu loaded, you can print hdu[0].header to see the full header, and learn what types of things you can index it to find. (Note, dictionaries are NOT case sensitive, so indexing dict['RA'] is identical to dict['ra']).

When we plot our image, we want to label it with the RA and DEC (basically, the sky coordinates) of the star field. To do that we need to extract that information from the header.

1. In your code, create two variable, ra and dec, and set them by indexing the header. (The keys are just the strings 'ra' and 'dec').

2.2 The Image

We already have a variable named image which is storing the 2d array we will be working with. To plot it, we use the plt.imshow command, with a bunch of specifications:

plt.imshow(img, cmap='gray_r', origin='lower',interpolation='none, vmin=np.median(img), vmax=np.median(img)*1.28)

- The first argument is the variable name of the 2d array we want to plot.
- the cmap refers to a colormap or gradient. There are many options (you can find them in the documentation or online), here we choose "reversed gray", which is basically a gradient from black to white, with white being low intensity (few photons) and black being high photon counts. The reason for the reversal (the real night sky has white as high intensity and black as low) is that a) contrast is easier to see with black on white, and b) when submitting papers for publication, black on white saves a lot of ink.
- The third argument sets the origin of the image to the lower left hand corner, as is traditional for images. Matplotlib, if you dont set this, uses matrix notation, and picks the top left corner as the origin.
- The vmin and vmax commands set the range of the linear gradient scale. Basically, its setting in what range there will be a scaled difference between black and white. Anything above vmax is the same black as vmax, and anything below min is the same white as vmin. We usually set a small range because we want to maximize the contrast in our images.

You should now see an image like in fig. 1, minus the X's. If the contrast looks off, you can try replotting with changing the 1.28 to other numbers and see how it looks. You should know from the textbook how to slice this 2d array any way you like. If not, definitely check that out, because you'll need it in a second.

3 Finding the Stars

We now want to be able to iterate over our array and locate the positions of all the stars. The "proper" way of doing this is a two dimensional centroiding technique, which you will learn in A120 if you take it. We will be using a more approximate method.

3.1 Preparing the Image

Preparing the image. As you will quickly learn working with any astronomical image, a lot of the CCD detectors on telescopes kind of suck. Particularly an old one like that on the 1m at Lick. Because of this, there are some rows, columns and zones of "dead" or "saturated" pixels in the image (see the black vertical lines?). Now, because these pixels are saturated (basically maxed out), we DON'T want to include them in our search for the stars, because if we look for the "bright" objects in our image, every pixel in these columns will light up.

1. I'm going to tell you the zones of the image you are working with that need to be accounted for. The quickest and dirtiest (but usually sufficient) way to do so is to simply set the image indexed in those zones to be 0- then it won't show up as the brightest.

2. Of course, if we go around deleting parts of our image, we want to have a hard copy lying around in case we want to plot the original. Underneath where you define your image from hdu[0].data, make something called image_2 = np.copy(image). This will ensure that you have the original image floating around, even if we start zeroing our parts of the "working" image.

3. Here are the zones in which you want to set all values to 0: columns 254-257, the columns from 998 to the end, the first second and third column (remember that means index 0, 1, 2), and the first (index 0th) row of the image. Index the image to pull those columns or rows and set them to 0. (Don't worry, we aren't really deleting any stars, and in any case we usually don't need to pull *all* the stars anyway.

3.2 Finding the peaks

Now that all the artificial things that could've caused problems are 0, we can focus on finding the stars. To find the "brightest" star is fairly easy- we just need the coordinates of the brightest pixel, i.e. the one with the highest numerical value (using the argmax function). Your goal is to write a script that will find the coordinates of the maximum value, append the x and y values of it to some lists/arrays, then find the next maximum value, and so on and so forth. You need to be careful though, because the next brightest pixel might be within the star you just "found". The good news is, all of the stars in this image are localized easily to within 20 pixels. The method I used to find the stars was a while loop in which I "found" a star using argmax, then set everything in a 20 pixel vicinity of that coordinate to 0. I had a variable called "found" outside the loop that I initialized at 0 and added to every time I found a star, and had my while loop continue as long as found was lower than, say, 25 stars.

Note: remember you can use ctrl-c or the canopy toolbar to stop your code if it's hanging in the while loop for some reason. Note: When you use np.argmax() on a 2d array, you get back a single number that represents the flattened index of the max value. To convert this back into row/col notation, you can use np.unravel_index(index, img.shape) (where index is the single index returned by argmax). This will return a tuple, (y,x) which you can index (for example, say x_val = outputofargmax[1]).

Once you have all the coordinates, you can use plt.plot to plot them as x's on top of your plt image (remember to plot the copy you made so the stars are actually there!), and they should line up with all the stars!

If your plot does the annoying thing where it is plotting in a region larger than the image (with white space around it), you can use plt.xlim((0,1000)) and plt.ylim((0,1000)) to manually set the axis limits to 0 and 1000, the primary dimensions of the actual image.

Last but not least, look up how to include the values of variable in your matplotlib titles, and title the plot youve made, with the RA and DEC of the image included.

4 Homework

For homework this week, turn in a zip of the final image you made of the starfield with overlaid x's, as well as the code you used to make it (comment!!). We'll go easy and not assign separate homework this week- you're welcome.