Mohammad Ashas

Python Tutorial: Comparing Data from Different Catalogues Using Sky Coordinates

Abstract

Different catalogs of observational data with overlapping observation can be compared to check how accurate the photometry is. To be able to compare the photometry, one must first cross-match the catalogs by determining whether the right ascension and declination of the observed objects are within a certain match radius in both catalogs. Finding these spatially coincident matches suggests that the observed objects are the same physical sources in both catalogs. This can be done quickly using Sky Coordinates and a for-loop in Python. In this tutorial, we compare 3D-HST GOODS-South field catalog to the Hubble Space Telescope Legacy (HLF) catalog[1].

Introduction

The goal of this tutorial is to show how to use Sky Coordinates and for-loops to compare flux data from two different catalogs, also introducing some of the basics of python in the process. Flux is the amount of light that falls on an area of the telescope lens. This data can be used to calculate the magnitude of celestial objects in the universe. By comparing the magnitude of celestial objects from different data sets we can confirm the location of objects in the night sky.

Sky Coordinates is a python utility which can be imported to make it easier to compare on-sky objects positions from different catalogs through the right ascension and declination. The right ascension and declination can be thought of like the x and y points of a graph but in terms of the night sky. Sky coordinates can use several different coordinate reference frames but in this tutorial we will be using International Celestial Reference System (ICRS) which is used by International Astronomical Union. By matching right ascension and declination from different catalogs but the same filter (same wavelength of light being transmitted) it is possible to determine how viable the data is.

The for-loop part of this tutorial is actually just a method used to shorten the amount of code needed to complete a task by instructing python to repeat the same lines of code with respect to a variable. In this case, the task is comparing the two catalogs with several different filters at the same time instead of going through each of the filters separately and the changing variable will be the filters. The for-loop will be used to formulate the magnitudes and the difference between magnitudes observed by different filters, organizing the results into easily understood scatter plots. The scatter plots will allow for the visualization of catalog data and confirmation within a range of error the position of stars and galaxies in the night sky.

Data

The files needed to complete this tutorial can be found on the following links:

http://3dhst.research.yale.edu/Data.php (Under the Photometry tab)
https://www.dropbox.com/sh/bwkqzmbfx9oc10h/AADYa7YjEzUW0575vTuCmYdDa?dl=0

Method

1. To begin comparing catalogs you must first import in a couple of utilities from astropy and matplotlib. This will allow python to perform certain functions like make graphs and organize data into tables. The utilities from astropy.table allow for data to be organized into tables and columns. The utilities from matplotlib.pyplot enables python to make plots and graphs as well as perform functions that would enhance the atheistic appeal of the plots and make the data more understandable. The numpy utility can be used to organize into arrays which can then be manipulated and used to calculate a wide variety of functions, including mathematical operations such as running mean and median. (An array is a data structure that stores values of same data type). The ascii import from astropy.io allows python to read in data from ascii files (which is mentioned later on). ***Note:*** *I always like to import these utilities whenever starting a new python document. It is a good starting point and generally comes in handy.*

```
In [1]: from astropy.table import Table, Column, join
        from astropy.table import Column
        from numpy import *
        import matplotlib.pyplot as plt
        %matplotlib inline
        import numpy as np
        from astropy.io import ascii
```

2. Now to let python know where the catalogs we want to use are we must define a variable describing the full path location. ***Note:*** *The easiest way to create a variable like this would be to save the file you want to access on the desktop that way all you have to input is catalog_path = "desktop/name of file/" but python can access the file anywhere as long as the input is correct. Also the Prelim Catalog refers the HLF files.*

```
In [2]: catalog_path="desktop/3DHST/"
```

```
In [3]: catalog_path2="desktop/Prelim Catalog/"
```

3. To be able to access the data from the catalog we must read in the specific file we want to analyze into python. ***Note****: It is good to realize that there are a quite a few different types of files and slightly different ways of reading them in. In this case we have an ascii file so in the code there is a ascii.read(….), However, in the case of .fits files you would want to use Table. Read(…). Also you can see now why we imported ascii in the first code input. In general using the ascii and fits method is a good first step in trying to read in a catalog. However, files can be named very differently depending on who is creating them. In this case, we have .cat and .nzpcat file but these are not standard names in any way.*

```
In [4]: HST = ascii.read(catalog_path+'goodss_3dhst.v4.1.cat',
        data_start=0,header_start=0,delimiter=' ')
```

```
In [5]: Prelim_Catalog = ascii.read(catalog_path2+'hlsp_hlf_hst_60mas_goodss.v1.0(2).nzpcat',
        data_start=0,header_start=0,delimiter=' ')
```

4. To check if the files were read in properly, input print (filename). ***Note****: It is always a good idea to make sure the file was read in properly, also this is a good way to get a visual of how the catalogs are structured.*

```
In [6]: print HST
```

```
  id      x         y         ra     ... nexp_f125w nexp_f140w nexp_f160w
----- --------- --------- --------- ... ---------- ---------- ----------
    1 11876.639   1632.89 53.093012 ...        4.0        0.0        4.0
    2 12056.715  1321.055 53.089613 ...        2.0        0.0        1.0
    3 11351.875  1327.244 53.102913 ...        1.0        0.0        1.0
    4 11415.681  1396.836 53.101709 ...        2.0        0.0        2.0
    5  11385.57  1384.729 53.102277 ...        2.0        0.0        2.0
    6 12005.665  1394.706 53.090577 ...        2.0        0.0        2.0
    7 11508.163  1409.004 53.099964 ...        2.0        0.0        2.0
    8 11710.638  1450.711 53.096144 ...        2.0        0.0        2.0
    9 11952.919  1403.613 53.091572 ...        2.0        0.0        2.0
   10 11938.062  1414.769 53.091852 ...        2.0        0.0        2.0
  ...       ...       ...       ... ...        ...        ...        ...
50497  8889.778 18988.826 53.149298 ...        3.0        0.0        3.0
50498  7596.216 19253.178 53.173638 ...        5.0        0.0        5.0
50499  7849.441 18896.686 53.168876 ...        6.0        0.0        6.0
50500  2944.198 18924.184 53.261183 ...        4.0        0.0        7.0
50501  8601.418 18808.895 53.154725 ...        6.0        4.0        6.0
50502  7876.559  19063.41 53.168364 ...        6.0        0.0        6.0
50503  3207.811 18767.998 53.256225 ...       24.0        4.0       26.0
50504  3319.077 18889.404 53.254129 ...       24.0        4.0       26.0
50505  7634.091 18915.908 53.172928 ...        6.0        0.0        6.0
50506  8669.859   18840.1 53.153437 ...        6.0        0.0        6.0
50507  3041.903  18822.67 53.259346 ...       24.0        4.0       26.0
Length = 50507 rows
```

```
In [7]: print Prelim_Catalog
```

```
   id      x       y         ra      ... flux_radius fwhm_image flags use
------- ------- ------- ----------- ... ----------- ---------- ----- ---
    1.0 14060.9  3510.6  53.0933075 ...       -99.0      -99.0    19   0
    2.0 14223.0  3258.3 53.09024811 ...       -99.0      -99.0     1   0
    3.0 14257.9  3347.6 53.08959198 ...       -99.0      -99.0     3   0
    4.0 14288.4  3438.2 53.08901596 ...       -99.0      -99.0     3   0
    5.0 13615.7  3298.2 53.10170746 ...       -99.0      -99.0     0   0
    6.0 13708.0  3309.3 53.09996796 ...       -99.0      -99.0     0   0
    7.0 14210.8  3266.1  53.0904808 ...       -99.0      -99.0     1   0
    8.0 13910.8  3351.0 53.09614182 ...       -99.0      -99.0     3   0
    9.0 13585.0  3286.9 53.10228729 ...       -99.0      -99.0     0   0
   10.0 14204.8  3295.2 53.09059143 ...       -99.0      -99.0     0   0
    ...    ...     ...         ... ...         ...        ...   ... ...
47968.0  9962.3 21363.2 53.17051315 ...       -99.0      -99.0     0   0
47969.0  9983.9 21377.9 53.17010498 ...       -99.0      -99.0     0   0
47970.0  9681.0 21381.2 53.17580414 ...       -99.0      -99.0     0   0
47971.0  9742.7 21406.5 53.17464447 ...       -99.0      -99.0     0   0
47972.0  9873.4 21394.9 53.17218399 ...       -99.0      -99.0     0   0
47973.0  9717.7 21441.1 53.17511368 ...       -99.0      -99.0     1   0
47974.0  9736.5 21440.0 53.17475891 ...       -99.0      -99.0     3   0
47975.0  9786.6 21440.9 53.17381668 ...       -99.0      -99.0     0   0
47976.0  9772.6 21441.6  53.1740799 ...       -99.0      -99.0     0   0
47977.0  9727.7 21455.3 53.17492294 ...       -99.0      -99.0     3   0
47978.0  9705.0 21456.5 53.17535019 ...       -99.0      -99.0     1   0
Length = 47978 rows
```

5. Just like before we need to import more utilities from astropy so python will be able to run the code we will input later on. The utilities being imported allow us to use sky coordinates and some of its specific features. ***Note:*** *I am importing these utilities now instead of before as a way to show some utilities like the ones above are used a lot more generally than these utilities which are for our specific task.*

```
In [8]: from astropy.coordinates import SkyCoord  # High-level coordinates
        from astropy.coordinates import ICRS, Galactic, FK4, FK5  # Low-level frames
        from astropy.coordinates import Angle, Latitude, Longitude  # Angles import astropy.units as u
        from astropy import units as u
```

At this point in the code all of the setting up for actually comparing the two catalogs is done. The catalogs have been read in and all the necessary utilities have been imported.

How to Cross-Match Photometry in a Single Filter:

6. We are finally at the point where we are using sky coordinates program to compare the two catalogs. The basic idea is that we are trying to find stars and galaxies that are in the same on-sky location by checking if their right ascension and declination is the same in the two catalogs (basically x and y position). If a star or galaxy is found in both catalogs with the same or similar right ascension and declination it could imply that it is the same object. In the following code ra is right ascension and dec is the declination. c1 and c2 are the name of the catalogs that are being compared. The keyword frame just tells the program what coordinate system python should use. In our case the coordinate system is International Celestial Reference System. The idx are indices into c2 that are the closet objects in coordinates in c1. The d2d is the on sky distance between the two objects and the d3d is the distance between the object in 3 dimensions. The 3 dimensions being the right ascension, declination, and the depth (redshift). The coordinate system we use in this step must match the coordinate system of the input catalog. The last line of code tells python which catalog is matched to the comparison catalog. In our case, catalog 2 (HST) is matched to catalog 1 (Prelim_Catalog). Basically, python is going through every single ra and dec in catalog 1 and finding the radial distance to the closest object. ***Note:*** *In this case there is no data for the third dimension however the d3d does not affect the resulting matched catalogs and it seems to be an error if the d3d is not included so we just keep it. This could just be how the imported sky coordinates utility works.*

```
In [9]: ra1 = HST['ra']
        dec1 = HST['dec']
        ra2 = Prelim_Catalog['ra']
        dec2 = Prelim_Catalog['dec']
        c1 = SkyCoord(ra1, dec1, frame='icrs', unit='deg')
        c2 = SkyCoord(ra2, dec2, frame='icrs', unit='deg')
        idx,d2d,d3d = c2.match_to_catalog_sky(c1)
```

7. In this step, we need to put some restrictions on the code so that if there are actually any matches between the catalogs we can be sure they have significance. For example, the radius tells python how close the ra and dec must be to be considered a match in units of degrees. match_index[selection] = -99 makes it that so every ra and dec not within the match radius is equal to -99. ***Note:*** *selection is just a restriction we can make. In our case, the selection dictates that the d2d (distance to the closest object) is greater than the radius defined before it. Writing in match_index[selection] just applies the selection defined before to the match index.*

```
In [10]: radius = 0.4/3600.
         selection = (d2d > radius*u.deg)
         match_index = idx
         match_index[selection] = -99.
         s2 = (match_index >= 0)
```

8. In this step, we are testing whether or not our sky coordinate code is working properly with only one filter as a test. The filter which is being used the f_f160w and is being labeled as j so it can be automatically replaced throughout the formulas below. The overall goal of this test is to calculate the del_mag and the mag (delta magnitude and magnitude). The magnitude is the brightness of the objects with respect to the flux recorded by the telescope used during observation. In this case, the mag calculated in the last line of code will be of the Prelim_Catalog filter. The delta magnitude is the magnitude of the Prelim_Catalog filter minus the magnitude of the same filter of the HST catalog. The del_mag code is taking objects with very similar ra and dec from both catalogs and subtracting their magnitudes, the closer the del_mag is to zero the higher the chance of the objects in the two catalogs are actually the same. ind and ind2 are selections to make sure we eliminate bad data and don't get any undesirable results. ind removes all -99 place holders in the calculation and ind2 removes the negative flux which would cause an error if you try to take the logarithm of a negative value. The match_index is placed on the HST side to tell where in the original catalog the corresponding physical object is located, eliminate unmatched objects and also making the sizes of the columns the same. It is not placed on the Prelim_Catalog side because the HST catalog is being matched to the Prelim_Catalog. ***Note:*** *the selection has to be put in the shown order or it won't work properly. Ind is part of ind2 so python has to read in first. Also you might be wondering what is the -2.5*log10(flux) is for. This is to convert the data from each filter to the AB magnitude scale in this case. In fact, there should also be a (+25) constant on the end of the formula but it cancels out because of the negative sign in the formula so to make everything a little simpler it is excluded. Different catalogs might need to be converted into the AB magnitude scale using different constants (called a "zero point") at the end, so they might not cancel out. Forgetting the constant might cause slight errors that can be easily overlooked.*

```
In [11]: j = 'f_f160w'
         ind = ((match_index >= 0)) # removing -99 indices
         ind2 = ((HST[j][match_index][ind] > 0) & (Prelim_Catalog[j][ind] > 0)) # removing negative flux or -99 flux
         del_mag = -2.5*log10(Prelim_Catalog[j][ind][ind2])+2.5*log10(HST[j][match_index][ind][ind2])
         mag = -2.5*log10(Prelim_Catalog[j][ind][ind2])+25.
```

9. To make sure the code worked we can print mag and del_mag. ***Note****: This is not necessary but it is always good check to make sure the data is looking good. You don't want to find out a lot later that the formula did not give the proper results. The typical range of magnitudes is 15-28, and the typical offsets between the magnitudes between the two catalogs is at least within +/- 1 magnitude (corresponding to a factor of 2.5 difference in the respective fluxes).*

```
In [12]: print mag
      f_f160w
-------------
23.3089315648
24.8437246003
19.7929540255
25.0602589604
25.7169950459
23.4491561666
 26.721058736
24.8127295289
24.1892501244
26.4341655845
          ...
21.8051974621
23.5906854716
24.6042551342
26.8314675294
25.7998228331
25.3168155055
26.6232778129
26.0753826811
25.9021549564
25.6931026782
24.8021672273
Length = 34959 rows
```

```
In [13]: print del_mag
       f_f160w
---------------
 -0.136471039832
-0.0697864684585
-0.0262703135069
   0.04903872952
  0.190881609701
 0.0475849802489
  0.159577569303
 -0.051025104552
 0.0473687531786
 -0.458586784952
                ...
 0.0179507884346
-0.0378659430901
   0.10290049246
  0.251231106991
  0.385214683846
  0.391216189424
  0.261973601341
 0.0550884681069
 -0.102724591412
  0.361987934245
  0.023468675508
Length = 34959 rows
```

How to Cross-Match in Multiple Filters:

10. In this step, we create a list of all of the Hubble Space Telescope filters that are available in both catalogs and create a for-loop to calculate the del_mag and mag for all of the seven filters. The code in the for-loop (under "*for a, i in enumerate(list):")* is exactly the same as the test code we made in step 8. Under that code, we next make a subplot graph of every filter in the for-loop. This is done to give a visual representation of the data. It is a lot easier to interpret the data with the use of graphs. "*sub.1 = fig.add_subplot (4,2, a+1)"* is the line that actually sets up the graphs. The (4,2) tells python that subplot is 4 rows x 2 column panels and the a+1 is what tells python to change the location of where the data is being displayed for each sub panel. The rest of the code is telling python what type of graph we want, the limits on the x and y axis and the axis labels. The for-loop may seem very complex but it can be broken down to fairly simple components. For example, the "*for a,i emumerate (list*):" is what tells python that a for-loop is beginning and anything indented after this is point is part of the for-loop. The a and i are the changing component. Throughout the code you see [i]. This loops through the values of the 'list'. In this case the 'list' consists of the seven filters. Instead of writing the same code seven different times the for-loop can run the same code seven times just changing the [i] to a different filter name provided in the list. The [a] in the code is used to index the list so the list array is easier to manipulate. Indexing is a method of marking the position of the components of an array, in this case the array is the list. The array's components are "zero-indexed", which means they are numbered from 0 to one less than the total number of components in the array. In the case of this

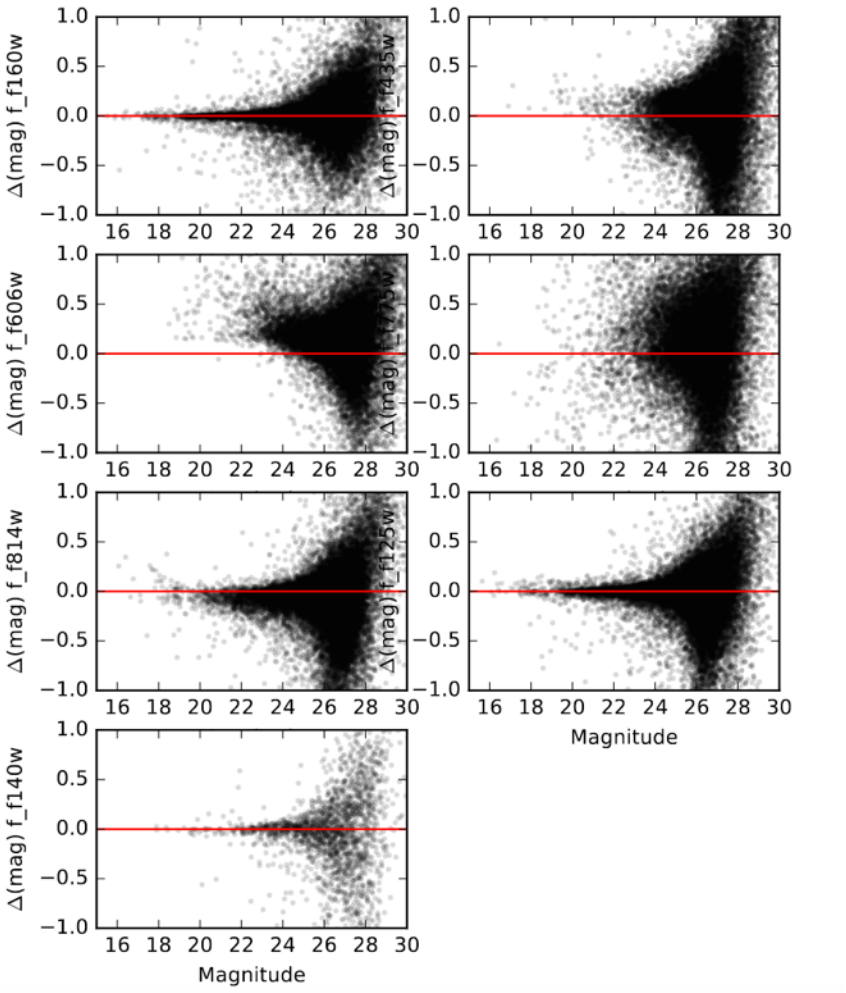
for-loop, the [a] does something very similar to the [i] but instead of changing the filter name it is used to change the sub panel where the data will be displayed. This is done by changing the assigned position number of each of the filters and matching it with the equivalent sub panel number. The enumerate (list) is telling python to look to the list = […] for the changing [a] and [i]. ***Note:*** *we assume here that the filter names must be exactly the same in both catalogs. Ind2 is in the for-loop but ind is placed outside the for-loop. This is because ind2 has to be applied to all of various filters but it does not matter whether or not ind is applied to all the filters because it is only setting up the match index, which will rule out all data that doesn't fall under the match radius. a is in reference to the plot sub panels, so when plotting a graph like ours we must use "a+1" instead of just "a" because in python "a" starts off from 0 and not 1. If we had put "a" instead of "a+1", python wouldn't be able to understand what a=0 is and it would get stuck at the first filter.*

All the work done until now has been to plot the seven graphs below. As you can probably already tell from the code, the x-axis is the magnitude of the Prelim_Catalog filter which is used as reference and the y-axis is the difference in magnitudes of the two catalogs in the respective filter. Each point on the graph is either a star or a galaxy emitting light. The brightest objects have lower magnitudes. The delta magnitude being closer to zero suggests that the magnitude of the object in the match radius of both catalogs were the same or very close to each other. This not only confirms that the observed star or galaxy is the same in both the catalogs, but that the independently measured fluxes are in good agreement. Stars and galaxies that are dimmer (larger magnitudes) have a larger scatter in their delta magnitudes. This does not necessarily suggest that we have mis-matched these objects, but rather when we measure the fluxes of very faint objects close to the noise limits of the image there will be a larger measurement uncertainty. This makes sense if you think about it. The dimmer the stars and galaxies are, the harder it would be for the telescope to accurately measure the flux, which could lead to small errors and differences between the catalogs due to observation at different time and positions, leading to a larger delta magnitude.

In [14]:
```
list = ['f_f160w','f_f435w','f_f606w','f_f775w','f_f814w','f_f125w','f_f140w']
fig = plt.figure(figsize=(6,8))

for a,i in enumerate(list):
    ind2 = ((HST[i][match_index][ind] > 0) & (Prelim_Catalog[i][ind] > 0)) # removing negative flux or -99 flux
    del_mag = -2.5*log10(Prelim_Catalog[i][ind][ind2])+2.5*log10(HST[i][match_index][ind][ind2])
    mag = -2.5*log10(Prelim_Catalog[i][ind][ind2])+25.
    x = mag
    y = del_mag
    sub1 = fig.add_subplot(4,2,a+1)
    sub1.scatter(x,y,marker='o',c='black',alpha=0.1,s=3)
    sub1.plot([15,30],[0,0],'red')
    sub1.set_xlim([15,30])
    sub1.set_ylim([-1,1])
    sub1.set_xlabel('Magnitude')
    sub1.set_ylabel('$\Delta$(mag) '+i)

fig.savefig('mag_delmag.pdf')
```



Conclusion

To reiterate, the validity of any two catalogs of observational data like the 3D-HST GOODS-S and HLF catalogs can be checked by figuring out if there are any observed stars or galaxies with the same right ascension and declination within a certain pre-determined match radius over a variety of different filters. This process allows us to strengthen the validity of the photometry of the catalogs by confirming the position of celestial objects in the night sky leading to more accurate the trustable results when being used in different experiments.

References

[1]https://archive.stsci.edu/prepds/hlf/

http://matplotlib.org/users/pyplot_tutorial.html

https://www.tutorialspoint.com/python/python_for_loop.htm

https://wiki.python.org/moin/ForLoop