Performing Image Subtraction and Computing the Light Curve of a Source

Northwestern University

Nicolas Guerra
Student of Professor Mel Ulmer
Summer of 2021

Contents

1 Introduction

The purpose of this report is to document how to perform image subtraction and compute the light curve of sources. Such a procedure allows one to discover variable sources and classify these sources as possibly transient phenomena. A transient phenomenon is an astronomical event that can range from a few seconds to a few months. The light curve of a transient source typically has a sharp increase in brightness followed by a gradual decay. This shape of the light curve differs from the usual variability seen in stars which is sinusoidal.

2 Image Subtraction

Image subtraction is the process of taking two images of the same space at different points in time and subtracting one from the other. While the general concept is easy to understand, there is a lot of preparation that needs to occur before doing so. Image subtraction allows one to see any changes from one image to the next. Ideally, if there are no changes, such as an increase or decrease in flux of a star, the difference image should be a zero matrix with noise and should look like a solid color. If there are any changes, one should see point sources at the locations of variability. If the changes in the difference image are not point-like, this is a good indication that the image subtraction was not as optimal as it could be, and parameters should be adjusted. This section documents how to perform image subtraction and how to code it in Python. For more information on image subtraction, great videos on the topic can be found on GROWTH Project's YouTube channel[1]. The paper "Optimal Image Subtraction"[2] is also a good reference that dives deep into the math.

2.1 Downloading Data

Hubble fits files are the main type of data that this report was written for. However, The basic principles should still apply to other telescopes as well. For Hubble, there are several ways to obtain data in Python.

2.1.1 Wget

If one happens to have the link to the fits file whereby just clicking it is downloaded, one should use *wget*. To install, go to terminal and enter *pip install wget*. Here is an example on how to use it.

```
from wget import download

link = 'https://mast.stsci.edu/api/v0.1/Download/file?uri=https://hla.stsci.edu/cgi-bin/
    getdata.cgi?dataset=hst_10861_18_acs_wfc_f475w_drz.fits'
fits_name = download(link)
```

2.1.2 ESA Hubble Science Archive

The HSA website[3] allows one to search for a target name or specific coordinates. One is also directed to this page from the Hubble Catalog of Variables Explorer website[4] if one were to click on a data point on a light curve and click the HSA icon. On the HSA website, click on the download icon and a tar file will be downloaded. One can also copy the link address of the download icon and use *wget* to download the tar file. Once downloaded, one can use the following code to reach the fits file. Sometimes the extracted items from the tar file can be gzipped, so included is example code on how to handle that as well.

```
from wget import download
import tarfile, os, glob, gzip

```

[1] GROWTH Project: https://www.youtube.com/channel/UC4d0KFk3ZFfv6xeyX5PHh5g/featured
[2] ZOGY: https://iopscience.iop.org/article/10.3847/0004-637X/830/1/27/pdf
[3] ESA HSA: http://hst.esac.esa.int/ehst/#home
[4] HCV Explorer: http://hst.esac.esa.int/hcv-explorer/

```
link = 'http://hst.esac.esa.int/ehst-sl-server/servlet/data-action?RETRIEVAL_TYPE=PRODUCT&
    OBSERVATION_ID=hst_13386_t3_wfc3_ir_f160w'
tar_name = download(link)
f = tarfile.open(tar_name, mode='r|*')
working_dir_path = '/content/fits'  # CHANGE TO WHEREVER YOU WANT THE DATA TO BE EXTRACTED
f.extractall(path=working_dir_path)

os.chdir('/content/fits/HST/') # CHANGE DIR HOLDING FITS FILES
folder_name = glob.glob('*')[0] # USE GLOB TO GET FOLDER NAME
fits_name = gzip.open(f'{folder_name}/{folder_name}_drz.fits.gz') # GUNZIP FILE
```

2.1.3 Hubble Legacy Archive

Another way to download data is through the Hubble Legacy Archive[5]. Search for the desired fits file and add it to the cart. One can filter certain options by typing in the blue boxes in the top row. Once ready to check out, there are three possible ways provided to download the data: Zipped File, Download Sequentially, or Curl Script.

2.1.4 Astroquery

Using *Astroquery*[6], this method of downloading data allows one to search for fits files within the Python program. It is a way to access databases such as HLA from the previous section or MAST right within the code. To install *Astroquery*, enter *pip install astroquery* in the terminal. The following is example code on how to search within MAST, filter down to HLA files, proceed to download the first five files, and save the name of the first downloaded fits file.

```
from astroquery.mast import Observations

obs_table = Observations.query_object("a2744",radius=".01 arcsec") # SEARCH
data_products_by_obs = Observations.get_product_list(obs_table) # TABLE WITH LEFT HAND SIDE
    BEING OBSID
print(data_products_by_obs)

satellites = []
for satellite in data_products_by_obs['obs_collection']:
  if satellites.count(satellite) == 0:
    satellites.append(satellite)
print(satellites) # SEE AVAILABLE TELESCOPES

hla_products = Observations.filter_products(data_products_by_obs,
                                            obs_collection="HLA",
                                            extension="fits") # FILTER TO ONLY HLA FITS FILES

first_hla_products = hla_products[0:5] # DOWNLOAD FIRST 5 FILES
download_hla = Observations.download_products(first_hla_products)

fits_name = download_hla['Local Path'][0] # GET PATH OF FIRST FITS FILE
```

2.2 Using Fits Files

Once one has a fits file downloaded and its path is saved to a variable, one can access the headers and data using *Astropy*[7]. To install *Astropy*, open the terminal and enter *pip install astropy*. A fits file is made up of header/data units, HDU for short. The header is made up of cards that hold the value of things such as exposure start time and duration. The data of a fits file is a matrix where each pixel holds the number of

[5]HLA: https://hla.stsci.edu/
[6]Astroquery: https://astroquery.readthedocs.io/en/latest/
[7]Astropy: https://www.astropy.org/

counts of photons detected or the rate of counts/second. The unit of the a pixel can be found in the header. The following is example code on how to interact with a fits file.

```
import matplotlib.pyplot as plt
from astropy.stats import sigma_clipped_stats
from wget import download
from astropy.io import fits

link = 'https://mast.stsci.edu/api/v0.1/Download/file?uri=https://hla.stsci.edu/cgi-bin/
    getdata.cgi?dataset=hst_10861_18_acs_wfc_f475w_drz.fits'
fits_name = download(link)

fitsfile = fits.open(fits_name)

print(fitsfile.info()) # SEE HDUS STORED
hdu = fitsfile['SCI'] # GET HDU LABELED SCI
print(hdu.header)

#PLOT DATA
mean, median, std = sigma_clipped_stats(hdu.data)
plt.imshow(hdu.data, vmin = median - 5*std, vmax = median + 5*std)
plt.colorbar()
```

2.3 Background Subtraction

The first step in image subtraction is to subtract the background of both input images, commonly referred to as the reference image and the science image. The background of the image is diffused light from empty parts of the sky. It is usually not constant in one image itself and does not stay the same from one image to the next. Here is an example of how the background of an image can look like.

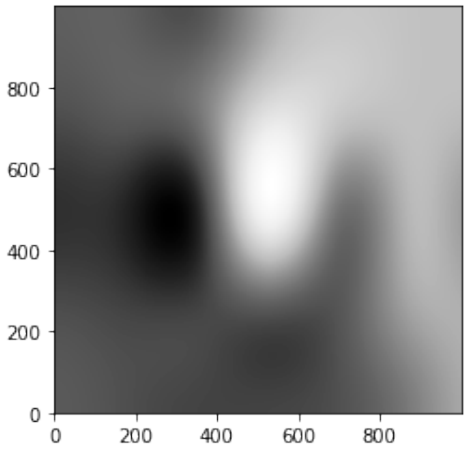

Figure 1: Example Background of an Image

Subtracting the background allows the sources in each of the images to be compared on a constant field. If one were to subtract images without subtracting the background from each of them first, areas of the difference image might be over-subtracted while other areas would be under-subtracted. To perform background subtraction for each image, one could *sep*[8], an SExtractor Python wrapper. To install sep, open terminal and enter *pip install sep*. When using *sep*, the input data might be rejected and one of the following errors might be returned.

[8]sep: https://sep.readthedocs.io/en/v1.0.x/index.html

[ValueError: Input array with dtype '¿f4' has non-native byte order. Only native byte order arrays are supported. To change the byte order of the array 'data', do 'data = data.byteswap().newbyteorder()']

OR

[ValueError: ndarray is not C-contiguous]

Follow the latter part of the first error to change the byte order. If that does not work, convert data using *np.array*() or a combination of both suggestions. Here is example code on how to use *sep*.

```
import sep
from astropy.io import fits
from wget import download

link = 'https://mast.stsci.edu/api/v0.1/Download/file?uri=https://hla.stsci.edu/cgi-bin/
    getdata.cgi?dataset=hst_10861_18_acs_wfc_f475w_drz.fits'
fits_name = download(link)
hdu = fits.open(fits_name)['SCI']

data = hdu.data.byteswap().newbyteorder()
bkg = sep.Background(data) # CALCULATE BACKGROUND
data_sub_bkg = data - bkg # SUBTRACT BACKGROUND
```

2.4 Image Alignment

The next step in the process is image alignment. The images need to be aligned just right because, if not, then one would get many artificial changes that are not actually there. A way to do this easily with Python is with *Astroalign*[9]. To install, open the terminal and enter *pip install astroalign*. This package finds at least three sources to match between the reference image and science image and reprojects the science image to match the orientation of the reference image. Here is example code on how to use it.

```
import astroalign as aa
...
sci_aligned, footprint = aa.register(sci_data, ref_data)
```

From here on you would then use *ref_data* and *sci_aligned*.

2.5 Match Point Spread Functions

The point spread function, PSF for short, is the essentially the image cropped onto just one source. One can typically think of the counts or counts/second of a source following a Gaussian shape. There are as many PSFs in an image as there are sources. If using data from observations taken on earth, the shape of the PSF usually will be different from the shape of the PSF from the same source on a different image. This is due to the scintillations caused by earth's atmosphere. For Hubble data, this step for the most part can be bypassed. For earth-based observations, one would have to transform the shapes of the PSFs in the science image to the shapes of the PSFs in the reference image. To do so, one could use *create_matching_kernel*()[10] from *Photutils* and *convolve*()[11] from *Astropy*. Here is example code one could follow.

```
from photutils.psf import create_matching_kernel
from astropy.convolution import convolve

#IDENTIFY POINT SOURCES IN REF IMAGE
bkg = sep.Background(ref_data)
sources = pd.DataFrame(sep.extract(np.array(ref_data), 500, err = bkg.globalrms)) # GET
    SOURCES ABOVE 500*SIGMA
final_sources = pd.DataFrame(columns = list(sources.columns))
```

[9] Astroalign: https://astroalign.readthedocs.io/en/latest/
[10] create_matching_kernel: https://photutils.readthedocs.io/en/stable/api/photutils.psf.matching.create_matching_kernel.html
[11] convolve: https://docs.astropy.org/en/stable/api/astropy.convolution.convolve.html

```
for index, row in sources.iterrows():
  if row['b']/row['a'] > .5: #GET ROUNDER OBJECTS (1 IS PERFECT CIRCLE)
    final_sources.loc[index] = row

#CONVOLVE THE PSFS TO MATCH
for index, row in final_sources.iterrows():
  ymin, ymax, xmin, xmax = int(row['ymin']), int(row['ymax']), int(row['xmin']), int(row['
    xmax'])
  # CONVOLVE ONLY ACCEPTS ODD NUMBER AXES
  if (ymax - ymin)%2 == 0:
    ymax += 1
  if (xmax - xmin)%2 == 0:
    xmax += 1
  kernel = create_matching_kernel(sci_aligned[ymin:ymax,xmin:xmax],ref_data[ymin:ymax,xmin:
    xmax])
  sci_aligned[ymin:ymax,xmin:xmax] = convolve(sci_data[ymin:ymax,xmin:xmax], kernel)
```

2.6 Scale Images

This step is to account for different exposure durations. Naturally, images that had longer exposure durations have higher counts of photons in general. If the pixel unit is in counts per second like in the case of Hubble data, then one can bypass this step since it has already been taken into account. For images with a pixel unit of counts, one way to account for this step is to multiply the data with longer exposure duration by the fraction (*shorter duration/longer duration*). This will then make the data from one image to the next comparable. Another way which might be more difficult is to calculate the flux of each source and compute the mean flux for both images. Then, multiply the image with the larger mean flux by the fraction (*smaller mean flux/larger mean flux*).

2.7 Subtract

After all the previous preparation, one can finally subtract the reference image from the science image. Subtracting the other way around will also work. If the science image is subtracted from the reference image and the reference image is older, the positive pixels in the difference image are sources that have decreased in brightness. If the reference image is subtracted from the science image and the reference image is older, the positive pixels in the difference image are sources that have increased in brightness. Subtraction is simply taking one matrix minus the other matrix.

2.8 Crop the Difference Image

When detecting changes from one image to another, only the overlapping area of the difference image is wanted. Leaving the surrounding border can lead to bugs in the code. Here is some sample code on how to crop and store the difference image in a fits file. The header used will be the reference image's header since the science image was reprojected to match the reference image.

```
#CROP
xstart, xend, ystart, yend = 2000, 4000, 1200, 3600

w = WCS(ref.header)
dif_cropped = fits.PrimaryHDU()
dif_cropped.data = difference[ystart:yend,xstart:xend]
dif_cropped.header = ref_hdu.header
dif_cropped.header.update(w[ystart:yend,xstart:xend].to_header())

#plot
mean, median, std = sigma_clipped_stats(difference)
plt.figure(2,figsize=(11,11))
plt.subplot(1,2,1)
plt.imshow(difference, vmin = median-5*std, vmax = median+5*std)
```

```
plt.colorbar()
plt.axhline(ystart)
plt.axhline(yend)
plt.axvline(xstart)
plt.axvline(xend)
plt.title('Difference Image')

mean, median, std = sigma_clipped_stats(dif_cropped.data)
plt.subplot(1,2,2)
plt.imshow(dif_cropped.data, vmin = median-5*std, vmax = median+5*std)
plt.colorbar()
plt.title('Cropped Difference Image')
plt.show()
```

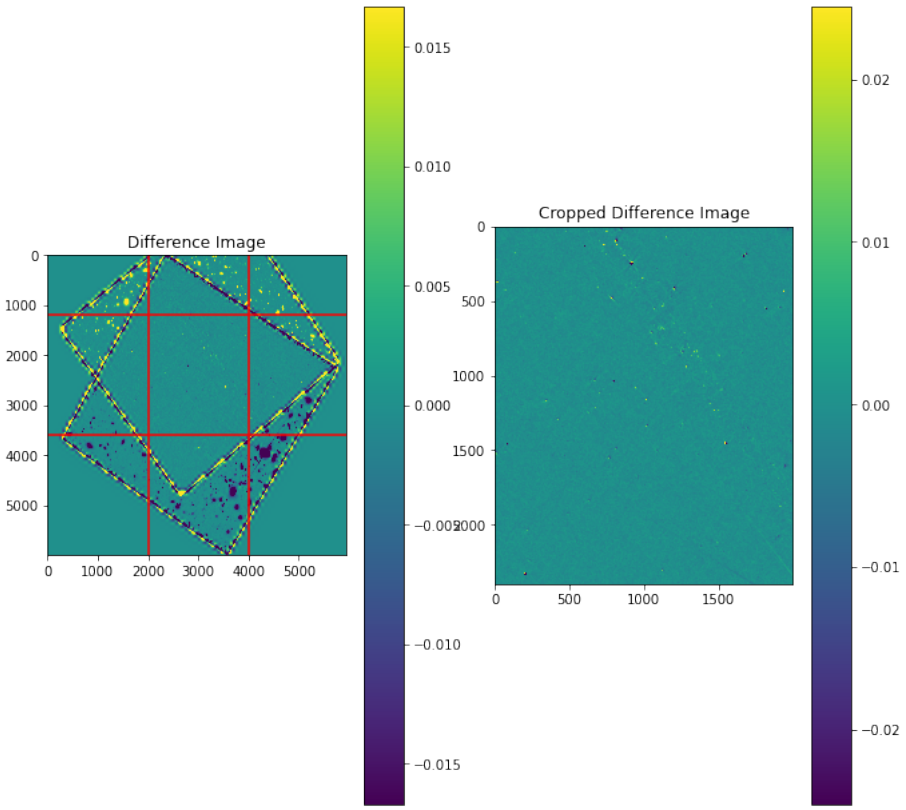

Figure 2: Example Crop of the Difference Image

2.9 Source Detection

To detect sources within the difference image, one could use *sep.extract*(). It would be best to then convert the returned table into a Pandas DataFrame which makes it easier to read and handle. One can then add the *RA* and *Dec* using the *pixel_to_world*() function in *Astropy* to the DataFrame.

```
dbkg = sep.Background(np.array(dif_cropped.data))
objects = pd.DataFrame(sep.extract(np.array(dif_cropped.data), threshold, minarea=minarea,
    err = dbkg.globalrms, deblend_nthresh=deblend)) #detect

#filter out non-round objects
final_objects = pd.DataFrame(columns = list(objects.columns))
```

```
for index, row in objects.iterrows():
  if row['b']/row['a'] > ab_fraction_min:
    final_objects.loc[index] = row
final_objects.reset_index(inplace=True, drop=True)

w = WCS(dif_cropped.header) # convert from pixel to world coords
final_objects['RA (deg)']=w.pixel_to_world(final_objects['x'],final_objects['y']).ra.deg
final_objects['Dec (deg)']=w.pixel_to_world(final_objects['x'],final_objects['y']).dec.deg

fig, ax = plt.subplots(figsize=(20,20))
mean, median, std = sigma_clipped_stats(dif_cropped.data)
im = ax.imshow(dif_cropped.data, vmin = median-5*std, vmax = median+5*std)

#factor to multiply ellipses to see better
multx = int(dif_cropped.data.shape[0]*.025)
multy = int(dif_cropped.data.shape[1]*.025)
for i in range(len(final_objects)):
  e = Ellipse(xy=(final_objects['x'][i], final_objects['y'][i]),
              width=multx*final_objects['a'][i],
              height=multy*final_objects['b'][i],
              angle=final_objects['theta'][i] * 180. / np.pi)
  e.set_facecolor('none')
  e.set_edgecolor('red')
  ax.add_artist(e)
plt.show()
```

It seems that the following parameters work well.

1. threshold = 5 (Threshold HCV used)
2. minarea = 2 (Minimum value able to discover SN HFF 14Tom)
3. deblend = 1 (A low value decreases double counting of sources)
4. ab_fraction_min = .90 (Can range from 0 to 1 with 1 being a perfect circle)

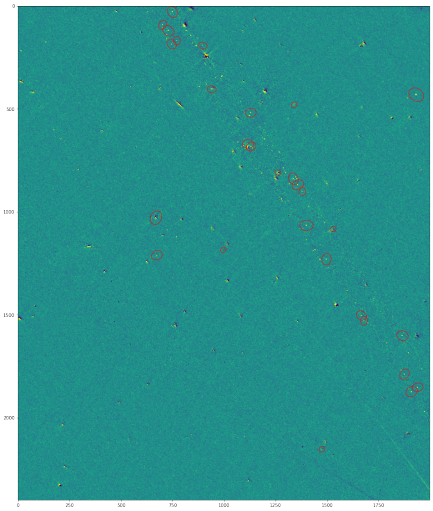

Figure 3: Example Detection of Variable Sources in A2744

To quickly add, a way to convert from a world coordinate to pixel coordinate is to use the *world_to_pixel*() function from *Astropy*. Here is a small example.

```
ra=3.574458
dec=-30.399917
c = SkyCoord(ra,dec,unit='deg')
x,y = w.world_to_pixel(c)
```

2.10 Saving Results

Noted here are ways to save three files: fits file of the difference image, region file of the detected sources which can then be loaded into DS9, and CSV file which contains the *final_objects* DataFrame from the previous section.

```
fits.writeto('difference.fits',dif_cropped.data,dif_cropped.header)
final_objects.to_csv('src.csv')

f = open("src.reg", "a")
f.write("# Region file format: DS9 version 4.1\n")
f.write("global color=green dashlist=8 3 width=1 font=\"helvetica 10 normal roman\" select=1
     highlite=1 dash=0 fixed=0 edit=1 move=1 delete=1 include=1 source=1\n")
f.write("fk5\n")
for index, row in final_objects.iterrows():
  f.write(f"circle({row['RA (deg)']},{row['Dec (deg)']},1.00\")\n")
f.close()
```

3 Light Curve Computation

A light curve is a graph that displays a source's brightness over time. A graph like this helps determine whether a variable source is sinusoidally variable or transient. This section goes through the process in computing the flux and magnitude of a source in Hubble. It is important to note that, when a source's brightness increases, the absolute magnitude decreases and vice versa. The process documented here seems to be limited to only isolated sources or sources that are brighter than those around it. Once one learns to calculate the magnitude of a source in one image, one should use a for loop for the other images the source is in, store the results, and plot the light curve.

3.1 Calculating Flux

Before beginning, be sure to subtract the background of the entire image. To calculate the flux of an object, one must zoom into the source. Make sure the source is centered and inside the entire square box. Sometimes the calibration is a bit off when the header was created so the source might be off-centered. To correct for this, simply place the brightest pixel of the object in the center of the box. However, be cautious that another object is not mistaken. One should only adjust the source by maybe one or two pixels. The next step is to sum all the pixels together. In Hubble data, each pixel's unit is in counts per second. By summing up a box, one gets the count rate in a specific aperture. To get the desired count rate in an infinite aperture, one must divide the sum by the fractional ensquared energy which is a constant dependent on the filter of the telescope and the size of the source's box. One can find this constant on the Space Telescope Science Institute's website[12]. One can also have the source be summed inside a circular box. There is a function in Python called *sum_circle*()[13] that can return the encircled count rate. If one gets an error using this function, a possible solution can be found in Section 2.3. One must still divide by the fractional encircled energy to get the count rate in an infinite aperture. One can also find the encircled energy on the Space Telescope Science Institute's website[14]. Do note that the footnotes direct to the EE table for WFC3 and not ACS. An analogous ACS version can be found in this footnote[15]. Once one has the count rate in an

[12]Ensquared Energy: https://www.stsci.edu/itt/APT_help/WFC3/c07_ir07.html
[13]sum_circle: https://sep.readthedocs.io/en/v0.5.x/api/sep.sum_circle.html
[14]WFC3 EE: https://www.stsci.edu/hst/instrumentation/wfc3/data-analysis/photometric-calibration/ir-encircled-energy
[15]ACS EE: https://www.stsci.edu/hst/instrumentation/acs/data-analysis/aperture-corrections

infinite aperture, the units must be converted to $erg\ cm^{-2}\ s^{-1}\ Å^{-1}$ to get flux. To do this, multiply the count rate by $PHOTFLAM$, a constant which can be found in the header of the fits file. To note the time of the recorded flux, one could take the average of the exposure start time, $EXPSTART$, and the exposure end time, $EXPEND$, also found in the header of the fits file. Here is an example of how to calculate the flux of a source in Python.

```
ra = 3.6047835350
dec = -30.4149532318
size = 1 #INITIAL CROP TO FIND CENTER OF SOURCE
radius = 2 #5x5 BOX (RADIUS=3 IS 7x7 BOX, RADIUS=4 IS 9x9 BOX)
ee = .825

#CONVERT FROM WORLD TO PIXEL
w = WCS(hdu.header)
c = SkyCoord(ra = ra, dec = dec, unit='deg')
x,y = w.world_to_pixel(c)
x,y = int(x),int(y)

time = (hdu.header['EXPSTART'] + hdu.header['EXPEND'])/2
print("MJD:", time)

#SUBTRACT BACKGROUND
bkg = sep.Background(hdu.data.byteswap().newbyteorder())
data = hdu.data.byteswap().newbyteorder() - bkg

#CROP
cropped_data = data[y-size:y+size,x-size:x+size]

#CENTER ON OBJECT
y_max, x_max = np.where(cropped_data == cropped_data.max())
x_max, y_max = int(x_max), int(y_max)
better_x, better_y = x_max+(x-size), y_max+(y-size)
object_center = pd.DataFrame({'x':[better_x],'y':[better_y]})

box = data[better_y-radius:better_y+radius+1,better_x-radius:better_x+radius+1]

cr = np.sum(box)
flux = (cr/ee)*hdu.header['PHOTFLAM']
print("Flux:",flux)
```

Here is a general formula where CR is the count rate and EE is the fraction ensquared energy.

$$F_\lambda = (\frac{CR}{EE}) * PHOTFLAM \tag{1}$$

The reason why flux is notated as F_λ is because there are two separate ways to calculate flux, F_λ and F_ν. F_λ is what Equation 1 returns which has units of $erg\ cm^{-2}\ s^{-1}\ Å^{-1}$. F_ν has units of $erg\ cm^{-2}\ s^{-1}\ Hz^{-1}$. To get F_ν, instead of multiplying the count rate by $PHOTFLAM$, multiply it by $PHOTFNU$. Since $PHOTFNU$ is in $Jy * sec/electron$ and since 1 Jy is $10^{-23} erg\ cm^{-2}\ s^{-1}\ Hz^{-1}$, multiply the result by 10^{-23} to get units of $erg\ cm^{-2}\ s^{-1}\ Hz^{-1}$.

$$F_\nu = (\frac{CR}{EE}) * PHOTFNU * 10^{-23} \tag{2}$$

3.2 Calculating Magnitude

There are different types of magnitude systems including standard, absolute, and Vega. The formulas to get to each magnitude system can be found on the Space Telescope Science Institute's website[16][17]. To

[16]Formulas: https://www.stsci.edu/hst/instrumentation/wfc3/data-analysis/photometric-calibration
[17]Formulas: https://www.stsci.edu/hst/instrumentation/acs/data-analysis/zeropoints

get VEGAmag, use the following formula where F_λ is the flux from Equation 1 and F_{vega} is the absolute CALSPEC flux of Vega[18]. Make sure F_λ and F_{vega} have the same units of $erg\ cm^{-2}\ s^{-1}\ \text{Å}^{-1}$.

$$VEGAmag = -2.5 * log_{10}(F_\lambda / F_{vega}) \tag{3}$$

Another way to obtain the VEGAmag of a source can be found in this example[19].

To get STmag, use the following formula where F_λ is the flux from Equation 1.

$$STmag = -2.5 * log_{10}(F_\lambda) - 21.10 \tag{4}$$

To get from STmag to ABmag, use the following conversion where $PHOTPLAM$ is the pivot wavelength which can be found in the header of the fits file.

$$ABmag = STmag - 5 * log_{10}(PHOTPLAM) + 18.6921 \tag{5}$$

Note that ABmag is the system that is used in the Hubble Source Catalog which includes the Hubble Catalog of Variables.

A way to get directly from flux to ABmag is to use the following formula where F_ν is the flux from Equation 2.

$$ABmag = -2.5 * log_{10}(F_\nu) - 48.60 \tag{6}$$

4 Next Steps

The next step is to make a program in Python that computes the light curve of a source regardless if the source is isolated or not. Since the current procedure is limited to only isolated sources, it would be great to have a program that works regardless of context. Since image subtraction can discover variable candidates it would be beneficial to create another program that cross-checks variable sources found with SIMBAD or NED, websites that classify objects. The objects that are categorized as unknown or not listed at all are the ones that would be interesting to look into their light curves. To automatically determine if an unknown source is sinusoidally variable or transient, a machine learning model can be trained to classify the object. An aspect in the subtraction process that must be improved is the cropping action. Code must be developed in order to crop out the overlapping section of the difference image automatically. Once this pipeline of inputting multiple images and outputting coordinates of variable sources, one must find a way to access a large amount of images reliably and feed them to the pipeline. A human can supervise the process to determine if variable and transient sources are truly discovered.

[18] F_{vega}: https://www.stsci.edu/hst/instrumentation/reference-data-for-calibration-and-tools/astronomical-catalogs/calspec

[19] Vega: https://www.stsci.edu/hst/instrumentation/wfc3/data-analysis/photometric-calibration/uvis-encircled-energy