# Documentation of python script for photometric reduction using aperture photometry


**Developed by:**

Raghav Wani
Vivek Jha
Gyanankur Bora

**Under guidance of**

Dr. Yogesh C. Joshi
Aryabhatta Research Institute of Observational Sciences (ARIES), Nainital

***June 2023***

# Table of contents

# 1. Introduction

This is a documentation for the python script developed during Indian Academy of Sciences Summer Research fellowship program 2023. The report documentation detailed working of the python pipeline to carry out the workflow of photometry from cleaning raw images captured by telescope to creating light curves of stars.

Photometry is the measurement of the brightness or intensity of light emitted by celestial objects. It plays a crucial role in studying stars, galaxies, and other astronomical phenomena. Different techniques are used in photometry, including aperture photometry, PSF photometry, differential photometry, and time-series photometry. Aperture photometry involves measuring the flux within a predefined circular or elliptical aperture around the object. PSF photometry models the shape of the object's point spread function to estimate the flux, which is useful in crowded fields. Differential photometry compares the brightness of an object with one or more reference objects to mitigate atmospheric and instrumental effects. Time-series photometry focuses on studying the variability of an object's brightness over time.

The IRAF/DAOPHOT-based pipelines are traditionally popular tools for photometry. But these pipelines might require a significant amount of time, experience and constrained usage. As an alternative, several other tools are available for performing photometry. Astropy, an open-source Python library, provides comprehensive tools for aperture and PSF photometry, as well as time-series analysis. Photutils, an affiliated package of Astropy, offers a user-friendly interface and a range of algorithms for various photometric analyses. SEP (Source Extraction and Photometry) is a Python library focused on source extraction and photometry, particularly in crowded fields. Sextractor is another method in photometry, which estimates the local background and subtracts it from the measured flux, allowing for accurate source separation. It is versatile, customizable, and capable of handling large-scale surveys. However, Sextractor may struggle with crowded fields or complex source morphologies.

Python-based pipelines for photometry have advantages, as python is widely used in the scientific community, providing accessibility and flexibility. Python-based tools are often user-friendly and customizable. Integration with other scientific libraries, such as NumPy, SciPy, and Matplotlib, enables advanced data analysis and visualization. Python's vibrant community ensures ongoing support, development, and availability of resources. Python's interoperability facilitates integration with other languages and software.

Hence, we have developed a python-pipeline to make the process of photometry efficient and dynamic. We performed tasks like

1. Cleaning, which involves master bias and master flat subtraction from science frames and removal of cosmic rays,
2. Aperture photometry which involves detection and calculating the magnitude of stars.
3. Calibration of these magnitude values using SIMBAD database.
4. Plotting light curves, phase-magnitude diagram and thus calculating period and amplitude of variation of magnitude of a star.

Using this python pipeline, we were able to detect variable stars in the NGC6709 star cluster. We used data captured from Devasthal Fast Optical Telescope (DFOT) covering a time period of more than a year.

The code is developed for the data on which astrometry is already performed. astrometry.net, a module within Astroquery, is a tool that solves astrometry problems by determining precise celestial coordinates in astronomical images. Astrometry works by analyzing the image and comparing it to a vast reference catalog of known stars and objects. Using advanced algorithms, astrometry.net identifies and matches patterns in the image with objects in the catalog, allowing it to accurately determine the coordinates. With astrometry.net, users can enhance the accuracy of celestial coordinates in images and enable more precise astronomical analysis.

## 2. Workflow

### A. Cleaning

1. Bias cleaning

Bias frames capture the inherent electronic noise present in the camera system. These frames are obtained by taking multiple exposures with zero integration time or extremely short exposure times, ensuring that no astronomical signal is recorded. By subtracting the bias frame from science or flat frames, unwanted noise like readout noise and dark current can be eliminated. For this, the bias frames are combined into a master bias frame to create a reference for calibration. Regularly capturing bias frames ensures accurate noise reduction and reliable data analysis.
Below is the Python code snippet for creating master bias frame:

```
masterbias = ccdproc.combine(biaslist,method='median',sigma_clip=True,
sigma_clip_low_thresh=5, sigma_clip_high_thresh=5,
sigma_clip_func=np.ma.median,sigma_clip_dev_func=mad_std)
```

The ccdproc.combine() function, a part of the ccdproc library, is used for combining or stacking multiple CCD (charge-coupled device) frames into a single frame. In the above code, biaslist is the variable containing the list of individual bias frames captured during the imaging session. Here we have used the Sigma clipping data processing technique to remove outliers from a dataset. It involves iteratively identifying data points that deviate significantly from the central value and excluding them from further analysis. median is used as a method, which calculates the median value at each pixel position from all the bias frames.
sigma_clip_low_thresh=5 and sigma_clip_high_thresh=5 are thresholds that define the range within which pixel values are considered valid. Any pixel value that deviates by more than 5 standard deviations from the median is considered an outlier and will be rejected during the

combination. sigma_clip_func specifies the function to use for computing the median when applying sigma clipping. We used the mad_std function to calculate the median absolute deviation (MAD) as a robust estimator of the standard deviation.

2. Flat-fielding

Flat frames are used to correct for non-uniformities in the optical system, such as dust particles, vignetting, and pixel-to-pixel sensitivity variations. Flat frames are obtained by taking exposures of a uniformly illuminated field, such as twilight sky or a diffusing screen. Multiple flat frames are combined using methods like averaging or median to create a master flat frame. The master flat frame serves as a reference to normalize the science frames, compensating for pixel-to-pixel variations and ensuring accurate representation of the astronomical objects' true intensities.
Usually 4-5 flat frames that are captured in the same band as science frames are used.
Below is the Python code snippet for creating master flat frame:

```
masterflat = ccdproc.combine(flatlist,method='median',
scale=inv_median,sigma_clip=True,
sigma_clip_low_thresh=5,sigma_clip_high_thresh=5,
sigma_clip_func=np.ma.median, sigma_clip_dev_func=mad_std)
```

Similar to master bias creation, here too we have made use of ccdproc.combine() function and its parameter to create master flat frames. Only difference being Flatlist is used as a variable containing the list of individual flat frames each of the same band.

3. Science frame cleaning:

```
bias_subtracted = ccdproc.subtract_bias(image, mbias)
flat_corrected = ccdproc.flat_correct(bias_subtracted, mflat)
```

Above code first subtracts the master bias frame from the science frame. For this ccdproc.subtract_bias() function is used to subtract the master bias frame (mbias) from the input science frame (image). This removes the bias signal from the individual science frame.
Next ccdproc.flat_correct(bias_subtracted, mflat) function performs flat field correction on the bias-subtracted image (bias_subtracted) using the master flat frame (mflat). Flat field correction compensates for pixel-to-pixel variations and non-uniformities in the imaging system's response.

4. Cosmic ray correction:

Cosmic rays are high-energy particles, such as protons and atomic nuclei, that originate from various sources, including the Sun, supernovae, and other astronomical events. Cosmic ray correction is used to remove the impact of cosmic rays on astronomical images or spectra. When cosmic rays interact with the detectors or sensors used in astronomical observations, they can create unwanted artifacts or spikes in the recorded data. These artifacts can significantly affect the accuracy and reliability of scientific analysis. There are several methods commonly used for cosmic ray correction in astronomical data:

- Sigma-Clipping: Apply a statistical technique to detect outliers and replace cosmic ray-affected pixels based on neighboring values.
- Median Filtering: Use a median filter to identify and replace cosmic ray hits with the median value of neighboring pixels.
- Masking and Interpolation: Flag affected pixels, interpolate values using neighboring pixels, and fill in the gaps caused by cosmic ray hits.
- Multiple Image Combining: Stack multiple exposures and identify cosmic ray hits as outliers among the frames.
- Wavelet Transform: Analyze the spatial frequency content of the data to identify and remove high-frequency signatures associated with cosmic rays.

We have used masking and interpolation method. The L.A.Cosmic (Laplacian Cosmic Ray Identification) algorithm uses this method and is a widely used technique for identifying and removing cosmic ray hits in astronomical images. It works by identifying pixels that deviate significantly from their local background and spatial context. Here's the brief working of this technique:

1. The algorithm first estimates the local background for each pixel in the image by applying a median filter or a more sophisticated background estimation technique.
2. A Laplacian operator is then used to detect cosmic ray hits as sharp positive or negative peaks in the image that deviate significantly from the local background.
3. The identified cosmic ray hits are masked or flagged in a separate "cosmic ray mask" that marks the affected pixels.
4. Interpolation or replacement is performed on the masked pixels using neighboring pixels or a more advanced interpolation technique, filling in the gaps left by the cosmic ray hits.

The above procedure is done in a single step using ccdproc.cosmicray_lacosmic() function provided by ccdproc library. Below is the python snippet for this purpose:

```
ccdproc.cosmicray_lacosmic(flat_corrected, readnoise=10,
sigclip=5, verbose=True)
```

This function performs cosmic ray removal using the LACosmic algorithm on the flat-corrected image (flat_corrected). The readnoise parameter represents the read noise of the camera (typically given in electrons). The sigclip parameter is the threshold for sigma clipping,

and verbose=True enables verbose mode to display additional information like number of cosmic rays removed in each iteration, etc. during the cosmic ray cleaning process.
Above code it looped over all the science frames to get cleaned images of the source.

### B. Aperture photometry

Aperture photometry is a technique used in astronomy to measure the total flux or brightness of a celestial object within a defined aperture or region of interest. It involves summing the pixel values within the aperture and subtracting the contribution from the background sky to obtain the net flux. Depending on the source, the aperture can be chosen to be circular, elliptical or rectangular.

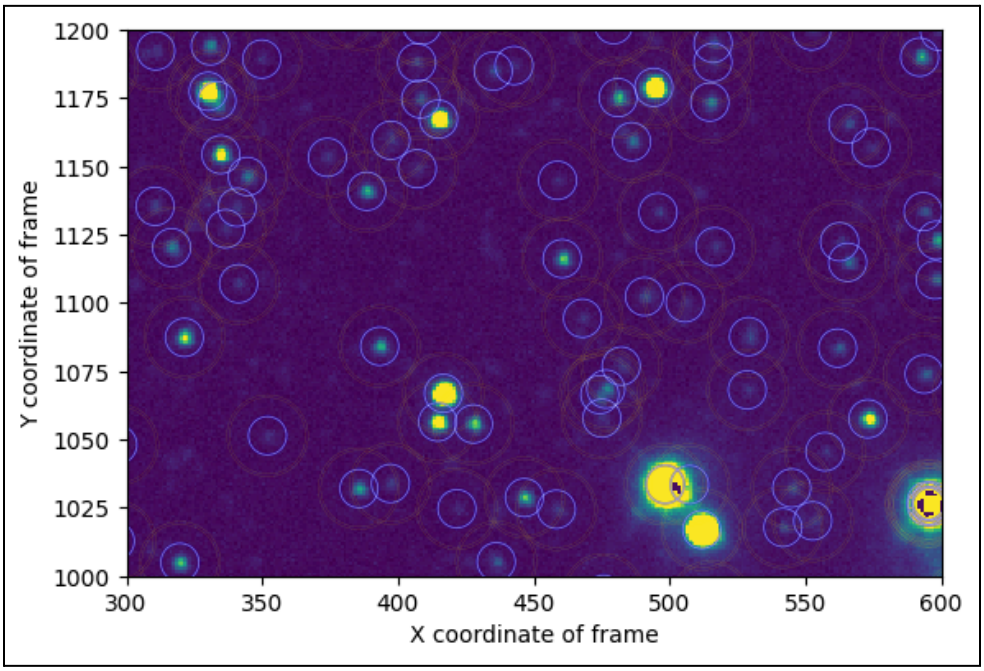


Figure 1: A region of NGC 6709 with aperture around the detected stars. The white circle is an aperture of radius 7 pixel and the region between two red circles is the annulus that accounts for the background of the aperture.

This is an important step of the entire process of photometric reduction. The two functions defined in the main code perform the procedure of source detection, aperture photometry and magnitude calculation. The first function detects the sources or stars from the data gathered from the cleaned science frame. The second function performs aperture photometry and lastly calculates the flux or magnitude of each star detected in the previous definition. Here is the detailed explanation of these functions.

1. Function 1st:

```
def source(data, header):
      sigma_clip = SigmaClip(sigma=3, maxiters=10)
      bkg_estimator = SExtractorBackground()
      bkg = Background2D(data, (10,10), filter_size=(3, 3),
sigma_clip=sigma_clip, bkg_estimator=bkg_estimator)
      back = bkg.background
      h = fits.open(filename)
      wcs = WCS(h[0].header)
      mask = data == 0
      unit = u.electron / u.s

      xdf_image = CCDData(data, unit=unit, meta=header, mask=mask)
      norm_image = ImageNormalize(vmin=1e-4, vmax=5e-2,
      stretch=LogStretch(), clip=False)
      xdf_image_clipped = np.clip(xdf_image, 1e-4, None)

      mean, median, std = sigma_clipped_stats(xdf_image.data,
      sigma=3.0, maxiters=20, mask=xdf_image.mask)
      daofind = DAOStarFinder(fwhm=4, threshold=5*std)
      return daofind(data - back)
```

First line of the definition creates a SigmaClip object named sigma_clip with a sigma value of 3 and a maximum number of iterations of 10. SigmaClip is a method commonly used for sigma clipping, which is a statistical technique to reject outliers in a dataset.
Next SExtractorBackground is a background estimator implementation that uses the SExtractor algorithm. This algorithm estimates the background by modeling it with a low-order polynomial. Next the code computes the 2D background estimation using the Background2D function. Here, data is the input data array on which the background is estimated, the size of the grid cells used for estimating the background is 10x10 pixels, filter_size is the size of the filter kernel used for smoothing the data, sigma_clip is the sigma clipping object used for rejecting outliers during background estimation and bkg_estimator is the background estimator used for modeling the background. The variable back represents the background estimation for the input data.

Further the code opens a FITS file (filename) using the fits.open function from the astropy.io.fits module and then extracts the header from the primary HDU (h[0].header) and assigns it to the variable wcs, which represents the World Coordinate System information. Since astrometry is already performed on all the FITS files, the pixels are mapped to the WCS coordinates.

The code creates a mask based on the equality comparison of the data array with zero. The mask can be used to identify and handle specific elements in the data array. u.electron represents the unit of electrons, and u.s represents the unit of seconds. Therefore, u.electron / u.s represents the unit of electrons per second, indicating the rate of electron accumulation or emission over time. The unit provides context for the data's physical interpretation.
The code then creates a CCDData object named xdf_image. The CCDData class is typically used to represent CCD (Charge-Coupled Device) image data. Here, unit is the physical unit of the data (electrons per second), meta=header is the metadata or header associated with the data, and mask is the mask array that indicates which elements of the data array are valid.
Next, an ImageNormalize object named norm_image is created. The ImageNormalize class is used for normalizing and stretching image data for display purposes. Here, vmin and vmax are the minimum and the maximum value for the data range after normalization and stretch is the stretch function to be applied to the data. clip=False is to check whether to clip the data range if it exceeds the provided vmin and vmax values.

A clipping operation is applied on the xdf_image array using np.clip from NumPy. The np.clip function ensures that all values in the xdf_image array are within the specified range. In this case, it clips values below 1e-4 to 1e-4, while values above the upper range are left unclipped (None is used as the upper range).
Next the sigma_clipped_stats function is called to compute the mean, median, and standard deviation of the data array of xdf_image. The function performs sigma clipping with a sigma value of 3.0 and a maximum number of iterations of 20. The mask of xdf_image is used to ignore masked or invalid values during the statistics calculation.

The photutils library provides various tools for detecting stars and performing photometry. It includes PSF photometry for fitting stellar PSFs (Point Spread Functions) to data, source detection algorithms like DAOFIND and starfind, background estimation algorithms which are helpful in reliable source detection and source deblending method that separates overlapping sources.

We have used the DAOFIND method, which is commonly chosen for star finding due to its simplicity, robustness against variations, computational efficiency, and wide usage and testing in the astronomical community. Using this the code creates a DAOStarFinder object named daofind for source detection.Here, the full width at half maximum (fwhm) of the stars expected in the image helps in determining the appropriate scale for detecting sources and threshold sets the minimum intensity level required for a pixel to be considered part of a source. Here, it is set to 5 times the computed standard deviation.
The function then returns the result of subtracting the estimated background (back) from the data array and passing it as input to the daofind object.

2. Function 2nd:

```
def magnitude(data, header, apertures, an_ap ):
      sigma_clip = SigmaClip(sigma=3, maxiters=10)
      bkg_estimator = SExtractorBackground()
      bkg = Background2D(data, (10,10), filter_size=(3,3),
sigma_clip=sigma_clip, bkg_estimator=bkg_estimator)
      back = bkg.background
      exposure = header['EXPTIME']
      effective_gain = exposure
      error = calc_total_error(data, back, effective_gain)
      phot_table = aperture_photometry(data - back, apertures,
error=error)
      phot_table2 = aperture_photometry(data - back, an_ap)
      bkg_mean = phot_table2['aperture_sum'] / an_ap.area
      bkg_sum = bkg_mean * an_ap.area
      final_sum0 = phot_table['aperture_sum_0'] - bkg_sum
      final_sum1 = phot_table['aperture_sum_1'] - bkg_sum
      final_sum2 = phot_table['aperture_sum_2'] - bkg_sum
      final_sum3 = phot_table['aperture_sum_3'] - bkg_sum
      final_sum4 = phot_table['aperture_sum_4'] - bkg_sum
      mag_back = -2.5 * np.log10(bkg_mean / exposure)
      mag_0 = -2.5 * np.log10(final_sum0 / exposure)
      mag_1 = -2.5 * np.log10(final_sum1 / exposure)
      mag_2 = -2.5 * np.log10(final_sum2 / exposure)
      mag_3 = -2.5 * np.log10(final_sum3 / exposure)
      mag_4 = -2.5 * np.log10(final_sum4 / exposure)
      flux_err_0 = phot_table['aperture_sum_err_0']
      mag_err_0 = 1.09 * flux_err_0 / final_sum0
      #rest of the code is skipped here
      return  mag_0, mag_err_0, mag_1, mag_err_1, mag_2, mag_err_2,
mag_3, mag_err_3, mag_4, mag_err_4
```

Firstly the code performs background estimation on the input data using the Background2D function. It estimates the background by applying a sigma clipping method (SigmaClip) and a background estimator (SExtractorBackground). The resulting background is assigned to the variable back.

Next, the code retrieves the exposure time (EXPTIME) from the header object. The effective_gain is set to the exposure time. The effective_gain is a parameter used in the

calculation of the error associated with the measured flux. It represents the conversion factor between the number of detected photoelectrons and the counts in the image. By setting effective_gain to exposure implies that each count in the image corresponds to one photoelectron.Thus the error calculation takes into account the statistical uncertainty associated with the measured flux, considering that the noise properties are related to the number of photoelectrons recorded during the exposure.

Additionally, the error is calculated using the calc_total_error function, which takes the input data, background (back), and effective gain as parameters. The error represents the uncertainty associated with the data.

Further the code performs aperture photometry on the background-subtracted data (data - back). The aperture_photometry function is used to compute photometric measurements within specified apertures. phot_table stores the photometric measurements using the provided aperture sizes (apertures) and the calculated error, while phot_table2 stores the measurements using an annulus of the aperture (an_ap).

Then the code calculates the mean background value (bkg_mean) by dividing the sum of aperture measurements (phot_table2['aperture_sum']) by the area of the annulus (an_ap.area). Then, the total background contribution (bkg_sum) is computed by multiplying the mean background value with the annulus area.

Next the background contribution (bkg_sum) is subtracted from the aperture measurements in phot_table, corresponding to different apertures ('aperture_sum_0', 'aperture_sum_1', etc.). The results are stored in variables final_sum0, final_sum1, etc., representing the final source fluxes after background subtraction.

Further the code calculates the magnitudes for different measurements. Magnitude is a logarithmic measure of the brightness of an astronomical object. The formula -2.5 * np.log10(flux / exposure) is used, where flux is the measured flux value after background subtraction and exposure is the exposure time. The magnitudes are stored in variables mag_back, mag_0, mag_1, etc.

Lastly, the magnitude errors (uncertainties) for the measured fluxes are computed. The flux errors are extracted from phot_table ('aperture_sum_err_0', etc.), and the magnitude errors are calculated using the formula 1.09 * flux_err / final_sum, where flux_err is the flux error and final_sum is the background-subtracted flux. This formula for magnitude (*mag*) can be derived from flux as:

$$mag = -2.5 * log_{10}(flux) + c$$

$$\delta mag / \delta flux = -2.5 / flux * ln\,10 \quad \text{…differentiating above equation}$$

$$mag\_err = |(\delta mag / \delta flux)| * flux\_err \quad \text{…formula for error propagation}$$

$$\therefore mag\_err = |(-2.5/ln10)| * flux\_err / flux$$

$$\therefore mag\_err = 1.0857 * flux\_err / flux$$

$$\therefore mag\_err \approx 1.09 * flux\_err / flux$$

The magnitude errors are stored in variables mag_err_0, mag_err_1, etc.
Finally, the function returns the computed magnitudes and magnitude errors: mag_0, mag_err_0, mag_1, mag_err_1, mag_2, mag_err_2, mag_3, mag_err_3, mag_4, mag_err_4. These represent the measured magnitudes and their associated uncertainties for different apertures or source measurements.

When performing aperture photometry, the choice of an optimum aperture size is important to ensure accurate and reliable flux measurements. The growth curve helps in determining this optimum aperture size. A growth curve represents the relationship between the aperture size used for measuring the flux of an astronomical object and the resulting flux or magnitude values obtained. It shows how the measured flux/magnitude changes as the aperture size increases. The optimum aperture size can be selected by considering the point on the growth curve where the flux values stabilize or show minimal change.

### C. Magnitude Calibration

Calibration of magnitude involves obtaining a set of reference stars with known magnitudes in the same field of view. We made use of standard values of magnitude from Simbad.

SIMBAD (the Set of Identifications, Measurements and Bibliography for Astronomical Data) is an astronomical database of objects beyond the Solar System. As of 1 June 2020, SIMBAD contains information for 5,800,000 stars and about 5,500,000 non stellar objects (galaxies, planetary nebulae, clusters, novae and supernovae, etc.)

Calibrating magnitude for each night is important in photometric analysis for accurate measurements due to variations in atmospheric conditions, instrumental parameters, sky background, and photometric zero point. It ensures consistency and reliable data across different observation sessions, especially for long-term monitoring.

Below is the detailed explanation of important lines in the simbad query code.

```
Simbad.reset_votable_fields()
Simbad.add_votable_fields('flux(R)', 'flux_error(R)')
```

These lines configure the Simbad query by resetting the default votable fields and adding specific fields of interest. In this case, the fields 'flux(R)' and 'flux_error(R)' are added, which correspond to the flux magnitude and flux magnitude error in the R-band.

```
for ra, dec in star_coordinates:
    coords = SkyCoord(ra=ra, dec=dec, unit=u.deg, frame='icrs')
    result_table=Simbad.query_region(coords, radius=5*u.arcsec)
    if result_table is not None:
        magnitude = result_table['FLUX_R'][0]
        magnitude_error = result_table['FLUX_ERROR_R'][0]
        ra_list.append(ra)
        dec_list.append(dec)
        magnitude_list.append(magnitude)
        magnitude_error_list.append(magnitude_error)
    else:
        ra_list.append(ra)
        dec_list.append(dec)
        magnitude_list.append('Notfound')
        magnitude_error_list.append('Notfound')
```

This block iterates over the star_coordinates list, which contains tuples of RA and DEC coordinates. For each pair of coordinates, a SkyCoord object is created using SkyCoord(ra, dec, unit=u.deg, frame='icrs'). This object represents the celestial coordinates in the International Celestial Reference System (ICRS) frame.
The Simbad.query_region() function is then called, passing the coords and a search radius of 5 arcseconds. This performs a query to Simbad, searching for objects within the specified radius around the given coordinates.
If the query result (result_table) is not None, indicating that objects were found, the flux magnitude and flux magnitude error for the first object in the result are retrieved using the 'FLUX_R' and 'FLUX_ERROR_R' fields, respectively. The corresponding RA, DEC, magnitude, and magnitude error values are appended to the respective lists.
If the query result is None, indicating no objects were found, placeholders ('Notfound') are appended to the lists for the RA, DEC, magnitude, and magnitude error.

Later the code saves the retrieved magnitudes and errors to a new CSV file which is later used for plotting and thus getting calibration value. For example, in the below plot, intercept or calibration value is 22.24. Here we have not calibrated the slope as it is within an acceptable range of 10% error (only 8%).

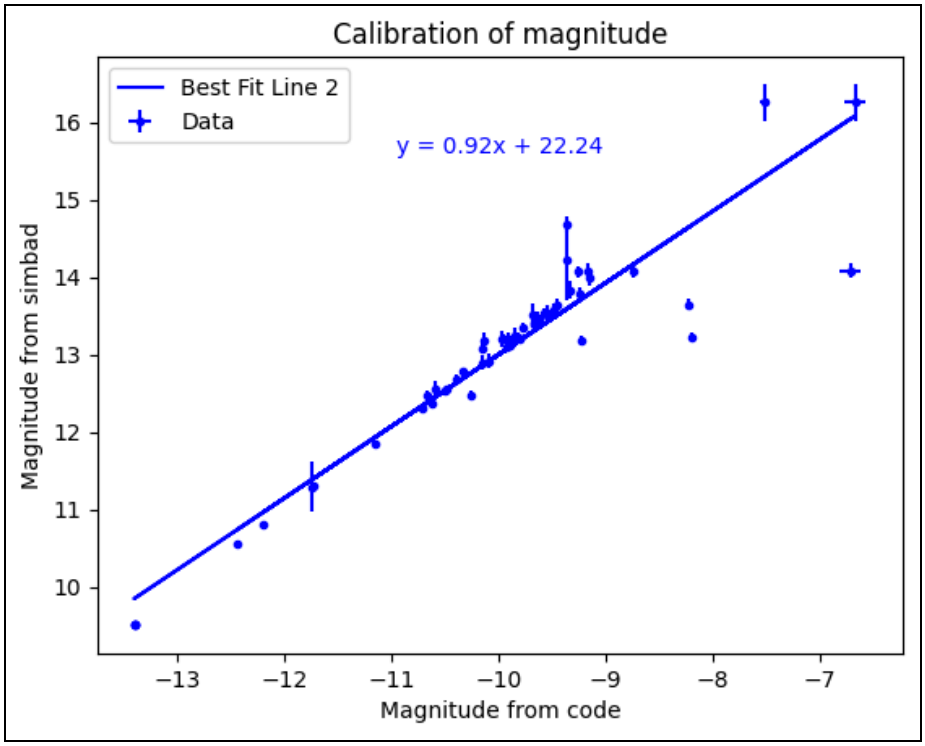


Figure 2: Best fit curve for magnitude values from SIMBAD against magnitude values from the code. An error bar for both the axes is also shown.

### D. Light curve

A light curve is a graph showing the changes in brightness of an object over time. It is crucial for studying variable stars, exoplanets, and transient events like supernovae. Light curves help identify variability, determine periods and changes in periods, probe exoplanets through transit observations, study stellar pulsations, and track transient phenomena. By analyzing the patterns in the light curve, astronomers gain insights into the nature, evolution, and physical properties of celestial objects.

In order to plot the light curve, we have used two methods. First method plots the light curve directly from the calibrated magnitude file generated for each frame each night. These files contain columns of JD, RA, DEC, calibrated magnitude and magnitude error of each star, where the JD column has the same values on each row as the data is from the same frame. Here each row has information about the star detected by aperture photometry in the frame. A file of a reference frame is first selected containing the maximum number of stars. The code then iterates over all other files and matches the RA DEC values of the reference file with other files. If the RA and DEC are matched within specified error, then the code plots the calibrated magnitude against JD of matched RA and DEC.

Second method involves making a transpose of the entire data. Firstly out of all frames from all nights, a frame with maximum number of stars is selected as a reference frame. This file is used for matching RA and DEC. This converts above files of frames to files of detected stars. These transposed files now contain columns of JD, Calibrated magnitude and magnitude error, while the rows contain information about the frame in which the star is detected. This method simplifies the task of plotting as we can directly plot JD and magnitude columns to get a light curve. We have also created a metadata file containing the filename of each star, along with its RA and DEC values. This makes it easy to get information about a particular star in the data of individual nights, and is later used to get period or phase.

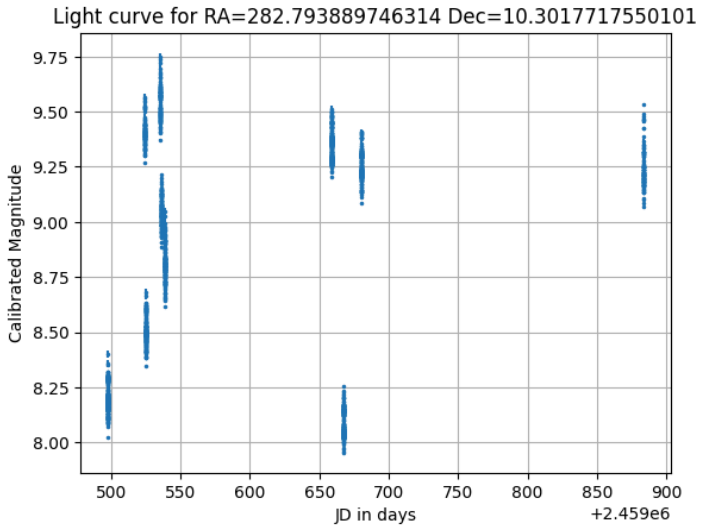


Figure 3: Light curves of a particular star over a period of 388 days. RA and DEC values of this star are mentioned on the plot

### E. Differential light curves

Once the data is transposed, differential light curves can also be easily plotted. We have done this only for stars with magnitude in particular range. For this, we have firstly filtered out the files of stars (generated after transpose) that have 90% of magnitude values lie in range ±1 of chosen value (say $x$) . This will select only those files whose at least 90% rows have value $x$ ±1 under calibrated magnitude column. Now, the code will select any 2 files from filtered files and perform subtraction of magnitude values corresponding to matching JD values; later it will calculate the standard deviation of this difference. This is done for all the possible combinations of selecting 2 files from all the filtered files. Now, the files for which this standard deviation is least or minimum are selected, and corresponding stars of these files are comparison stars (C1 and C2).
Now plotting the difference of magnitude between C1 and C2 against common JD values will give almost a straight horizontal line around 0. Selecting any file, say T, from filtered files (other than C1 and C2) and plotting T - C1, T-C2, C1-C2 against common JD will give differential light curves.

### F. Phase - Magnitude diagrams

We have used the Lomb-Scargle algorithm to get the period of variability. This method is a widely used method in astrophysics for analyzing unevenly sampled time series data. It is particularly useful for detecting periodic signals in phenomena like variable stars and exoplanet transits. The algorithm involves preprocessing the data to remove biases, creating a frequency grid, calculating the power associated with each frequency, and estimating the significance of the detected signals. By comparing the power values to the expected distribution for white noise, significant periodicities are identified. The algorithm generates a periodogram, which visualizes the power values across frequencies. This allows astronomers to identify peaks and candidate periods in the data. The Lomb-Scargle algorithm is a powerful tool for uncovering hidden periodic signals in irregularly sampled data, contributing to our understanding of various astronomical phenomena and their underlying mechanisms.

Next phase is also determined from the period obtained from Lomb-Scargle algorithm using formula:

$$phase = (time - t_0)/period - int((time - t_0)/period)$$

In this formula the *phase* represents the normalized phase of the signal at a specific time. Here, *time* is the specific time at which the phase is being calculated, *t_0* is the reference time or the starting point in the time series of data and *period* is the calculated period obtained from the Lomb-Scargle algorithm. Lastly, *int()* denotes the integer part of the division.

The amplitude of a variable star refers to the difference in brightness between its maximum and minimum values during a complete cycle. It is determined by analyzing a phase-magnitude curve, which plots the star's brightness against its phase. From the curve, the maximum and minimum magnitudes are identified, and the amplitude is calculated by subtracting the minimum magnitude from the maximum magnitude. The amplitude provides valuable information about the star's variability and can be used to classify different types of variable stars.

To get frequency and power we have used below code:

```
frequency, power = LombScargle(jd, magnitudes).autopower()
```

The LombScargle class is specifically designed for Lomb-Scargle periodogram analysis. It takes the Julian Date values (jd) and corresponding magnitudes (magnitudes) as inputs. The autopower() method is then called on the LombScargle instance. This method performs the Lomb-Scargle periodogram analysis and computes the power spectrum. The power spectrum represents the power associated with each frequency. It returns two arrays - frequency, an array containing the frequencies at which the periodogram was computed and power, an array

containing the corresponding power values. Each frequency corresponds to a potential period of variation in the data and higher power values indicate stronger signals or periodicities at those frequencies.

Below code gives period and amplitude values:

```
dominant_period = 1 / frequency[np.argmax(power)]
phase = ((jd - jd[0]) / dominant_period) % 1
```

This line finds the period by determining the frequency with the highest power (np.argmax(power)) and calculating its reciprocal to obtain the period (1/frequency[np.argmax(power)]).

Next, the phase for each data point is calculated. The phase represents the fractional position of each measurement within the period. It is computed by subtracting the initial Julian Date value (jd[0]), dividing by the period, and taking the modulo 1 to ensure that the phase remains within the range [0, 1].

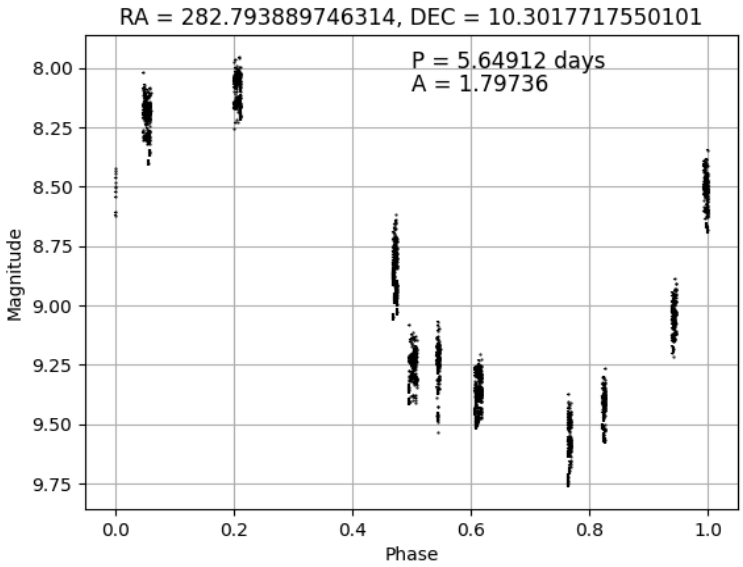


Figure 4: Phase - Magnitude curve for light curve of star plotted previously. Period (P) and amplitude (A) is also mentioned in the plot.

## References


1) Photutils documentation for aperture photometry: https://photutils.readthedocs.io/en/stable/aperture.html
2) SIMBAD Query: https://simbad.u-strasbg.fr/simbad/sim-fcoo
3) Astrometry: https://astroquery.readthedocs.io/en/latest/astrometry_net/astrometry_net.html