

# Analyzing Light Curves for Variable Star Classification

Melinda Soares-Furtado<sup>1</sup>, Christopher Moore<sup>2</sup>, and Rachel McClure<sup>2</sup>

<sup>1</sup>Melinda Soares

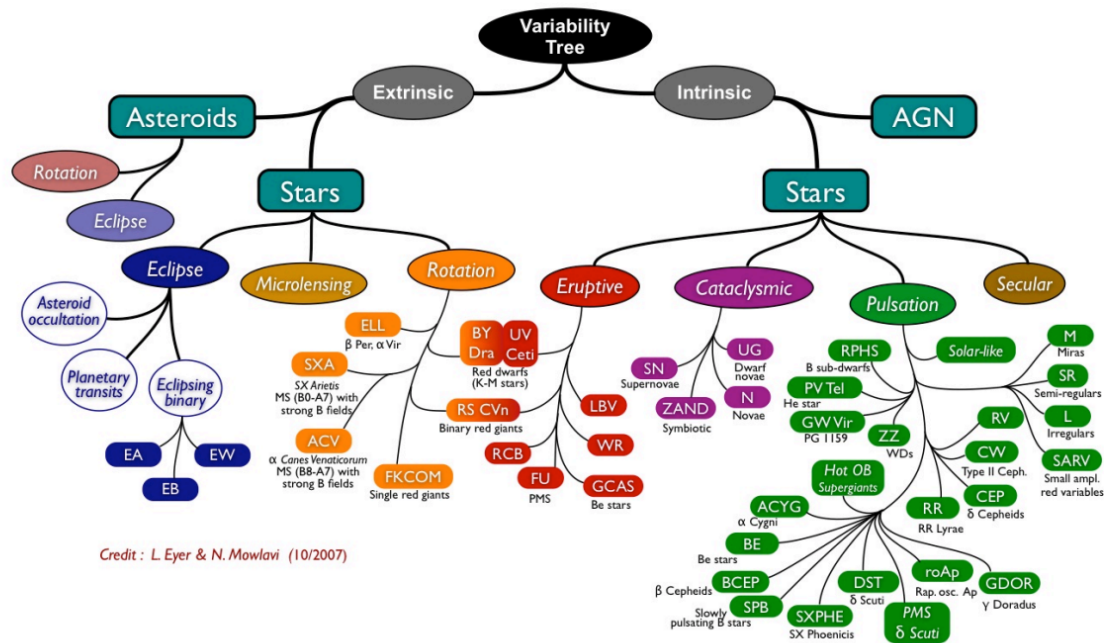
<sup>2</sup>University of Wisconsin – Madison

September 16, 2020

Today we will think about analyzing stars that are photometrically variable.

*What is the range of astrophysical phenomena are we talking about here?*

- extrinsic vs. intrinsic
- periodic vs. aperiodic



Credit : L. Eyer & N. Mowlavi (10/2007)

Figure 1:

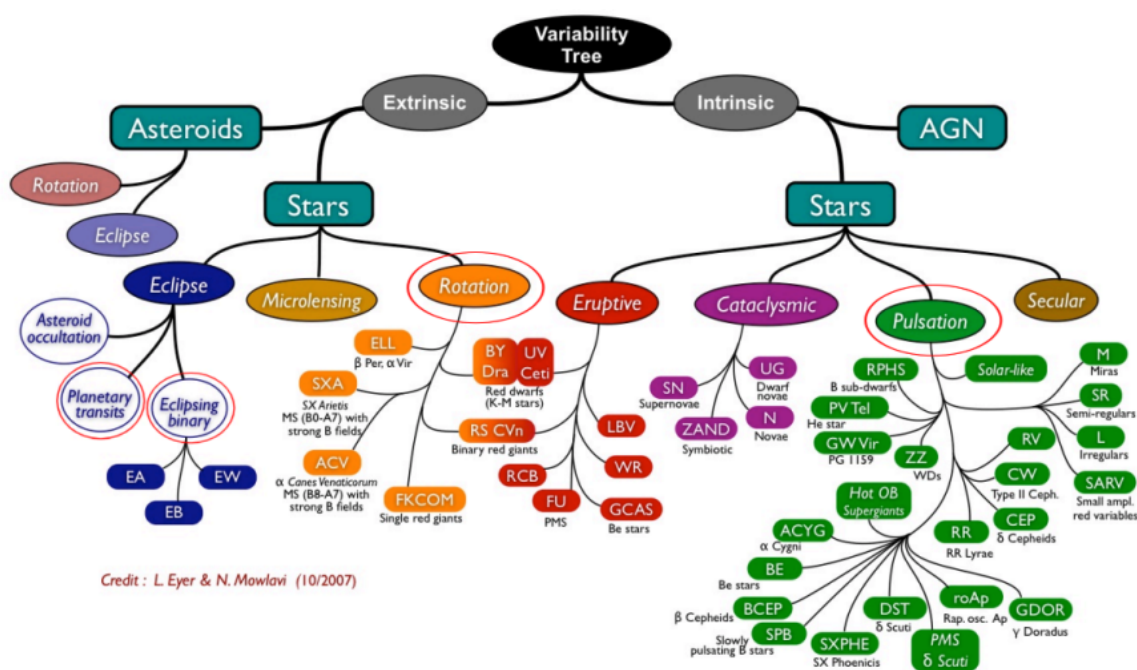


Figure 2: The nodes of interest in this study

Summary:

We can find rotational variables, pulsational variables, eclipsing binaries, and transiting planets using the techniques I've implemented into the past.

None of the aperiodic sources have been analyzed in my open cluster fields. There are likely interesting sources therein.

**Our variable classification starting point is the light curve.**

***What is a light curve?***

A light curve is a graph of light intensity of a celestial object or region, as a function of time.

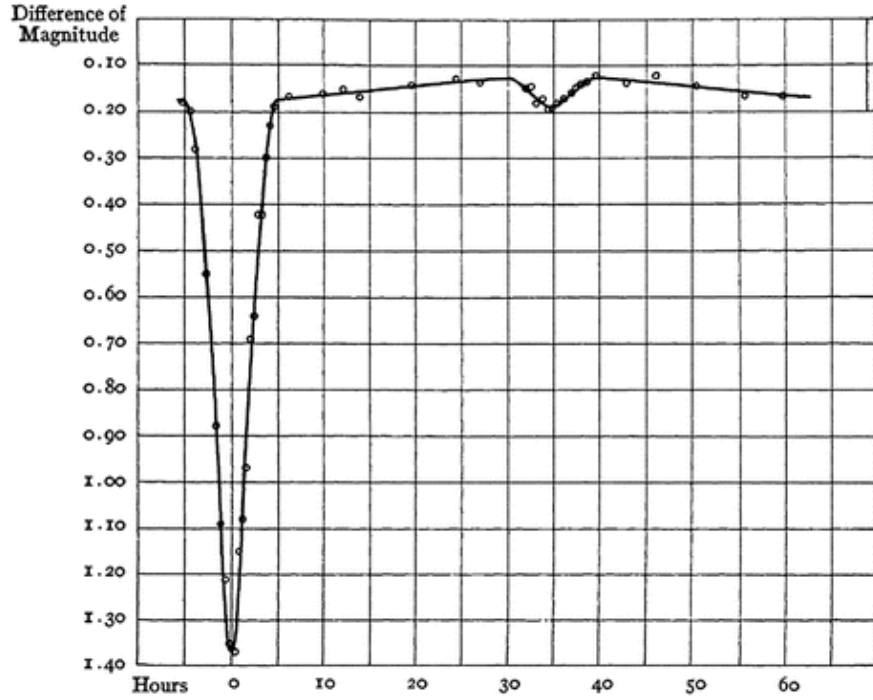


Figure 3: Light curve of the naked-eye variable Algol by Joel Stebbins in the *Astrophysical Journal*, vol. 32, p. 185, 1910.

We are considering the light output from stars. The readout generally provides information regarding the flux of an object.

Light curves are high-level science products, meaning that they've been augmented by an analyst from the original, raw science product. Bear in mind that the light curves will come in different formats (normalized flux, normalized mag, mag, flux, [BJD](#), cadence).

$$Flux = 10^{-0.4(Mag)}$$

$$Mag = -2.5 \log_{10}(Flux)$$

When you have a light curve, sometimes your eye can see repeated patterns in it. The eye is a really wonderful tool for spotting patterns. Sometimes your eye can be fooled, however. Things that seem periodic might not be. Similarly, it may seem like you have a noisy light curve, while you are instead sampling many repeated cycles of a periodic variation. This is why we also use mathematical tools in our efforts to classify variables. We'll come back to these tools in just a bit. I first want to introduce the concept of phase-folding.

### What is the Julian Date and the BJD?

The Julian day is the continuous count of days since the beginning of the Julian Period and is used primarily by [astronomers](#), and in [software](#) for easily calculating elapsed days between two events. The Julian Day Number is the integer assigned to a whole solar day in the Julian day count starting from noon [Universal time](#), with Julian day number 0 assigned to the day starting at noon on Monday, January 1, [4713 BC](#). It was proposed by J. J. [Scaliger](#) in 1583, so the name for this system derived from Julius [Scaliger](#), not Julius Caesar.

The Barycentric *Julian Date* (*BJD*) is the *Julian Date* (JD) corrected for differences in the Earth's position with respect to the barycentre of the Solar System.

## What is a phase-folded light curve?

With a preselected period for your light curve, you can “fold” the data. This allows you to plot the brightness as a function of phase. By folding the data points at this period, the light curve of the variable star is built up. Poorly determined periods will be inherently more scattered about the average.

Figure 4: *Observations of Algol made and submitted by John Isles. It is printed in Chapter 11 of the [Hands-On Astrophysics](#) manual.*

## How do you determine the period?

For rare systems, like Algol, the variation is observable to the naked-eye. Algol was seen to brighten and dim over the course of almost three days and was one of the first variable stars detected. For most variables, the fluctuations are too dim to be seen by eye.

There is a huge range of variability timescales. Some variables fluctuate on timescales of a few seconds (e.g., X-ray bursters), while others may take several years (eruptive T Tauri stars). It’s important to know your period range of interest, which is dependent upon two things (a) the integration time — you cannot search for periods less than the integration time and (b) the total observation window.

$$P_{\{\max\}} \leq \frac{1}{3}\tau$$

In order to look for periodic signals, there are several mathematical tools. A periodogram is one of them. In signal processing, a periodogram is an estimate of the spectral density of a signal. Periodograms are super handy in determining the optimal period of a time series.

A periodogram is similar to the Fourier Transform but is optimized for unevenly time-sampled data, and for different shapes in periodic signals. This is handy, as unevenly sampled data is particularly common in astronomy. For instance, your target might rise and set over several nights or you have to stop observing with your spacecraft to download the data. Moreover, time-series data of astrophysical objects are inherently noisy measurements – photon noise, atmospheric conditions and other factors can introduce random variation into the magnitude of the observations.

Like a Fourier Transform, a periodogram calculates the significance of different frequencies in time-series data to identify any intrinsic periodic signals. A periodogram is a brute-force tool. Many different frequencies and candidate periodic signals are evaluated for a given light curve. The statistical significance of each frequency is computed based upon a series of algebraic operations that depend on the particular algorithm used and periodic signal shape assumed. The brute-force nature of periodograms makes their computation time intensive on single-CPU systems, particularly as the number of observations and periods sampled grow. For example, on my Macbook workstation a periodogram for a 30-day light curve can take up to 20 minutes on a single CPU.

The brute-force nature of periodograms also makes the algorithms readily available to parallelization. The NASA Exoplanet Archive has optimized its algorithms to be fully parallelized across a 128-core cluster. If we do not get a server/workstation set up, we may want to consider submitting our jobs to this platform. Computation times can be reduced to less than a minute for time-series data with hundreds of thousands of data values.

<https://exoplanetarchive.ipac.caltech.edu/cgi-bin/Pgram/nph-pgram>

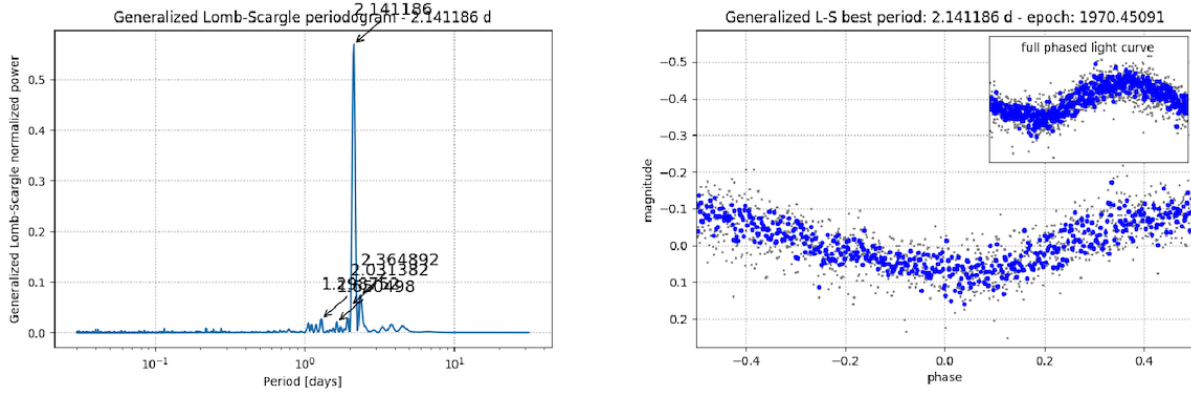


Figure 5: Periodogram results (Lomb Scargle) for an arbitrary Cepheid variable

## Types of Periodograms

Periodograms come in many flavors and it is important to know the strengths and weaknesses of those that you use. Different periodogram algorithms are sensitive to different periodic signal shapes, and can be used in tandem as a powerful complementary analysis technique.

So far we have employed three basic types:

1. Box Least Square Periodogram – optimized to detect periodic transits, by fitting the time series to a repeating “box”-shaped light curve.
2. Generalized Lomb Scargle Periodogram – an approximation of the Fourier Transform for unevenly spaced time sampling. It identifies periodic signals that are simple combinations of sines and cosines.
3. Phase Dispersion Method Periodogram (aka Plavchan Method) – binless phase-dispersion minimization algorithm that identifies periods with coherent phased light curves (i.e., least “dispersed”). There is no assumption about the underlying shape of the periodic signal.

Let’s continue our focus on the overall picture and avoid getting into the weeds. We will come back to how these periodograms work later.

## How do you classify a given source?

Some characteristics to look for:

- light curve shape
- period
- depth of transit (EBs & planets) or amplitude of variation (rotational/pulsational variables)
- stellar parameters: color, magnitude, location in the HR diagram, cluster membership, chemical abundance

Let’s discuss these parameters one by one.

### Light Curve Shape

Variables often have signature light curve shapes. Eclipsing binaries are one such example that we will explore today.

Let's look at the light curve of the detached eclipsing binary, Algol.

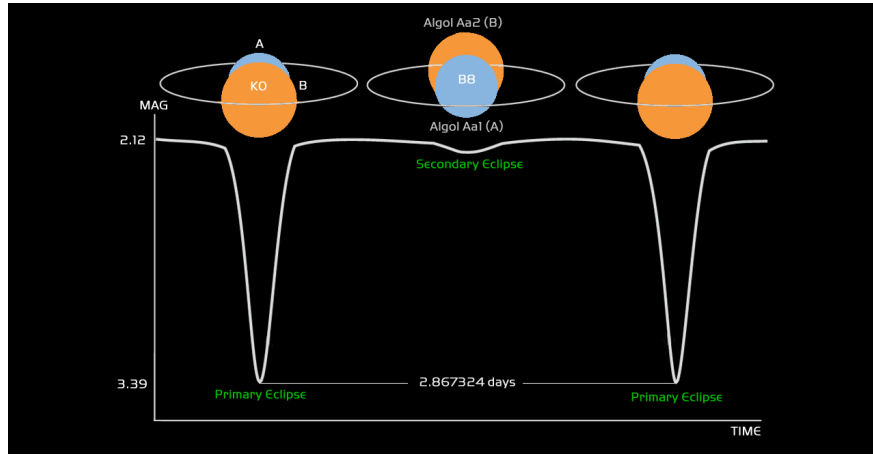


Figure 6: Light curve of the Algol variable system

Eclipsing binaries (EBs) do not look all the same, however. Their light curve shape very much depends on the architecture of the system.

Let's look at three major groups: (a) detached, (b) semi-detached, (c) contact binary.



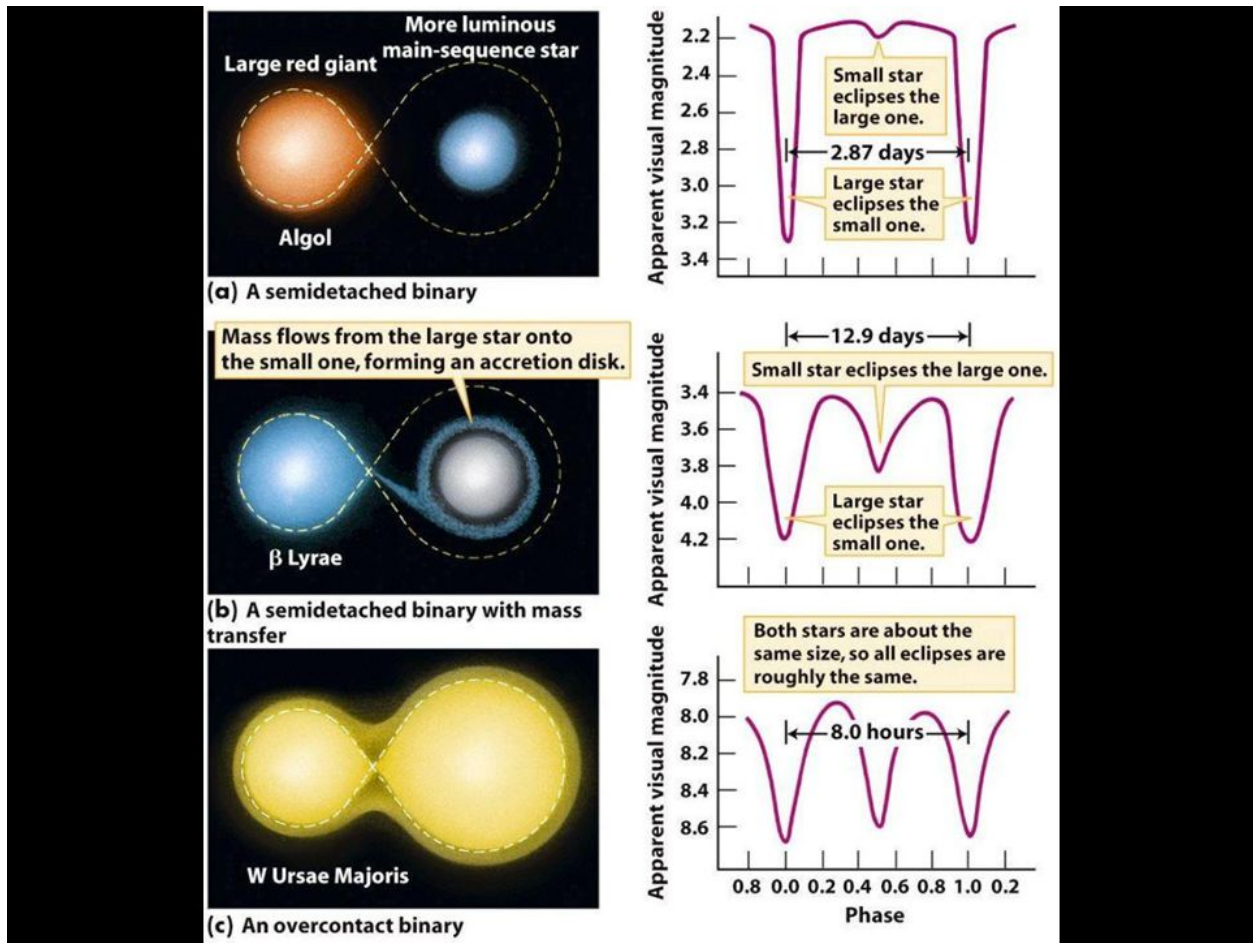


Figure 7: 3 Main Types of Eclipsing Binaries

There are also partial eclipsing binaries, which exhibit a more sharp, v-shaped light curve.

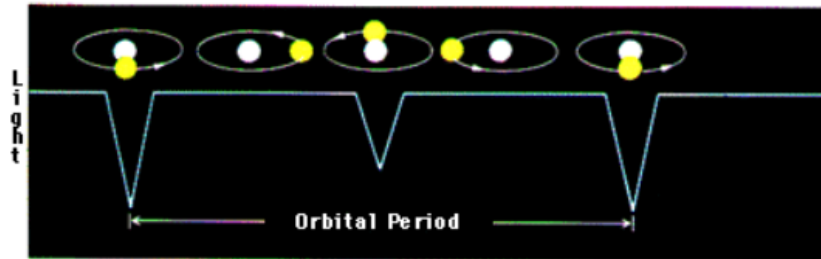
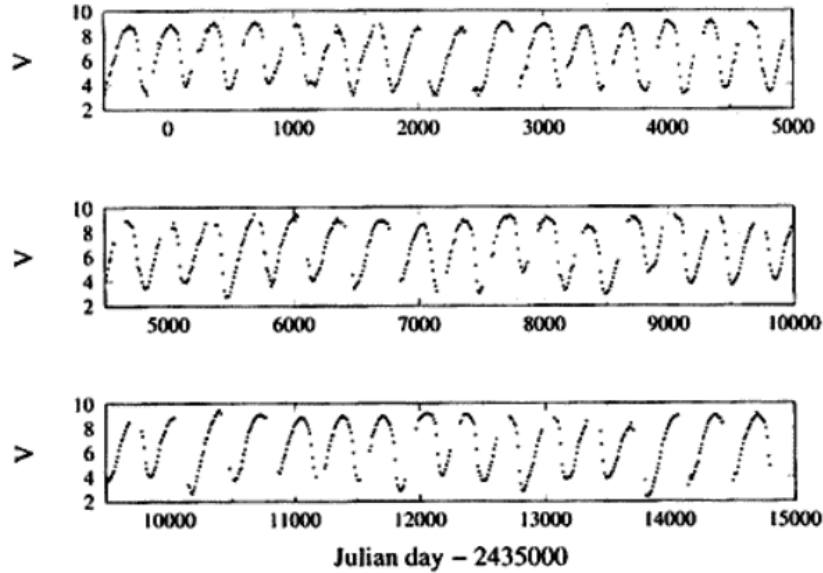


Figure 8: This is a caption

## Pulsational Variables

[https://faculty.virginia.edu/ASTR5610/lectures/stellar\\_evolution/pulsational.html](https://faculty.virginia.edu/ASTR5610/lectures/stellar_evolution/pulsational.html)

Figure 9: This is a caption



**Figure 14.1** The light curve of Mira from 1953 to March 1995. The time is measured in **Julian days**; Julian day 0 started at noon (UT) on January 1, 4713 B.C., and JD 2435000 is September 14, 1954. (Courtesy of Janet A. Mattei, AAVSO Director.)

From Carroll & Ostlie.

## Rotational Variables

### How are variables officially classified?

Resources like the [AAVSO International Variable Star Index \(VSX\)](#) describe the many categories of variables (periodic and aperiodic). Many groups can be further divided into subgroups. For example, a Delta Cepheid is a kind of pulsational variable, which can be further subdivided into the groups: [DCEPS](#) and [DCEPS\(B\)](#). I recommend referring to these resources to build your understanding of the many types of variable stars.

**\*\*stop here\*\***

### Creating the Light Curves

This is actually a separate post. As soon as my account is upgraded this content will move to a separate document. This describes the generation of the image-subtracted, trend-filtered, light curves from start to



finish.

## 1. Determine the channel and quarters that you will need for your cluster of choice

- This is slightly trickier than it sounds. The *Stellar Cluster Project* is a special component of the Science Program. Data can be identified on the Kepler Data Search and Retrieval page, using the investigation ID = STC, as shown in the figure below. We will use long cadence data only.
- Sort and compare the RA/DEC listed in the data to that of your chosen cluster. Write down the Sky Group ID (this should not change with quarter or channel). Also write down the channel corresponding to each of the quarters.

The screenshot shows the MAST Kepler Data Search & Retrieval page. The page has a navigation bar with links like MAST, STScI, Tools, Mission Search, Search Website, Follow Us, Register, and Forum. Below the navigation bar, there's a notice about the transition to HTTPS. The main section is titled 'Kepler Data Search & Retrieval' and includes a 'Standard Form' and a 'File Upload Form'. The search form contains several input fields: Target Name, Right Ascension, Declination, Kepler ID, Investigation ID (with 'STC' entered), 2MASS ID, Target Type (with 'Long Cadence' selected), Release Date, Teff, Log G, Quarter, and Condition Flag (with 'All Targets' selected). There are also sections for User-specified fields and Field Descriptions.

Figure 10: Accessing Kepler/K2 stellar cluster data

## 2. Determine what cadence numbers you'll need

What is a cadence? – The cadence is simply an integer corresponding to a particular observation. With Kepler, the cadences may be *short* or *long* depending upon the integration time. We use only the long, 30-minute cadences.

Cadence numbers can be found in the links below.

page 8: [https://archive.stsci.edu/kepler/release\\_notes/release\\_notes25/KSCI-19065-002DRN25.pdf](https://archive.stsci.edu/kepler/release_notes/release_notes25/KSCI-19065-002DRN25.pdf)

page 7: [https://archive.stsci.edu/missions/kepler/docs/drn/release\\_notes02/DataRelease\\_02\\_Notes\\_2009102213.pdf](https://archive.stsci.edu/missions/kepler/docs/drn/release_notes02/DataRelease_02_Notes_2009102213.pdf)

Below I list the quarters, channels, cadences, and the total number of cadences for the cluster NGC 6819. Note that there are no data for this cluster in quarters 6, 10, and 14.

Q	Ch	Cadences	NumCad
0	50	568-1043	476
1	50	1105-2743	1639
2	6	2965-7318	4354
3	34	7404-11773	4370
4	78	11914-16310	4397
5	50	16373-21006	4634
7	34	25509-29883	4375
8	78	30657-33935	3279
9	50	34237-39004	4768
11	34	43667-48420	4754
12	78	48473-52516	4044
13	50	52551-56971	4421
15	34	61886-66665	4780
16	78	66712-70914	4203
17	50	70976-72531	1556

### 3. Generate an output file containing a URL list of the TPFs associated with each of the channels and quarters

- To do this next step, I use [k2mosaic](#). We will first generate a list of all the TPFs required for a particular quarter and channel. It's important that these data live somewhere where there is (a) lots of space — we need about 5 TB per cluster — and (b) in directories that are well-organized. I create a directory called NGC6798 that lives on my external HD. Within NGC6798 I make a directory called k2mosaic. That is where all the k2mosaic output will live. Within k2mosaic I create a folder for each of the quarters and corresponding channels.
- Quarter 0, Channel 50 is located in the following folder: /Volumes/Seagate\_Expansion/NGC6819/k2mosaic/q00ch50
- Similarly, Quarter 3, Channel 34 is therefore located in the following folder: /Volumes/Seagate\_Expansion/NGC6819/k2mosaic/q03ch34
- One by one, cd into the folder corresponding to quarter and channel and generate the tpf list containing the URL for each of the TPFs associated with that quarter and channel. This can be done using [k2mosaic](#) by performing the following command: **k2mosaic tpflist Q0 50 > tpflist.out**



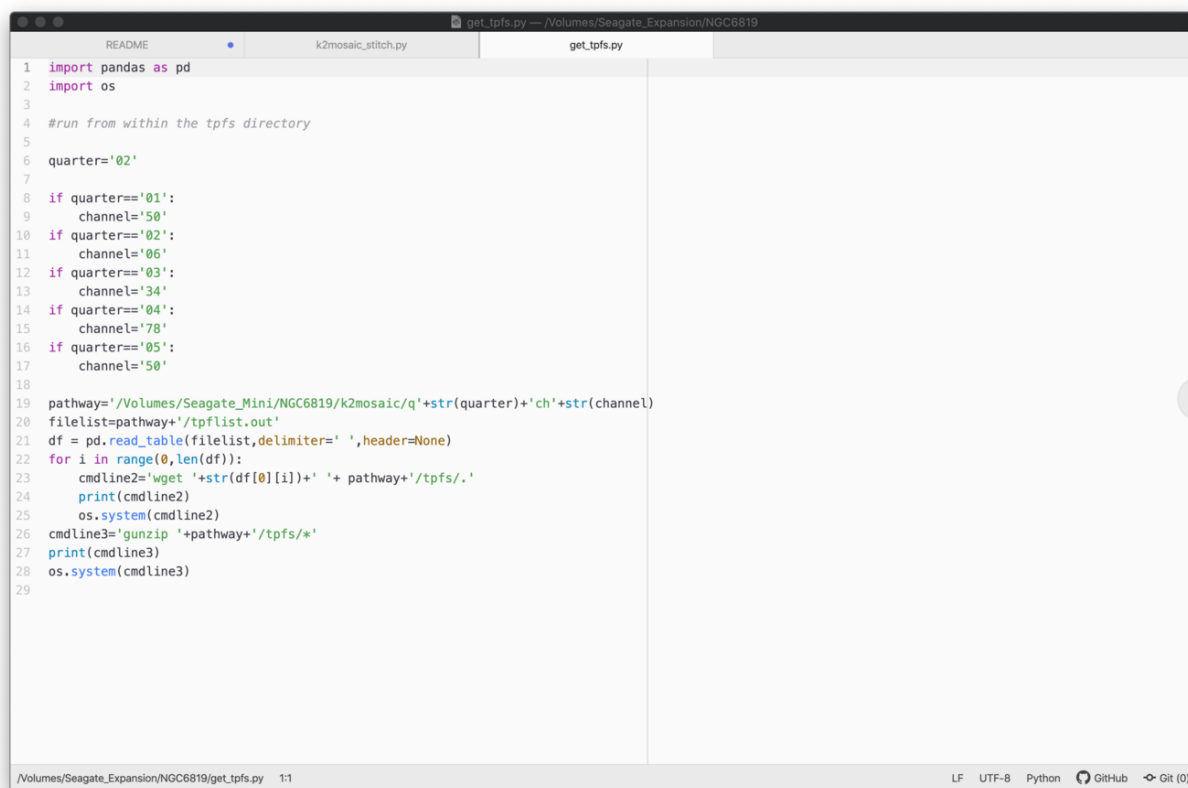
```
q00ch50 — more tpflist.out — 80x24
https://archive.stsci.edu/missions/kepler/target_pixel_files/0045/004572911/kplr
004572911-2009131105131_lpd-targ.fits.gz
https://archive.stsci.edu/missions/kepler/target_pixel_files/0046/004665509/kplr
004665509-2009131105131_lpd-targ.fits.gz
https://archive.stsci.edu/missions/kepler/target_pixel_files/0046/004665808/kplr
004665808-2009131105131_lpd-targ.fits.gz
https://archive.stsci.edu/missions/kepler/target_pixel_files/0046/004665816/kplr
004665816-2009131105131_lpd-targ.fits.gz
https://archive.stsci.edu/missions/kepler/target_pixel_files/0046/004665859/kplr
004665859-2009131105131_lpd-targ.fits.gz
https://archive.stsci.edu/missions/kepler/target_pixel_files/0046/004666185/kplr
004666185-2009131105131_lpd-targ.fits.gz
https://archive.stsci.edu/missions/kepler/target_pixel_files/0046/004666248/kplr
004666248-2009131105131_lpd-targ.fits.gz
https://archive.stsci.edu/missions/kepler/target_pixel_files/0047/004756681/kplr
004756681-2009131105131_lpd-targ.fits.gz
https://archive.stsci.edu/missions/kepler/target_pixel_files/0047/004756746/kplr
004756746-2009131105131_lpd-targ.fits.gz
https://archive.stsci.edu/missions/kepler/target_pixel_files/0047/004756864/kplr
004756864-2009131105131_lpd-targ.fits.gz
https://archive.stsci.edu/missions/kepler/target_pixel_files/0047/004756934/kplr
004756934-2009131105131_lpd-targ.fits.gz
https://archive.stsci.edu/missions/kepler/target_pixel_files/0047/004757017/kplr
tpflist.out
```

Figure 11: Example contents of a tpflist.out file

#### 4. Download and Unzip the TPFs listed in each of the URLs

Now we will download and unzip each of these listed URLs to generate a local copy of the desired data. I wrote a script to do this called `get_tpf.py`. The contents of this file are shown in the figure below. You can see that I'm not yet done, as it stops at quarter 5. Note that I place all the TPFs in a folder called `tpfs`. This is to keep things organized, which will be important going forward. Run this script from within the `tpfs` folder.

For example, for Quarter 2, Channel 6 I would run from within the following pathway: `/Volumes/Seagate_-Expansion/NGC6819/k2mosaic/q02ch06/tpfs`



```

1 import pandas as pd
2 import os
3
4 #run from within the tpfs directory
5
6 quarter='02'
7
8 if quarter=='01':
9     channel='50'
10 if quarter=='02':
11     channel='06'
12 if quarter=='03':
13     channel='34'
14 if quarter=='04':
15     channel='78'
16 if quarter=='05':
17     channel='50'
18
19 pathway='/Volumes/Seagate_Mini/NGC6819/k2mosaic/q'+str(quarter)+'ch'+str(channel)
20 filelist=pathway+'/tpflist.out'
21 df = pd.read_table(filelist,delimiter=' ',header=None)
22 for i in range(0,len(df)):
23     cmdline2='wget '+str(df[0][i])+' '+pathway+'/tpfs/.'
24     print(cmdline2)
25     os.system(cmdline2)
26     cmdline3='gunzip '+pathway+'/tpfs/*'
27     print(cmdline3)
28     os.system(cmdline3)
29

```

Figure 12: Python script to download and unzip all TPFs

## 5. Create a new tpf list pointing to these unzipped files

This is done with a quick command from the terminal.

```
ls tpfs/*.fits > tpflist_downloaded.out
```

## 6. For a given quarter and channel, stitch together TPFs for each of the cadences

Because of memory issues experienced by both myself and R. McClure, we were stitching the cadences one-by-one using a python script. This is in contrast to the bulk cadence stitching performed in the example [here](#). We can ignore that now that we have done steps 4 & 5.

Simply perform the following to stitch your tpfs:

```
k2mosaic mosaic tpflist_downloaded.out
```

This is about an order of magnitude faster than our previous work-around.

## 7. For a given quarter,

