Workflow for Stellar Flare Lightcurve Analysis

Nicole Loncke

April 14, 2014

1 Introduction

The Kepler Mission is designed to survey our galaxy in the hopes of discovering planets in or near the habitable zone of their stars and determine how many of the billions of stars in our galaxy have such planets. It essentially stares at a relatively small portion of the sky for a long time to gather brightness data about these stars.

In addition to planet detection, this data can be used to gather properties about the stars themselves. In this paper we are concerned with the flaring behaviors of these nearby stars and their effect on any orbiting planets.

2 Formatting

The lightcurves module makes some assumptions about the format of the input data. The original light curve data should be in a file containing a whitespace-separated table with time in the first column and flux in the second.

808.51470	*6338.22*
808.53514	*6340.73*
808.55557	*6346.89*
808.57601	*6341.10*
808.59644	*6340.22*
808.61688	*6340.61*
808.63731	*6342.13*
808.65774	*6349.23*
808.67818	*6343.68*
808.69861	*6334.51*
808.71905	*6337.67*
808.73948	*6348.09*

Table 1: Actual light curve data sampled from kid10068383.txt.

The other fundamental input file is the one containing the potential flare events, or “flags.” This file simply contains a column listing indices into the time array at the points that mark a suspected event.

350
351
352
370
371
372
373
374
375
549
550
551
552
553

Table 2: Actual flags sampled from the file corresponding to kid10068383.

Those two files — the light curve data and the flags — are all you need to start using these tools.

3 Plotting

If your aim is simply to generate numpy arrays of the lightcurve data — one array for time, another for mean-normalized flux — then use `ltcurve()`. This function takes as its primary argument a string of the name of the file containing the Kepler data and returns the time and brightness arrays. By default it also displays the light curve corresponding to the file on a time vs. brightness plot, but this feature can be switched off by passing the function an optional argument.

If you have *multiple* light curve files and would like to view them one at a time, then `ltcurves()` is more appropriate. Its only required argument is a list or array of filename strings. Note that this function does not return any of the data. In addition, if for each of the Kepler data files you have a set of corresponding event flags[1], you may use the `flags` kwarg to overplot the potential flares.

4 Vetting

Instead of cycling through the light curves with overplotted flags, you may find it helpful to inspect and record whether or not the marked events could potentially be stellar flares. In that case, you should use `flareshow()`, which writes user input (either 'y', 'n', 'm') to two files for later retrieval. One file

[1]You can generate these flags using `getflags()` and passing it a list of the names of the files holding the flare flags.

contains a space-separated table of the Kepler IDs and the corresponding user responses to its events. The other file contains information about the length of each event. These two files work in conjunction to gather more information about the potential flares.

8848271	n
8908102	n
8953257	n n n n n n n n
9002237	n n n y

Table 3: Example output.txt file.

8848271	3735 03
8908102	1757 03
8953257	1454 6 1610 7 1890 4 2359 3 2516 4 2829 5 2985 6 3265 5
9002237	3337 4 3547 5 3756 3 3967 4

Table 4: Corresponding example output_indices.txt file.

Note that before using `flareshow()`, you must have your flags in the proper format, generated by `getflags()`. This helper function outputs a nested list of event indices given a list of the names of the files containing the flags.

5 Data Processing

After evaluating the marked events by eye, quantifying data about the remaining candidates is the next step. Assuming that you used `flareshow()` for vetting, you now have two output files for the set of lightcurve data. Use the helper function `getEvents()`, which reads from these output files to pare down your list of flags to only those that have been marked with 'y' or 'm' depending on how you set the kwargs.

To calculate the cumulative brightness of each event found within a single light curve, use `intFlare()`. This function returns an array of the integrated brightness over the course of the events, an array of the duration of the events (in hours), and the peak brightnesses of each event. It is important to note that this function assumes you have vetted the flags already and are only providing those about which you would like to find more information (ie, you have used `getEvents()`).

6 Machine Learning

After creating the tools for by-eye vetting, the major bottleneck for data analysis was just that—as a human, vetting manually takes a lot of time. The flare detection program was trained to recognize some metrics as potenetial flares but didn't get things right 100% of the time. Our human brains allow us to do the same thing but with more nuance. If we could write a program to recognize the same patterns that humans so easily detect in the lightcurves then we could nearly entirely automate the vetting process.