Version history

version 2.0.7

This version provides several features to ease the processing of light curves that are timed with LED flashes from iPhones (John Grismore's AstroFlashTimer) or Android phones (Eric Couto's Occult Flash) rather than VTI timestamped files

- Adds a button to calculate the edge position of an LED timing flash.
- Adds a checkbox to enable/disable the tooltip messages that appear when a control is hovered over. Tooltip display defaults to 'enabled' because tooltips are an important aid for guiding users initially. Later, when such help is no longer needed, the user can turn them off (they are annoying when you don't need them).
- Adds the ability to select which light curve is to be analyzed. Previous versions would only analyze the first light curve for D and R events. This flexibility is useful in general, but was particularly needed to support LED flash timing.
- Adds a checkbox to force manual entry of timestamp info. This is useful when OCR on a VTI timed light curve has catastrophic errors. It is always employed when using LED flash timing.
- During the error bar calculation, it is possible for the Cholesky decomposition needed for treating correlated noise to fail. Previous versions treated this as a fatal error and would not produce a final report. This version instead treats the noise as uncorrelated and continues processing to produce a final report.

version 2.0.6

- Added additional instruction in the popup that appears when no timestamps are found in the csv file. This will give casual users additional guidance and clarification for the manual timestamp entry process.

version 2.0.5

- files generated by pyote now contain PYOTE in the filename.

- Timestamps can be corrupted to the point that a timeDelta of 0.0 can result. This version traps that event and reports it clearly --- 2.0.4 failed silently with a divide by zero exception

version 2.0.4

- improves the handling of errors during the reading of Tangra files by showing the offending line in the report panel. Tangra, if it has a tracking problem (i.e., loses it) will emit an empty field for that measurement, leaving it up to the user to decide how to fill in the missing value. Prior pyote versions simply reported 'format error' without providing a printout of the offending line. This version fixes that.

version 2.0.3

- detects and handles situations in which fewer than 14 baseline points are available for calculation of correlated noise coefficients. When fewer than 14 points are available, the correlation coefficients are set to: [1, 0, …] (i.e., coefficients are set to 'no correlated noise')

version 2.0.2

- Note: this version has many significant changes. If you lose confidence in this version, remember that you can always go back to version 1.47 by typing ---

  pip install pyote==1.47

  in an Anaconda console. (Be sure to use double == signs in the command.)

- improved handling of D and R region selection so that one cannot enter an invalid configuration --- automatic corrections/changes are applied.

- incorporates a new 'solver' that no longer requires an initial estimation of baseline noise. This

'solver' is also much faster. With this 'solver', the two-pass modification added in version 1.46 is no longer needed.

- removes unneeded 'analyze noise' buttons and rearranged other buttons to be in-line rather than one above the other to allow the vertical splitter between the plot area and report area more room to change (a help to those using screens with relatively low pixel densities).

version 1.47

- adds bold red highlighting to message:
  ! There is something wrong with timestamps at D and/or R or frames have been dropped !
  so that it is harder to miss.

version 1.46

- adds automatic recalculation of baseline and event noise parameters utilizing all available data points during a second solution pass; this removes the variability in calculated error bars due to user selection of a necessarily less complete set of data points for noise analysis during the first solution pass.

- adds bold blue text in the 'Excel' portion of the final report to indicate whether or not the light curve was block integrated, trimmed, or normalized. Failing to block integrate a light curve that needed it is a common error. Highlighting the presence or absence of block integration in the most looked at portion of the final report will hopefully help reduce the number of such errors.

Version 1.45

- the initial fully functional release of pyote.

Introduction to *pyote*

Bob Anderson (bob.anderson.ok@gmail.com)

pyote is an occultation timing extraction utility program written primarily in python and distributed through PyPI (the python package repository).

This program is specifically designed for those who will use such a program infrequently; it has been designed to the best of my ability to produce consistent results in the hands of both infrequent and frequent users --- the same results should be obtained no matter who processed the data.

One important feature of the program intended to give confidence to the occasional user is the production of a log file that documents all processing steps/decisions made in sufficient detail that anyones result can be reviewed by more experienced users easily --- it is sufficient to simply send such a reviewer just two things: the light curve and the log file.

1. *pyote* is designed for ease-of-use in the analysis of so-called square wave occultation light curves (defined as occultation recordings that exhibit no detectable diffraction effects). Such light curves are common with star/asteroid occultations when the star is effectively a point source and the asteroid transit speed is such that the disappearance/reappearance events occur much faster than the frame rate of the video recorder.

2. Correlated noise caused by atmospheric scintillation is frequently present in occultation observations recorded at normal video rates of 25 or 30 frames per second. *pyote* utilizes statistically rigorous calculations to properly characterize the increased uncertainty in D/R time estimates due to such correlated noise.

3. Physically realistic models are fit to the light curves with all decisions about details (complexity) of the model used made using the Akaike Information Criterion (AIC). In particular, an AIC calculation is always used to justify or reject sub-frame timing.

4. Maximum Likelihood Estimation is used throughout to determine 'best fit' of model light curves to the actual data.

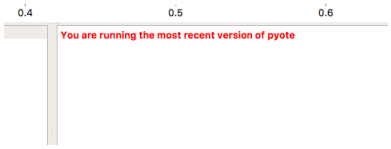
The gui for *pyote* is designed to lead the user through the necessary steps by enabling the buttons in sequence as each task is performed. So, initially, only two principal buttons are enabled: the 'info' button that brought up this document and the 'Read light curve' button. After reading this document, open a light curve, and follow the enabled buttons.

All of the major buttons have hover text associated. To learn (or refresh) how to use the program to analyze a light curve, spending a little time 'hovering' on the buttons will pay dividends.

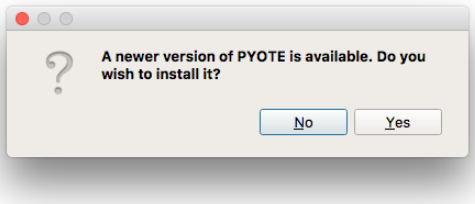
pyote will never change the input light curve, so experimentation is encouraged. There is a 'Start Over' button at the bottom that I encourage you to use freely.

Every step you make in the analysis is recorded in a log file. This is done because experience has shown that some light curves are touchy to analyze and it is useful to ask someone more experienced in running the program to look over your work. With the original light curve and a copy of the log file, your work can be exactly duplicated by someone else. And that log file is never deleted once it is opened for a particular light curve; it is simply appended to, so a record of each 'experiment' is thus always available.

Every time *pyote* is started, it connects to PyPI (assuming you have an internet connection) and checks to see if a more recent version of *pyote* has been added to the repository. If your version is completely up-to-date, you will see this



in the log file panel in the lower right-hand corner of the gui. Otherwise, this will appear:



Normally, you will want to click 'yes'. That will cause your current version of *pyote* to install (but not run) the newest version. Of course, to execute that new version, you will need to do a close and reopen.

As convenient as this is, there is always a small risk that a new version will actually 'break' something and that the 'cure' may take some time to be posted. But it is always possible to return to a specific previous version of *pyote.* The procedure to do this is explained below.

Open an Anaconda Prompt window if you are running Windows.

For a Mac installation, open a command window and type ***source activate***.

Then, type the following line in that command window:

1. pip install pyote==1.42

This command will uninstall the current (flawed) version of *pyote* and installs a specific version, in this case, version 1.42. Note the double == followed by the specific version number to be installed. (You can always determine a version of *pyote* that was working for you by opening a recent log file --- the *pyote* version that produced that log file is recorded there.)