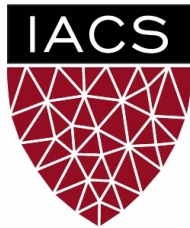


picture



Institute for  
**Applied Computational Science**  
HARVARD SCHOOL OF ENGINEERING AND APPLIED SCIENCE

## Feature Analysis for Time Series

**Authors:** Isadora Nun [isadoranun@g.harvard.edu](mailto:isadoranun@g.harvard.edu) , Pavlos Protopapas [pavlos@seas.harvard.edu](mailto:pavlos@seas.harvard.edu)

**Contributors:** Daniel Acuña, Nicolás Castro, Rahul Dave, Cristobal Mackenzie, Jorge Martinez, A Miller, Karim Pichara, Andrés Riveros, Brandon Sim and Ming Zhu

### Introduction

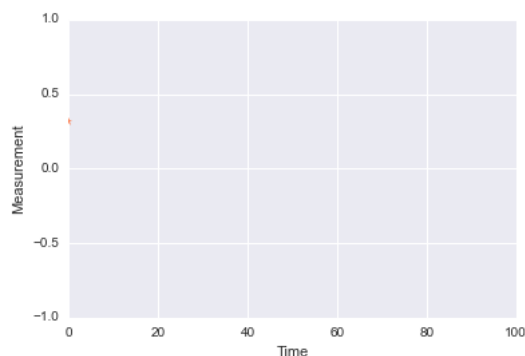
A time series is a sequence of observations, or data points, that is arranged based on the times of occurrence. The hourly measurement of wind speeds in meteorology, the minute by minute recording of electrical activity along the scalp in electroencephalography, and the weekly changes of stock prices and financials are just some examples of time series, among many others.

Some of the following properties may be observed in time series data [1]:

- the data is not generated independently
- their dispersion varies in time
- they are often governed by a trend and/or have cyclic components

The study and analysis of time series can have multiple ends: to gain a better understanding of the mechanism generating the data, to predict future outcomes and behaviors, to classify and characterize events, or more.

```
animation.FuncAnimation(fig, animate, init_func=init,
                        frames=100, interval=200, blit=True)
```



Once Loop Reflect

In time-domain astronomy, data gathered from the telescopes is usually represented in the form of light curves which are time series that show the brightness variation of an object through a period of time (a visual representation see video below). Based on the variability characteristics of the light-curve of celestial objects can be classified into different groups (quasars, long period variables, eclipsing binaries, etc.) and consequently can be studied in depth independently.

Classification of data into groups can be performed in several ways given light curve data: previously existing methods found in the literature use machine learning algorithms that group light-curves into categories through feature extraction from the light-curve data. These light-curve features, the top of this work, are numerical or categorical properties of the light-curves which can be used to characterize and distinguish the different variability classes. Features can range from basic statistical properties such as the mean or the standard deviation to more complex time series characteristics such as autocorrelation function. These features should ideally be informative and discriminative, thus allowing for machine learning or other algorithms to use them to distinguish between classes of light-curves.

In this document, we describe a library that allows for the fast and efficient calculation of a compilation of many existing light-curve features. The main goal is to create a collaborative and open tool where users can characterize or analyze an astronomical photometric database while also contributing to the library by adding new features. However, it is important to highlight that this library is not necessarily restricted to the astronomical domain and can also be applied to any kind of time series data.

Our vision is to be capable of analyzing and comparing light curves from any available astronomical catalog in a standard and universal way. This would facilitate and make more efficient tasks such as modeling, classification, data cleaning, outlier detection, and data analysis in general. Consequently, when studying light curves, astronomers and data analysts using our library would be able to compare and match different features in a standardized way. In order to achieve this goal, the library should run and features generated for every existent survey (MACHO, EROS, OGLE, Catalina, Pan-STARRS etc.), as well as for future surveys (LSST), and the results shared openly, as is this library.

In the remainder of this document, we provide an overview of the features developed so far and explain how users can contribute to the library. A Readme file is also available in case of needing more information.

*Video 1: Light-Curve of Triple Star*

The video below shows how data from the brightness intensity of a star through time results on a light curve. In this particular case we are observing a complex triple system in which three stars have mutual eclipses as each of the stars gets behind or in front of the others.

```
from IPython.display import YouTubeVideo
YouTubeVideo('qMx4ozpSRuE', width=750, height=360, align='right')
```

The following figure [5] presents example light-curves of each class in the MACHO survey. The x-axis is the modified Julian Date (MJD), and the y-axis is the MACHO B-magnitude.

```
picture = Image(filename='curvas_ejemplos11.jpg')
picture.size = (100, 100)
picture
```

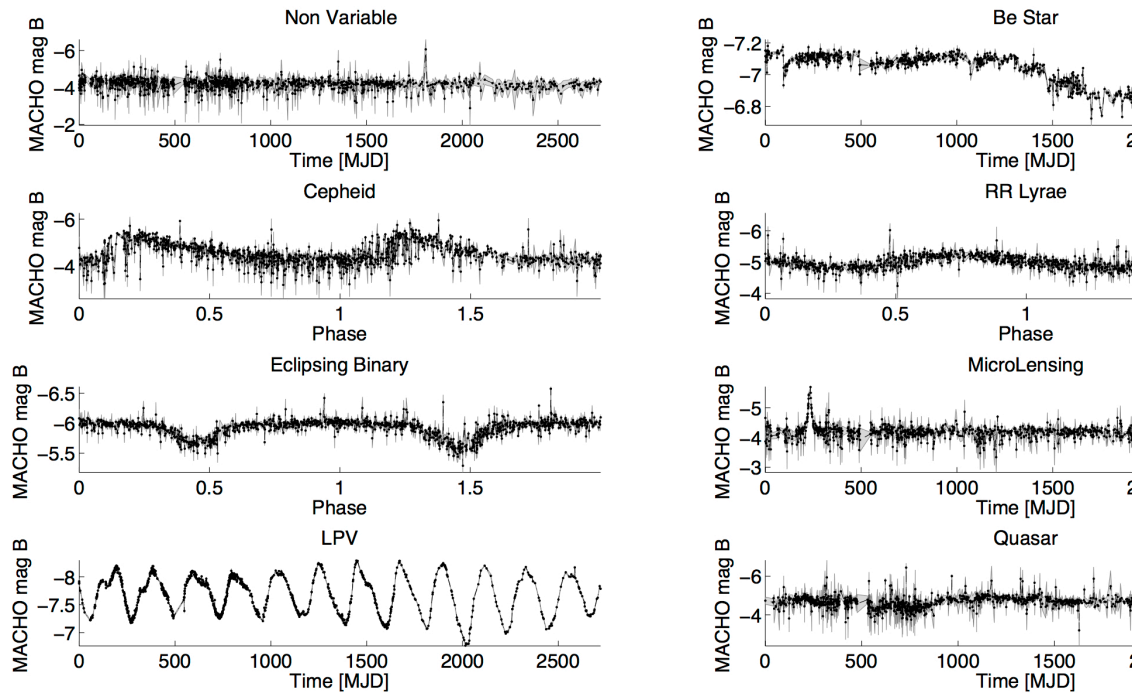


Figure 1: Some variability classes

## The library

### How to cite FATS

The user can cite this library by:

- making reference to the arXiv article available in <http://arxiv.org/abs/1506.00010>
- directly using the Bibtex entry in [http://adsabs.harvard.edu/cgi-bin/nph-bib\\_query?bibcode=2015arXiv150600010N&data\\_type=BIBTEX&db\\_key=PRE&nocookieset=1](http://adsabs.harvard.edu/cgi-bin/nph-bib_query?bibcode=2015arXiv150600010N&data_type=BIBTEX&db_key=PRE&nocookieset=1).

### Library structure

The library is coded in python and can be downloaded from the Github repos <https://github.com/isadoranun/FATS>. New features may be added by issuing pull requests via Github version control system. For a quick guide on how to use github <https://guides.github.com/activities/hello-world/>.

It is also possible to obtain the library by downloading the python package <https://pypi.python.org/pypi/FATS> or by directly installing it from the terminal as follows:

```
$ pip install FATS
```

The library receives as input the time series data and returns as output an array with the calculated features. Depending on the available input the user can calculate different features. For example, if a user has only the vectors magnitude and time, just the features that need this data will be able to be computed.

In order to calculate all the possible features the following vectors (also termed as raw data) are needed per light curve:

- **magnitude**
- time
- error
- magnitude2
- time2
- error2

where **2** refers to a different observation band. It is worth pointing out that the magnitude vector is only input strictly required by the library given that it is necessary for the calculation of all the features. The remaining vectors are optional since they are needed just by some features. In other words, if a user does not have this additional data or he is analyzing time series other than light curves, it is possible to calculate some of the features. More details are presented in the next section.

This is an example of how the input could look like if you have only magnitude and time as input vectors:

```
lc_example = np.array([magnitude_ex, time_ex])
lc_example
array([[ 0.46057565,  0.5137294 ,  0.70136533,  0.21454228,
         0.547923  ,  0.33433717,  0.4487987 ,  0.55571062,
         0.24388037,  0.44793366,  0.30175873,  0.88326381,
         0.12208977,  0.37088649,  0.5945731 ,  0.74705894,
         0.24551664,  0.36009236,  0.80661981,  0.04961063,
         0.87747311,  0.97388975,  0.95775496,  0.34195989,
         0.54201036,  0.87854618,  0.07388174,  0.21543205,
         0.59295337,  0.56771493],
       [ 0.,  1.,  2.,  3.,
        4.,  5.,  6.,  7.,
        8.,  9., 10., 11.,
       12., 13., 14., 15.,
       16., 17., 18., 19.,
       20., 21., 22., 23.,
       24., 25., 26., 27.,
       28., 29.]])
```

When observed in different bands, light curves of a same object are not always monitored for the same time length and at the same precise times. For some features, it is important to align the light curves and to only consider the simultaneous measurements from both bands. The aligned vectors refer to arrays obtained by synchronizing the raw data.

Thus, the actual input needed by the library is an array containing the following vectors and in the following order:

- magnitude
- time
- error
- magnitude2
- aligned\_magnitude
- aligned\_magnitude2
- aligned\_time
- aligned\_error
- aligned\_error2

The library structure is divided into two main parts. Part one: *Feature.py*, is a wrapper class that allows the user to select the features to be calculated based on the available time series vectors or to specify a list of features. Part two: *FeatureFunctionLib.py*, contains the actual code for calculating the features. Each feature has its own class with at least two functions:

**init:** receives the necessary parameters (when needed) for the feature calculation. It also defines the required vectors for the computation (e.g. magnitude and time).

**fit:** returns the calculated feature. The output can only take one value; features like the autocorrelation function must consequently be summarized in one single scalar.

The following code is an example of a class in *FeatureFunctionLib.py* that calculates the slope of a linear fit to the light-curve:

```
class LinearTrend(Base):
```

```
    def __init__(self):
        self.Data=['magnitude','time']

    def fit(self,data):

        magnitude = data[0]
        time = data[1]
        regression_slope = stats.linregress(time, magnitude)[0]

        return regression_slope
```

If the user wants to contribute with features to the library, they must be added to *FeatureFunctionLib.py* following the explained format. There is no need to modify *Feature.py*. The user should also add a validity test to the unit test file (see explanation [further down](#)).

## Choosing the features

The library allows the user to either choose the specific features of interest to be calculated or calculate them all simultaneously. Nevertheless, as already mentioned, the features are divided depending on the input data needed for their computation (magnitude, time, error, second data, etc). If unspecified, this will be used as an automatic selection parameter. For example, if the user wants to calculate all the available features but only has the vectors *magnitude* and *time*, only the features that need *magnitude* and/or *time* as an input will be computed.

**Note:** some features depend on other features and consequently must be computed together. For instance, *Period\_fit* returns the false alarm probability of the estimated period. It is necessary to calculate also the period *PeriodLS*.

The list of all the possible features with their corresponding input data, additional parameters and literature source is presented in the following table:

```
make_table(FeaturesList)
apply_theme('basic')
set_global_style(float_format='%0.3E')
```

Feature	Depends on feature	Input data (besides magnitude)	Parameter
Amplitude			
Anderson-Darling test			
Autocor_length			Number of
Beyond1Std		error	
CAR_mean	CAR_sigma	time, error	
CAR_sigma		time, error	
CAR_tau	CAR_sigma	time, error	
Color		mag2	
Con			Consecut
Eta_color		aligned_mag, aligned_mag2, aligned_time	
Eta_e		time	
FluxPercentileRatioMid20			
FluxPercentileRatioMid35			
FluxPercentileRatioMid50			
FluxPercentileRatioMid65			
FluxPercentileRatioMid80			
Freq1_harmonics_amplitude_0		time	
Freq{i}_harmonics_amplitude_{j}	Freq1_harmonics_amplitude_0	time	
Freq{i}_harmonics_rel_phase_{j}	Freq1_harmonics_amplitude_0	time	
LinearTrend		time	
MaxSlope		time	
Mean			
Meanvariance			
MedianAbsDev			
MedianBRP			
PairSlopeTrend			
PercentAmplitude			
PercentDifferenceFluxPercentile			
PeriodLS		time	Oversam

Period_fit	PeriodLS	time	
Psi_CS	PeriodLS	time	
Psi_eta	PeriodLS	time	
Q31			
Q31_color		aligned_mag, aligned_mag2	
Rcs			
Skew			
SlottedA_length		time	Slot size
SmallKurtosis			
Std			
StetsonJ		aligned_mag, aligned_mag2, aligned_error, aligned_error2	
StetsonK		error	
StetsonK_AC	SlottedA_length		
StetsonL		aligned_mag, aligned_mag2, aligned_error, aligned_error2	
VariabilityIndex			

The possible ways of how an user can choose the features from the library to be calculated are presented next.

#### - List of features as an input:

The user can specify a list of features as input by specifying the features as a list for the parameter *featureList*. In the following example, we aim to calculate the standard deviation and Stetson L of data:

```
a = FATS.FeatureSpace(featureList=['Std', 'StetsonL'])
a=a.calculateFeature(lc)

Table(a)
```

Feature	Value
Std	0.14157
StetsonL	0.58237

#### - Available data as an input:

In case the user does not have all the input vectors mentioned above, it is necessary to specify available data by specifying the list of vectors using the parameter *Data*. In the example below calculate all the features that can be computed with the magnitude and time as an input.

```
a = FATS.FeatureSpace(Data=['magnitude', 'time'])
a=a.calculateFeature(lc)

Table(a)
```

Warning: the feature Beyond1Std could not be calculated because ['magnitude', 'error'] are needed.  
Warning: the feature CAR\_mean could not be calculated because ['magnitude', 'time', 'error'] are needed.  
Warning: the feature CAR\_sigma could not be calculated because ['magnitude', 'time', 'error'] are needed.  
Warning: the feature CAR\_tau could not be calculated because ['magnitude', 'time', 'error'] are needed.  
Warning: the feature Color could not be calculated because ['magnitude', 'time', 'magnitude2'] are needed.

Warning: the feature Eta\_color could not be calculated because ['magnitude', 'time', 'magnitude2'] are needed.  
Warning: the feature Q31\_color could not be calculated because ['magnitude', 'time', 'magnitude2'] are needed.  
Warning: the feature StetsonJ could not be calculated because ['magnitude', 'time', 'error', 'magnitude2'] are needed.  
Warning: the feature StetsonK could not be calculated because ['magnitude', 'error'] are needed.  
Warning: the feature StetsonK\_AC could not be calculated because ['magnitude', 'time', 'error'] are needed.  
Warning: the feature StetsonL could not be calculated because ['magnitude', 'time', 'error', 'magnitude2'] are needed.

Feature	Value
Amplitude	0.26500
AndersonDarling	1.00000
Autocor_length	1.00000
Con	0.00000
Eta_e	905.63620
FluxPercentileRatioMid20	0.09131
FluxPercentileRatioMid35	0.17817
FluxPercentileRatioMid50	0.31626
FluxPercentileRatioMid65	0.52339
FluxPercentileRatioMid80	0.79955
Freq1_harmonics_amplitude_0	0.13438
Freq1_harmonics_amplitude_1	0.08178
Freq1_harmonics_amplitude_2	0.05087
Freq1_harmonics_amplitude_3	0.02620
Freq1_harmonics_rel_phase_0	0.00000
Freq1_harmonics_rel_phase_1	0.35899
Freq1_harmonics_rel_phase_2	0.78374
Freq1_harmonics_rel_phase_3	1.31484
Freq2_harmonics_amplitude_0	0.01835
Freq2_harmonics_amplitude_1	0.00342
Freq2_harmonics_amplitude_2	0.00664
Freq2_harmonics_amplitude_3	0.00467
Freq2_harmonics_rel_phase_0	0.00000
Freq2_harmonics_rel_phase_1	3.08479
Freq2_harmonics_rel_phase_2	0.46901
Freq2_harmonics_rel_phase_3	1.77596
Freq3_harmonics_amplitude_0	0.01655
Freq3_harmonics_amplitude_1	0.00188
Freq3_harmonics_amplitude_2	0.00661
Freq3_harmonics_amplitude_3	0.00375
Freq3_harmonics_rel_phase_0	0.00000
Freq3_harmonics_rel_phase_1	0.62032
Freq3_harmonics_rel_phase_2	-0.12834
Freq3_harmonics_rel_phase_3	-1.28916
LinearTrend	0.00001
MaxSlope	54.72526
Mean	-5.91799
Meanvariance	-0.02392
MedianAbsDev	0.05450



MedianBRP	0.74539
PairSlopeTrend	0.03333
PercentAmplitude	-0.11309
PercentDifferenceFluxPercentile	-0.07528
PeriodLS	0.93697
Period_fit	0.00000
Psi_CS	0.19179
Psi_eta	0.58910
Q31	0.14100
Rcs	0.03917
Skew	0.95647
SlottedA_length	0.10605
SmallKurtosis	1.37948
Std	0.14157

#### - List of features and available data as an input:

It is also possible to provide the available time series input vectors and calculate all possible features from a feature list using this data:

```
a = FATS.FeatureSpace(featureList=['Mean', 'Beyond1Std', 'CAR_sigma', 'Color', 'SlottedA_length'], Data=
'error'])
a=a.calculateFeature(lc)

Table(a)
```

Warning: the feature CAR\_sigma could not be calculated because ['magnitude', 'time', 'error'] are needed.  
Warning: the feature Color could not be calculated because ['magnitude', 'time', 'magnitude2'] are needed.  
Warning: the feature SlottedA\_length could not be calculated because ['magnitude', 'time'] are needed.

Feature	Value
Mean	-5.91799
Beyond1Std	0.22278

#### - Excluding list as an input

The user can also create a list of features to be excluded from the calculation. To do so, the list of features to be excluded can be passed as a list via the parameter *excludeList*. For example:

```
a = FATS.FeatureSpace(Data= ['magnitude', 'time'], excludeList = ['SlottedA_length', 'StetsonK_AC', 'P
a = a.calculateFeature(lc)

Table(a)
```

Warning: the feature Beyond1Std could not be calculated because ['magnitude', 'error'] are needed.  
Warning: the feature CAR\_mean could not be calculated because ['magnitude', 'time', 'error'] are needed.  
Warning: the feature CAR\_sigma could not be calculated because ['magnitude', 'time', 'error'] are needed.  
Warning: the feature CAR\_tau could not be calculated because ['magnitude', 'time', 'error'] are needed.  
Warning: the feature Color could not be calculated because ['magnitude', 'time', 'magnitude2'] are needed.  
Warning: the feature Eta\_color could not be calculated because ['magnitude', 'time', 'magnitude2'] are needed.  
Warning: the feature Q31\_color could not be calculated because ['magnitude', 'time', 'magnitude2'] are needed.  
Warning: the feature StetsonJ could not be calculated because ['magnitude', 'time', 'error', 'magnitude2'] are needed.  
Warning: the feature StetsonK could not be calculated because ['magnitude', 'error'] are needed.  
Warning: the feature StetsonL could not be calculated because ['magnitude', 'time', 'error', 'magnitude2'] are needed.

' ] are needed.

error: please run PeriodLS first to generate values for Period\_fit

error: please run PeriodLS first to generate values for Psi\_CS

error: please run PeriodLS first to generate values for Psi\_eta

Feature	Value
Amplitude	0.26500
AndersonDarling	1.00000
Autocor_length	1
Con	0.00000
Eta_e	905.63620
FluxPercentileRatioMid20	0.09131
FluxPercentileRatioMid35	0.17817
FluxPercentileRatioMid50	0.31626
FluxPercentileRatioMid65	0.52339
FluxPercentileRatioMid80	0.79955
Freq1_harmonics_amplitude_0	0.13438
Freq1_harmonics_amplitude_1	0.08178
Freq1_harmonics_amplitude_2	0.05087
Freq1_harmonics_amplitude_3	0.02620
Freq1_harmonics_rel_phase_0	0.00000
Freq1_harmonics_rel_phase_1	0.35899
Freq1_harmonics_rel_phase_2	0.78374
Freq1_harmonics_rel_phase_3	1.31484
Freq2_harmonics_amplitude_0	0.01835
Freq2_harmonics_amplitude_1	0.00342
Freq2_harmonics_amplitude_2	0.00664
Freq2_harmonics_amplitude_3	0.00467
Freq2_harmonics_rel_phase_0	0.00000
Freq2_harmonics_rel_phase_1	3.08479
Freq2_harmonics_rel_phase_2	0.46901
Freq2_harmonics_rel_phase_3	1.77596
Freq3_harmonics_amplitude_0	0.01655
Freq3_harmonics_amplitude_1	0.00188
Freq3_harmonics_amplitude_2	0.00661
Freq3_harmonics_amplitude_3	0.00375
Freq3_harmonics_rel_phase_0	0.00000
Freq3_harmonics_rel_phase_1	0.62032
Freq3_harmonics_rel_phase_2	-0.12834
Freq3_harmonics_rel_phase_3	-1.28916
LinearTrend	0.00001
MaxSlope	54.72526
Mean	-5.91799
Meanvariance	-0.02392
MedianAbsDev	0.05450
MedianBRP	0.74539
PairSlopeTrend	0.03333

PercentAmplitude	-0.11309
PercentDifferenceFluxPercentile	-0.07528
Period_fit	None
Psi_CS	None
Psi_eta	None
Q31	0.14100
Rcs	0.03917
Skew	0.95647
SmallKurtosis	1.37948
Std	0.14157

### - All the features:

To calculate all the existing features in the library, the user can set the *Data* parameter to the string and set the *featureList* parameter to be *None*. Obviously, in order to do this, the user must have provided all the necessary time series input vectors.

```
a = FATS.FeatureSpace(Data='all', featureList=None)
a=a.calculateFeature(lc)
```

Table(a)

Feature	Value
Amplitude	0.26500
AndersonDarling	1.00000
Autocor_length	1.00000
Beyond1Std	0.22278
CAR_mean	-9.23070
CAR_sigma	-0.21928
CAR_tau	0.64112
Color	-0.33326
Con	0.00000
Eta_color	12930.68526
Eta_e	905.63620
FluxPercentileRatioMid20	0.09131
FluxPercentileRatioMid35	0.17817
FluxPercentileRatioMid50	0.31626
FluxPercentileRatioMid65	0.52339
FluxPercentileRatioMid80	0.79955
Freq1_harmonics_amplitude_0	0.13438
Freq1_harmonics_amplitude_1	0.08178
Freq1_harmonics_amplitude_2	0.05087
Freq1_harmonics_amplitude_3	0.02620
Freq1_harmonics_rel_phase_0	0.00000
Freq1_harmonics_rel_phase_1	0.35899
Freq1_harmonics_rel_phase_2	0.78374

Freq1_harmonics_rel_phase_3	1.31484
Freq2_harmonics_amplitude_0	0.01835
Freq2_harmonics_amplitude_1	0.00342
Freq2_harmonics_amplitude_2	0.00664
Freq2_harmonics_amplitude_3	0.00467
Freq2_harmonics_rel_phase_0	0.00000
Freq2_harmonics_rel_phase_1	3.08479
Freq2_harmonics_rel_phase_2	0.46901
Freq2_harmonics_rel_phase_3	1.77596
Freq3_harmonics_amplitude_0	0.01655
Freq3_harmonics_amplitude_1	0.00188
Freq3_harmonics_amplitude_2	0.00661
Freq3_harmonics_amplitude_3	0.00375
Freq3_harmonics_rel_phase_0	0.00000
Freq3_harmonics_rel_phase_1	0.62032
Freq3_harmonics_rel_phase_2	-0.12834
Freq3_harmonics_rel_phase_3	-1.28916
LinearTrend	0.00001
MaxSlope	54.72526
Mean	-5.91799
Meanvariance	-0.02392
MedianAbsDev	0.05450
MedianBRP	0.74539
PairSlopeTrend	0.03333
PercentAmplitude	-0.11309
PercentDifferenceFluxPercentile	-0.07528
PeriodLS	0.93697
Period_fit	0.00000
Psi_CS	0.19179
Psi_eta	0.58910
Q31	0.14100
Q31_color	0.10600
Rcs	0.03917
Skew	0.95647
SlottedA_length	0.10605
SmallKurtosis	1.37948
Std	0.14157
StetsonJ	1.39841
StetsonK	0.69063
StetsonK_AC	0.78703
StetsonL	0.58237

## Library output

When calculating the features of a light-curve, the output can be returned in three different formats:

- "dict": returns a vector with the name of each feature followed by its value.
- "array": returns a vector with only the features values.
- "features" : returns a vector with the list of the calculated features.

The output format can be specified via a string for the parameter *method*, as shown in the example below:

```

a.result(method='dict')

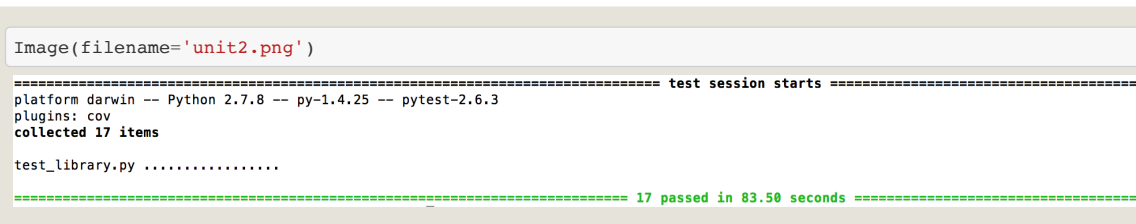
{'Amplitude': 0.265000000000000057,
 'AndersonDarling': 1.0,
 'Autocor_length': 1.0,
 'Beyond1Std': 0.22278056951423786,
 'CAR_mean': -9.230698873903961,
 'CAR_sigma': -0.21928049298842511,
 'CAR_tau': 0.64112037377348619,
 'Color': -0.33325502453332145,
 'Con': 0.0,
 'Eta_color': 12930.685257570141,
 'Eta_e': 905.63620081228805,
 'FluxPercentileRatioMid20': 0.091314031180401739,
 'FluxPercentileRatioMid35': 0.1781737193763922,
 'FluxPercentileRatioMid50': 0.3162583518930947,
 'FluxPercentileRatioMid65': 0.52338530066815037,
 'FluxPercentileRatioMid80': 0.79955456570155925,
 'Freq1_harmonics_amplitude_0': 0.13437970279331715,
 'Freq1_harmonics_amplitude_1': 0.081782103003790088,
 'Freq1_harmonics_amplitude_2': 0.05087059528477509,
 'Freq1_harmonics_amplitude_3': 0.026198886048114284,
 'Freq1_harmonics_rel_phase_0': 0.0,
 'Freq1_harmonics_rel_phase_1': 0.35898717688700943,
 'Freq1_harmonics_rel_phase_2': 0.7837375001939666,
 'Freq1_harmonics_rel_phase_3': 1.3148410684236285,
 'Freq2_harmonics_amplitude_0': 0.018352700251784763,
 'Freq2_harmonics_amplitude_1': 0.0034160049640413102,
 'Freq2_harmonics_amplitude_2': 0.0066381164603388594,
 'Freq2_harmonics_amplitude_3': 0.0046673628756084525,
 'Freq2_harmonics_rel_phase_0': 0.0,
 'Freq2_harmonics_rel_phase_1': 3.084788968440014,
 'Freq2_harmonics_rel_phase_2': 0.4690091609786593,
 'Freq2_harmonics_rel_phase_3': 1.7759633632591398,
 'Freq3_harmonics_amplitude_0': 0.016548091278371951,
 'Freq3_harmonics_amplitude_1': 0.0018836301087421642,
 'Freq3_harmonics_amplitude_2': 0.0066137938752542803,
 'Freq3_harmonics_amplitude_3': 0.0037519587436371116,
 'Freq3_harmonics_rel_phase_0': 0.0,
 'Freq3_harmonics_rel_phase_1': 0.62032188924875542,
 'Freq3_harmonics_rel_phase_2': -0.12834003984379072,
 'Freq3_harmonics_rel_phase_3': -1.289156932085701,
 'LinearTrend': 6.1736585768121618e-06,
 'MaxSlope': 54.725258361167832,
 'Mean': -5.9179891122278052,
 'Meanvariance': -0.023922513589418142,
 'MedianAbsDev': 0.054499999999999993,
 'MedianBRP': 0.74539363484087107,
 'PairSlopeTrend': 0.033333333333333333,
 'PercentAmplitude': -0.11308575739793782,
 'PercentDifferenceFluxPercentile': -0.075278732500628692,
 'PeriodLS': 0.93697445905023935,
 'Period_fit': 3.1076457440339972e-113,
 'Psi_CS': 0.19179114069585729,
 'Psi_eta': 0.58910268294132195,
 'Q31': 0.14100000000000001,
 'Q31_color': 0.106000000000000076,
 'Rcs': 0.039171450772665782,
 'Skew': 0.95646986755937902,
 'SlottedA_length': 0.10605400000349618,
 'SmallKurtosis': 1.3794786801255068,
 'Std': 0.14157317495929828,
 'StetsonJ': 1.3984111401436403,
 'StetsonK': 0.69062662628891813,
 'StetsonK_AC': 0.78703471849632078,
 'StetsonL': 0.5823703637198997}

```

## Unit test

In order to systematically check the correctness of the feature generation implementation in our library, a unit test is created for each feature in a unit test file named "test\_library.py". This script should always run before using the library by executing `$ py.test` at the command line. In addition, if a contributor adds a new feature for the library, a pertinent test should be added to the unit test file. The idea is to guarantee, as far as possible, that every feature calculated is correct. In most cases, this can be reached by calculating the expected feature value for a known distribution (normal, uniform, etc.), and then checking it with the value obtained from the library.

The following image shows how `py.test` output should look if all the tests are passed:



```
Image(filename='unit2.png')
===== test session starts =====
platform darwin -- Python 2.7.8 -- py-1.4.25 -- pytest-2.6.3
plugins: cov
collected 17 items
test_library.py .....
===== 17 passed in 83.50 seconds =====
```

## Importing light-curves Toolbox

In addition to the features library, we provide a basic toolbox for importing and preprocessing the data. For every feature in the library to be calculated, six vectors should be available per light-curve: magnitude, time of measurement and the associated observational errors, each one of them in different bands.

For example, the function `ReadLC_MACHO()` receives a MACHO id (object id assigned in the MACHO survey) as an input and returns the following output:

- **mag**: magnitude measurement
- **time**: time of measurement
- **error**: associated observational error

A demonstration of how to import a MACHO light-curve is presented below. Besides opening the data file, the data is:

- **preprocessed**: points within 5 standard deviations from the mean are considered as noise and thus are eliminated.
- **synchronized**: it synchronizes the light-curves in the two different bands and returns **aligned\_mag**, **aligned\_mag2**, **aligned\_time**, **aligned\_error** and **aligned\_error2**.

Note: `mag`, `time` and `error` must have the same length as well as `aligned_mag`, `aligned_mag2`, `aligned_time`, `aligned_error` and `aligned_error2`.

```

#We open the ligh curve in two different bands
lc_B = FATS.ReadLC_MACHO('lc_1.3444.614.B.mjd')
lc_R = FATS.ReadLC_MACHO('lc_1.3444.614.R.mjd')

#We import the data
[mag, time, error] = lc_B.ReadLC()
[mag2, time2, error2] = lc_R.ReadLC()

#We preprocess the data
preprocessed_data = FATS.Preprocess_LC(mag, time, error)
[mag, time, error] = preprocessed_data.Preprocess()

preprocessed_data = FATS.Preprocess_LC(mag2, time2, error2)
[mag2, time2, error2] = preprocessed_data.Preprocess()

#We synchronize the data
if len(mag) != len(mag2):
    [aligned_mag, aligned_mag2, aligned_time, aligned_error, aligned_error2] = \
        FATS.Align_LC(time, time2, mag, mag2, error, error2)

lc = np.array([mag, time, error, mag2, aligned_mag, aligned_mag2, aligned_time, aligned_error, aligne

```

It is sometimes helpful to visualize the data before processing it. For a representation of the light curve we can plot it as follows:



Note: for periodic light-curves we are able to transform the photometric time series into a single light curve in which each period is mapped onto the same time axis as follows:

$$t' = \left\{ \frac{t - t_0}{T} \right\}$$

where

$T$

is the period,

$t_0$

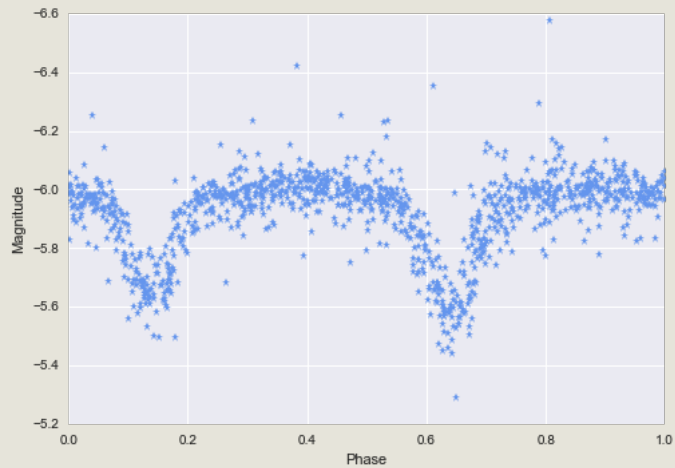
is an arbitrary starting point and the symbol  $\{ \}$  represents the non-integer part of the fraction. The process produces a folded light-curve on an x-axis of folded time that ranges from 0 to 1. The corresponding folded light-curve of the previous example is shown next:



```

Color = [ 0.392157, 0.584314 ,0.929412];
T = 2 * 0.93697
new_b=np.mod(time, T) / T;
idx=np.argsort(2*new_b)
plt.plot( new_b, mag, '*', color = Color)
plt.xlabel("Phase")
plt.ylabel("Magnitude")
plt.gca().invert_yaxis()

```



## The features

The next section details the features that we have developed in order to represent light curves. For each feature, we also describe a benchmark test performed in order to test the feature's correctness.

Note: Each feature was also tested to ensure invariance to unequal sampling. Because telescope observations are not always taken at uniformly spaced intervals, the light curve features should be invariant to this nonuniformity. These tests are explained in the [appendix](#) of this document.

Let's first create some characteristic synthetic light-curves in order to test each one of the features along the library:

- *lc\_normal* : its magnitude follows a Gaussian distribution
- *lc\_periodic* : its magnitude has a periodic variability
- *lc\_uniform* : its magnitude follows a uniform distribution

```

mag_normal = np.random.normal(size=10000)
time_normal = np.arange(10000)
error_normal = np.random.normal(loc=1, scale =0.008, size=10000)
mag_normal2 = np.random.normal(size=10000)
error_normal2 = np.random.normal(loc=1, scale =0.008, size=10000)
aligned_mag_normal = mag_normal
aligned_mag_normal2 = mag_normal2
aligned_time_normal = time_normal
aligned_error_normal = error_normal
aligned_error_normal2 = error_normal2

lc_normal = np.array([mag_normal, time_normal, error_normal, mag_normal2, aligned_mag_normal, aligned_time_normal, aligned_error_normal, aligned_error_normal2])

```

```

N=100
time_periodic = np.arange(N)
Period = 10
cov = np.zeros([N,N])
mean = np.zeros(N)
for i in np.arange(N):
    for j in np.arange(N):
        cov[i,j] = np.exp( -(np.sin( (np.pi/Period) *(i-j))**2))
mag_periodic = np.random.multivariate_normal(mean, cov)
lc_periodic = np.array([mag_periodic, time_periodic])

```

-c:9: RuntimeWarning: covariance is not positive-semidefinite.

```

mag_uniform=np.random.uniform(size=10000)
time_uniform=np.arange(10000)
lc_uniform = np.array([mag_uniform, time_uniform])

```

### Mean

Mean magnitude. For a normal distribution it should take a value close to zero:

```

a = FATS.FeatureSpace(featureList=[ 'Mean' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')
{'Mean': 0.0082611563419413246}

```

### Standard deviation

The standard deviation

$\sigma$

of the sample is defined as:

$$\sigma = \frac{1}{N-1} \sum_i (y_i - \bar{y})^2$$

For example, a white noise time serie should have

$\sigma = 1$

```

a = FATS.FeatureSpace(featureList=[ 'Std' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')
{'Std': 0.99320419310116881}

```

### Range of a cumulative sum

$R_{CS}$

$R_{cs}$

is the range of a cumulative sum (Ellaway 1978) of each light-curve and is defined as:

$$R_{cs} = \max(S) - \min(S)$$
$$S = \frac{1}{N\sigma} \sum_{i=1}^l (m_i - \bar{m})$$

where  $\max(\min)$  is the maximum (minimum) value of  $S$  and  
 $l = 1, 2, \dots, N$

.

$R_{cs}$

should take a value close to zero for any symmetric distribution:

```
a = FATS.FeatureSpace(featureList=[ 'Rcs' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'Rcs': 0.0094459606901065168}
```

### Period Lomb-Scargle

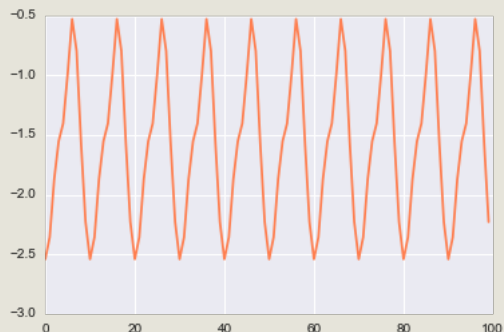
The Lomb-Scargle (L-S) algorithm (Scargle, 1982) is a variation of the Discrete Fourier Transform (DFT) in which a time series is decomposed into a linear combination of sinusoidal functions. The basic sinusoidal functions transform the data from the time domain to the frequency domain. DFT techniques often assume evenly spaced data points in the time series, but this is rarely the case with astrophysical time-series data. Scargle has derived a formula for transform coefficients that is similar to the DFT in the limit of evenly spaced observations. In addition, an adjustment of the values used to calculate the transform coefficients makes the transform invariant to time shifts.

The Lomb-Scargle periodogram is optimized to identify sinusoidal-shaped periodic signals in time series data. Particular applications include radial velocity data and searches for pulsating variable stars. L-S is not optimal for detecting signals from transiting exoplanets, where the shape of the periodic light curve is not sinusoidal.

Next, we perform a test on the synthetic periodic light-curve we created (which period is 20) to compare the accuracy of the period found by the L-S method:

```
Color = [ 1 ,0.498039, 0.313725];
plt.plot(time_periodic,mag_periodic, color=Color)
```

```
[<matplotlib.lines.Line2D at 0x10a0be690>]
```



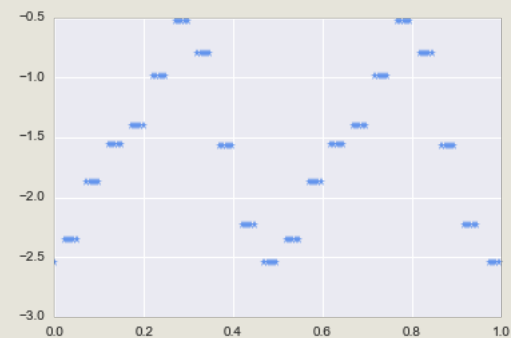
```
a = FATS.FeatureSpace(featureList=['PeriodLS'])
a=a.calculateFeature(lc_periodic)
print "Period calculated:", a.result(method='array'), "True Period:", Period
```

```
Period calculated: [ 10.06779661] True Period: 10
```

We can also confirm the validity of this result by folding the light-curve as explained in [the introductory](#)

```
Color = [ 0.392157, 0.584314 ,0.929412];
T = 2 * a.result(method='array')
new_b = np.mod(time_periodic, T) / T;
idx = np.argsort(2 * new_b)
plt.plot( new_b, mag_periodic, '*', color=Color)
#plt.plot(np.sort(new_b), data3[np.argsort(new_b)], '*', color='red')
```

```
[<matplotlib.lines.Line2D at 0x106073690>]
```



## Period fit

Returns the false alarm probability of the largest periodogram value. Let's test it for a normal distrib data and for a periodic one:

```
a = FATS.FeatureSpace(featureList=['PeriodLS', 'Period_fit'])
a=a.calculateFeature(lc_normal)
print "White noise data:", a.result(method='dict')

a = FATS.FeatureSpace(featureList=['PeriodLS', 'Period_fit'])
a=a.calculateFeature(lc_periodic)
print "Periodic data:", a.result(method='dict')

White noise data: {'Period_fit': 1.0, 'PeriodLS': 22.545659526493797}
Periodic data: {'Period_fit': 3.1982987167114662e-13, 'PeriodLS': 19.800000000000001}
```

$\Psi_{CS}$

$R_{CS}$

applied to the phase-folded light curve (generated using the period estimated from the Lomb-Sc. method).

```
a = FATS.FeatureSpace(featureList=['PeriodLS', 'Psi_CS' ])
a=a.calculateFeature(lc_periodic)
print a.result(method='dict')

{'PeriodLS': 19.800000000000001, 'Psi_CS': 0.26373298715550053}
```

Color

The color is defined as the difference between the average magnitude of two different b. observations.

```
a = FATS.FeatureSpace(featureList=[ 'Color' ])
a=a.calculateFeature(lc)
print a.result(method='dict')

{'Color': -0.33325502453332145}
```

### Autocorrelation\_length

The autocorrelation, also known as serial correlation, is the cross-correlation of a signal with i. Informally, it is the similarity between observations as a function of the time lag between them. It is a mathematical tool for finding repeating patterns, such as the presence of a periodic signal obscured by noise, or identifying the missing fundamental frequency in a signal implied by its harmonic frequency.

For an observed series

$y_1, y_2, \dots, y_T$

with sample mean

$\bar{y}$

, the sample lag

$-h$

autocorrelation is given by:

$$\hat{\rho}_h = \frac{\sum_{t=h+1}^T (y_t - \bar{y})(y_{t-h} - \bar{y})}{\sum_{t=1}^T (y_t - \bar{y})^2}$$

Since the autocorrelation function of a light curve is given by a vector and we can only return one value as a feature, we define the length of the autocorrelation function where its value is smaller than  $e^{-1}$ .

```
a = FATS.FeatureSpace(featureList=[ 'Autocor_length' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'Autocor_length': 1}
```

### Slotted Autocorrelation

In slotted autocorrelation, time lags are defined as intervals or slots instead of single values. The slotted autocorrelation function at a certain time lag slot is computed by averaging the cross product between samples whose time differences fall in the given slot.

$$\hat{\rho}(\tau = kh) = \frac{1}{\hat{\rho}(0) N_\tau} \sum_{t_i} \sum_{t_j=t_i+(k-1/2)h}^{t_i+(k+1/2)h} \bar{y}_i(t_i) \bar{y}_j(t_j)$$

Where

$h$

is the slot size,

$\bar{y}$

is the normalized magnitude,

$\hat{\rho}(0)$

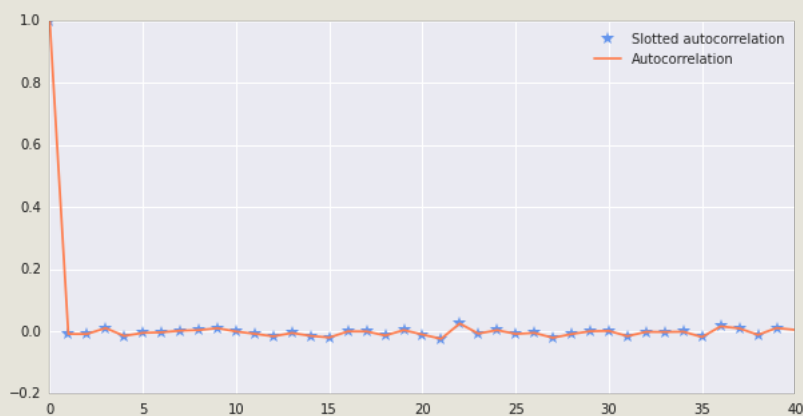
is the slotted autocorrelation for the first lag, and

$N_\tau$

is the number of pairs that fall in the given slot. In order to check the validity of this feature let's calculate the slotted autocorrelation for a normal distribution with  $T=1$  and compare it with the autocorrelation function. These two should be equivalent in this case where the time intervals are constant.

```
Color = [ 0.392157, 0.584314, 0.929412];
Color2 = [ 1, 0.498039, 0.313725];
plt.figure(figsize=(10,5))
SAC = slotted_autocorrelation(mag_normal, time_normal, 1, 100)
a, = plt.plot(SAC[0:40], '*', color=Color, markersize=10)
AC = stattools.acf(mag_normal)
b, = plt.plot(AC, color=Color2)
plt.legend([a, b], ['Slotted autocorrelation', 'Autocorrelation'])
```

<matplotlib.legend.Legend at 0x101094750>



```
AC = stattools.acf(mag_normal)
k = next((index for index,value in enumerate(AC) if value < np.exp(-1)), None)
print "From the autocorrelation function:", k

a = FATS.FeatureSpace(featureList=['SlottedA_length'], SlottedA_length = 1)
a=a.calculateFeature(lc_normal)
print a.result(method='dict')
```

```
From the autocorrelation function: 1
{'SlottedA_length': 1}
```

Stetson K, Stetson K\_AC, Stetson J and Stetson L

These three features are based on the Welch/Stetson variability index

$I$   
(Stetson, 1996) defined by the equation:

$$I = \sqrt{\frac{1}{n(n-1)} \sum_{i=1}^n \left( \frac{b_i - \hat{b}}{\sigma_{b,i}} \right) \left( \frac{v_i - \hat{v}}{\sigma_{v,i}} \right)}$$

where

$b_i$

and

$v_i$

are the apparent magnitudes obtained for the candidate star in two observations closely spaced in time on some occasion

$i$

,

$\sigma_{b,i}$

and

$\sigma_{v,i}$

are the standard errors of those magnitudes,

$\hat{b}$

and

$\hat{v}$

are the weighted mean magnitudes in the two filters, and

$n$

is the number of observation pairs.

Since a given frame pair may include data from two filters which did not have equal number of observations overall, the "relative error" is calculated as follows:

$$\delta = \sqrt{\frac{n}{n-1} \frac{v - \hat{v}}{\sigma_v}}$$

allowing all residuals to be compared on an equal basis.

#### - Stetson K

Stetson K is a robust kurtosis measure:

$$\frac{1/N \sum_{i=1}^N |\delta_i|}{\sqrt{1/N \sum_{i=1}^N \delta_i^2}}$$

where the index

$i$

runs over all

$N$

observations available for the star without regard to pairing. For a Gaussian magnitude distribution, the kurtosis should take a value close to

$\sqrt{2/\pi} = 0.798$

, let's test it:

```
a = FATS.FeatureSpace(featureList=[ 'StetsonK' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'StetsonK': 0.79914938521401002}
```

#### - Stetson K\_AC

Stetson K applied to the slotted autocorrelation function of the light-curve.

```
a = FATS.FeatureSpace(featureList=[ 'SlottedA_length', 'StetsonK_AC' ])
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'SlottedA_length': 1.0, 'StetsonK_AC': 0.20917402545294403}
```

#### - Stetson J

Stetson J is a robust version of the variability index. It is calculated based on two simultaneous curves of a same star and is defined as:

$$J = \sum_{k=1}^n \text{sgn}(P_k) \sqrt{|P_k|}$$

with  
 $P_k = \delta_{i_k} \delta_{j_k}$

For a Gaussian magnitude distribution, J should take a value close to zero:

```
a = FATS.FeatureSpace(featureList=[ 'StetsonJ' ])
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'StetsonJ': 0.010765631555204736}
```

#### - Stetson L

Stetson L variability index describes the synchronous variability of different bands and is defined as:

$$L = \frac{JK}{0.798}$$

Again, for a Gaussian magnitude distribution, L should take a value close to zero:

```
a = FATS.FeatureSpace(featureList=[ 'StetsonL' ])
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'StetsonL': 0.0085957106316273714}
```

#### Variability index

$\eta$

REMOVED FROM THE LIBRARY

Variability index

$\eta$

is the ratio of the mean of the square of successive differences to the variance of data points. The index was originally proposed to check whether the successive data points are independent or not. In other words, the index was developed to check if any trends exist in the data (von Neumann 1941). It is defined as:

$$\eta = \frac{1}{(N-1)\sigma^2} \sum_{i=1}^{N-1} (m_{i+1} - m_i)^2$$

The variability index should take a value close to 2 for a normal distribution.

This feature was removed from the library because it does not take into account unequal sampling.



```
a = FATS.FeatureSpace(featureList=[ 'VariabilityIndex' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')
```

could not find feature VariabilityIndex

An exception has occurred, use %tb to see the full traceback.

SystemExit: 1

To exit: use 'exit', 'quit', or Ctrl-D.

## Variability index

$\eta$

Although

$\eta$

is a powerful index for quantifying variability characteristics of a time series, it does not take account unequal sampling. Thus

$\eta$

is defined as:

$$\eta = \bar{w} (t_{N-1} - t_1) \frac{\sum_{i=1}^{N-1} w_i (m_{i+1} - m_i)^2}{\sigma^2 \sum_{i=1}^{N-1} w_i}$$

$$w_i = \frac{1}{(t_{i+1} - t_i)^3}$$

```
a = FATS.FeatureSpace(featureList=[ 'Eta_e' ])
a=a.calculateFeature(lc_normal)
print a.result(method='dict')
```

```
{'Eta_e': 2.0028592616231866}
```

```
mag_normal2 = 1000 * np.random.normal(size=1000, scale = 25)
time_normal2 = np.arange(1000)
```

```
lc_normal2 = np.array([mag_normal2, time_normal2])
```

```
a = FATS.FeatureSpace(featureList=[ 'Eta_e' ])
a=a.calculateFeature(lc_normal2)
print a.result(method='dict')
```

```
{'Eta_e': 2.0151803402689414}
```

## Variability index

$\eta_{color}$

$\eta$

index calculated from the color light-curve.

```
a = FATS.FeatureSpace(featureList=[ 'Eta_color' ])
a=a.calculateFeature(lc_normal)
print a.result(method='dict')
```

```
{'Eta_color': 1.991749074648397}
```

$\Psi_\eta$

$\eta^f$

index calculated from the folded light curve.

```
a = FATS.FeatureSpace(featureList=[ 'PeriodLS', 'Psi_eta' ])
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'Psi_eta': 2.0004188881194875, 'PeriodLS': 7.3225924569754675}
```

### Small Kurtosis

Small sample kurtosis of the magnitudes:

$$Kurtosis = \frac{N(N+1)}{(N-1)(N-2)(N-3)} \sum_{i=1}^N \left( \frac{m_i - \hat{m}}{\sigma} \right)^4 - \frac{3(N-1)}{(N-2)(N-3)}$$

For a normal distribution, the small kurtosis should be zero:

```
a = FATS.FeatureSpace(featureList=[ 'SmallKurtosis' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'SmallKurtosis': 0.044451779515607193}
```

### Skewness

The skewness of a sample is defined as follow:

$$Skewness = \frac{N}{(N-1)(N-2)} \sum_{i=1}^N \left( \frac{m_i - \hat{m}}{\sigma} \right)^3$$

For a normal distribution it should be equal to zero:

```
a = FATS.FeatureSpace(featureList=[ 'Skew' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'Skew': -0.00023325826785278685}
```

### Median absolute deviation

The median absolute deviation is defined as the median discrepancy of the data from the median data:

$$MedianAbsoluteDeviation = median(|mag - median(mag)|)$$

It should take a value close to 0.675 for a normal distribution:

```
a = FATS.FeatureSpace(featureList=[ 'MedianAbsDev' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'MedianAbsDev': 0.66332131466690614}
```

### Amplitude

The amplitude is defined as the half of the difference between the median of the maximum 5% and median of the minimum 5% magnitudes. For a sequence of numbers from 0 to 1000 the amplitude should be equal to 475.5:

```
a = FATS.FeatureSpace(featureList=[ 'Amplitude' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'Amplitude': 1.9547603768486537}
```

## Con

Index introduced for the selection of variable stars from the OGLE database (Wozniak 2000) calculate Con, we count the number of three consecutive data points that are brighter or fainter than  $2\sigma$  and normalize the number by  $N - 2$ .

For a normal distribution and by considering just one star, Con should take values close to 0.045:

```
a = FATS.FeatureSpace(featureList=[ 'Con' ] , Con=1)
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'Con': 0.0476}
```

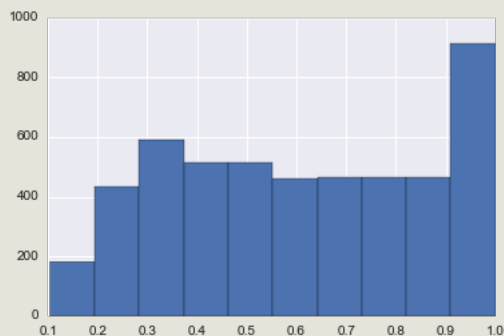
## Anderson-Darling test

The Anderson-Darling test is a statistical test of whether a given sample of data is drawn from a  $\chi^2$  probability distribution. When applied to testing if a normal distribution adequately describes a set of data, it is one of the most powerful statistical tools for detecting most departures from normality.

For a normal distribution the Anderson-Darling statistic should take values close to 0.25:

```
b=[]
for i in xrange(5000):
    data2 = np.random.normal(size=10000)
    a = FATS.FeatureSpace(featureList=[ 'AndersonDarling' ] )
    a=a.calculateFeature(data2)
    b.extend(a.result())

fig = plt.hist(b)
```



## Linear trend

Slope of a linear fit to the light-curve.

```
a = FATS.FeatureSpace(featureList=[ 'LinearTrend' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'LinearTrend': -3.2084065290292509e-06}
```

### Max slope

Maximum absolute magnitude slope between two consecutive observations.

```
a = FATS.FeatureSpace(featureList=[ 'MaxSlope' ])
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'MaxSlope': 5.4943105823904741}
```

### Beyond 1 std

Percentage of points beyond one standard deviation from the weighted mean.

For a normal distribution, it should take a value close to 0.32:

```
a = FATS.FeatureSpace(featureList=[ 'Beyond1Std' ])
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'Beyond1Std': 0.317}
```

### Pair slope trend

Considering the last 30 (time-sorted) measurements of source magnitude, the fraction of increasing differences minus the fraction of decreasing first differences.

```
a = FATS.FeatureSpace(featureList=[ 'PairSlopeTrend' ])
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'PairSlopeTrend': -0.16666666666666666}
```

### Flux percentile ratio mid20, mid 35, mid 50, mid 65 and mid 80

In order to characterize the sorted magnitudes distribution we use percentiles. If

$F_{5,95}$

is the difference between

95%

and

5%

magnitude values, we calculate the following:

- flux\_percentile\_ratio\_mid20: ratio  $F_{40,60}/F_{5,95}$
- flux\_percentile\_ratio\_mid35: ratio  $F_{32.5,67.5}/F_{5,95}$
- flux\_percentile\_ratio\_mid50: ratio  $F_{25,75}/F_{5,95}$
- flux\_percentile\_ratio\_mid65: ratio  $F_{17.5,82.5}/F_{5,95}$
- flux\_percentile\_ratio\_mid80: ratio  $F_{10,90}/F_{5,95}$

For the first feature for example, in the case of a normal distribution, this is equivalent to calculate

$$\frac{\text{erf}^{-1}(2 \cdot 0.6 - 1) - \text{erf}^{-1}(2 \cdot 0.4 - 1)}{\text{erf}^{-1}(2 \cdot 0.95 - 1) - \text{erf}^{-1}(2 \cdot 0.05 - 1)}$$

. So, the expected values for each of the flux percentile features are:

- flux\_percentile\_ratio\_mid20 = 0.154
- flux\_percentile\_ratio\_mid35 = 0.275
- flux\_percentile\_ratio\_mid50 = 0.410
- flux\_percentile\_ratio\_mid65 = 0.568
- flux\_percentile\_ratio\_mid80 = 0.779

```
a = FATS.FeatureSpace(featureList=[ 'FluxPercentileRatioMid20', 'FluxPercentileRatioMid35', 'FluxPercentileRatioMid50', 'FluxPercentileRatioMid65', 'FluxPercentileRatioMid80' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')
```

```
{'FluxPercentileRatioMid80': 0.77279316966429079, 'FluxPercentileRatioMid50': 0.41288273174285733, 'FluxPercentileRatioMid65': 0.568252869198, 'FluxPercentileRatioMid35': 0.27711282267863224, 'FluxPercentileRatioMid20': 0.15413440040940599}
```

### Percent difference flux percentile

Ratio of

$F_{5,95}$

over the median magnitude.

```
a = FATS.FeatureSpace(featureList=[ 'PercentDifferenceFluxPercentile' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')
```

```
{'PercentDifferenceFluxPercentile': -134.93590403825007}
```

$Q_{3-1}$   
 is the difference between the third quartile,  
 $Q_3$   
 , and the first quartile,  
 $Q_1$   
 , of a raw light curve.  
 $Q_1$   
 is a split between the lowest  
 25%  
 and the highest  
 75%  
 of data.  
 $Q_3$   
 is a split between the lowest  
 75%  
 and the highest  
 25%  
 of data.

```

a = FATS.FeatureSpace(featureList=[ 'Q31' ])
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'Q31': 1.3320376563134508}

```

$Q_{3-1}|B-R$

$Q_{3-1}$   
 applied to the difference between both bands of a light curve (B-R).

```

a = FATS.FeatureSpace(featureList=[ 'Q31_color' ])
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'Q31_color': 1.8840489594535512}

```

### Percent amplitude

Largest percentage difference between either the max or min magnitude and the median.

```

a = FATS.FeatureSpace(featureList=[ 'PercentAmplitude' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'PercentAmplitude': -168.991253993057}

```

### Mean variance

$\frac{\sigma}{m}$

This is a simple variability index and is defined as the ratio of the standard deviation,

$\sigma$

, to the mean magnitude,

$\bar{m}$

. If a light curve has strong variability,

$\frac{\sigma}{\bar{m}}$

of the light curve is generally large.

For a uniform distribution from 0 to 1, the mean is equal to 0.5 and the variance is equal to 1/12, the mean-variance should take a value close to 0.577:

```
a = FATS.FeatureSpace(featureList=[ 'Meanvariance' ] )
a=a.calculateFeature(lc_uniform)
print a.result(method='dict')

{'Meanvariance': 0.5816791217381897}
```

### Median buffer range percentage (MedianBRP)

Fraction of photometric points within amplitude/10 of the median magnitude.

```
a = FATS.FeatureSpace(featureList=[ 'MedianBRP' ] )
a=a.calculateFeature(lc_normal)
print a.result(method='dict')

{'MedianBRP': 0.559}
```

### CAR features

In order to model the irregular sampled times series we use CAR(1) (Brockwell and Davis, 2000) continuous time auto regressive model. CAR(1) process has three parameters, it provides a natural consistent way of estimating a characteristic time scale and variance of light-curves. CAR(1) process described by the following stochastic differential equation:

$$dX(t) = -\frac{1}{\tau}X(t)dt + \sigma_C\sqrt{d\tau}\epsilon(t) + bdt,$$

for  $\tau, \sigma_C, t \geq 0$

where the mean value of the lightcurve

$X(t)$

is

$b\tau$

and the variance is

$$\frac{\tau\sigma_C^2}{2}$$

.

$\tau$

is the relaxation time of the process

$X(T)$

, it can be interpreted as describing the variability amplitude of the time series.

$\sigma_C$

can be interpreted as describing the variability of the time series on time scales shorter than

$\tau$

.

$\epsilon(t)$

is a white noise process with zero mean and variance equal to one. The likelihood function of a CAR(1) model for a light-curve with observations

$x = \{x_1, \dots, x_n\}$

observed at times

$\{t_1, \dots, t_n\}$

with measurements error variances

$\{\sigma_1^2, \dots, \sigma_n^2\}$

is:

$$p(x|b, \sigma_C, \tau) = \prod_{i=1}^n \frac{1}{[2\pi(\Omega_i + \sigma_i^2)]^{1/2}} \exp\left\{-\frac{1}{2} \frac{(x_i^* - x_i^*)^2}{\Omega_i + \sigma_i^2}\right\}$$

$$x_i^* = x_i - b\tau$$

$$x_0^* = 0$$

$$\Omega_0 = \frac{\tau\sigma_C^2}{2}$$

$$x_i^* = a_i x_{i-1}^* + \frac{a_i \Omega_{i-1}}{\Omega_{i-1} + \sigma_{i-1}^2} (x_{i-1}^* + x_{i-1}^*)$$

$$\Omega_i = \Omega_0 (1 - a_i^2) + a_i^2 \Omega_{i-1} \left(1 - \frac{\Omega_{i-1}}{\Omega_{i-1} + \sigma_{i-1}^2}\right)$$

$$a_i = e^{-(t_i - t_{i-1})/\tau}$$

To find the optimal parameters we maximize the likelihood with respect to

$\sigma_C$

and

$\tau$

and calculate

$b$

as the mean magnitude of the light-curve divided by

$\tau$

.



```

a = FATS.FeatureSpace(featureList=[ 'CAR_sigma', 'CAR_tau', 'CAR_mean' ])
a=a.calculateFeature(lc)
print a.result(method='dict')

{'CAR_mean': -9.230698873903961, 'CAR_sigma': -0.21928049298842511, 'CAR_tau': 0.64112037377348619}

```

### Periodic features extracted from light-curves using Lomb–Scargle (Richards et al., 2011)

Here, we adopt a model where the time series of the photometric magnitudes of variable stars is modeled as a superposition of sines and cosines:

$$y_i(t|f_i) = a_i \sin(2\pi f_i t) + b_i \cos(2\pi f_i t) + b_{i,0}$$

where

$a$

and

$b$

are normalization constants for the sinusoids of frequency

$f_i$

and

$b_{i,0}$

is the magnitude offset.

To find periodic variations in the data, we fit the equation above by minimizing the sum of squares, which we denote

$\chi^2$   
:

$$\chi^2 = \sum_k \frac{(d_k - y_i(t_k))^2}{\sigma_k^2}$$

where

$\sigma_k$

is the measurement uncertainty in data point

$d_k$

. We allow the mean to float, leading to more robust period estimates in the case where the period is not uniformly sampled; in these cases, the model light curve has a non-zero mean. This can be important when searching for periods on the order of the data span

$T_{\text{tot}}$

. Now, define

$$\chi^2_0 = \sum_k \frac{(d_k - \mu)^2}{\sigma_k^2}$$

where

$\mu$

is the weighted mean

$$\mu = \frac{\sum_k d_k / \sigma_k^2}{\sum_k 1 / \sigma_k^2}$$

Then, the generalized Lomb-Scargle periodogram is:

$$P_f(f) = \frac{(N-1)\chi^2_0 - \chi^2_m(f)}{2\chi^2_0}$$

where

$\chi^2_m(f)$

is

$\chi^2$

minimized with respect to

$a, b$

, and

$b_s$

.

Following Debosscher et al. (2007), we fit each light curve with a linear term plus a harmonic sinusoids:

$$y(t) = ct + \sum_{i=1}^3 \sum_{j=1}^4 y_i(t|jf_i)$$

where each of the three test frequencies

$f_i$

is allowed to have four harmonics at frequencies

$f_{ij} = jf_i$

. The three test frequencies

$f_i$

are found iteratively, by successfully finding and removing periodic signal producing a peak in

$P_f(f)$

, where

$P_f(f)$

is the Lomb-Scargle periodogram as defined above.

Given a peak in

$P_f(f)$

, we whiten the data with respect to that frequency by fitting away a model containing that frequency

well as components with frequencies at 2, 3, and 4 times that fundamental frequency (harmonics). Then

we subtract that model from the data, update

$\chi^2_s$

, and recalculate

$P_f(f)$

to find more periodic components.

Algorithm:

1) For  $i = \{1, 2, 3\}$ :

2) Calculate Lomb-Scargle periodogram

$P_f(f)$

for light curve.

3) Find peak in

$P_f(f)$

, subtract that model from data.

4) Update

$\chi^2_s$

, return to Step 1.

Then, the features extracted are given as an amplitude and a phase:

$$A_{i,j} = \sqrt{a_{i,j}^2 + b_{i,j}^2}$$

$$PH_{i,j} = \arctan\left(\frac{b_{i,j}}{a_{i,j}}\right)$$

where

$A_{i,j}$

is the amplitude of the

$j$

th harmonic of the  
 $i$

th frequency component and  
 $PH_{i,j}$

is the phase component, which we then correct to a relative phase with respect to the phase of the component:

$$PH_{i,j} = PH_{i,j} - PH_{00}$$

and remapped to  
 $[-\pi, +\pi]$

.

### Amplitudes

$A_{i,j}$

```
a = FATS.FeatureSpace(featureList=['PeriodLS', 'Freq1_harmonics_amplitude_0', 'Freq1_harmonics_amplitude_1', 'Freq1_harmonics_amplitude_2', 'Freq1_harmonics_amplitude_3', 'Freq2_harmonics_amplitude_0', 'Freq2_harmonics_amplitude_1', 'Freq2_harmonics_amplitude_2', 'Freq2_harmonics_amplitude_3', 'Freq3_harmonics_amplitude_0', 'Freq3_harmonics_amplitude_1', 'Freq3_harmonics_amplitude_2', 'Freq3_harmonics_amplitude_3'])
a=a.calculateFeature(lc_periodic)
Table(a)
```

Feature	Value
PeriodLS	10.06780
Freq1_harmonics_amplitude_0	0.10253
Freq1_harmonics_amplitude_1	0.03805
Freq1_harmonics_amplitude_2	0.00144
Freq1_harmonics_amplitude_3	0.00115
Freq2_harmonics_amplitude_0	0.09813
Freq2_harmonics_amplitude_1	0.00088
Freq2_harmonics_amplitude_2	0.00074
Freq2_harmonics_amplitude_3	0.00070
Freq3_harmonics_amplitude_0	0.03391
Freq3_harmonics_amplitude_1	0.00193
Freq3_harmonics_amplitude_2	0.00099
Freq3_harmonics_amplitude_3	0.00722

### Phase components

$PH_{i,j}$

```

a = FATS.FeatureSpace(featureList=['PeriodLS','Freq1_harmonics_amplitude_0','Freq1_harmonics_rel_phase_0','Freq1_harmonics_rel_phase_1','Freq1_harmonics_rel_phase_2','Freq1_harmonics_rel_phase_3','Freq2_harmonics_rel_phase_0','Freq2_harmonics_rel_phase_1','Freq2_harmonics_rel_phase_2','Freq2_harmonics_rel_phase_3','Freq3_harmonics_rel_phase_0','Freq3_harmonics_rel_phase_1','Freq3_harmonics_rel_phase_2','Freq3_harmonics_rel_phase_3'])
a=a.calculateFeature(lc_periodic)
Table(a)

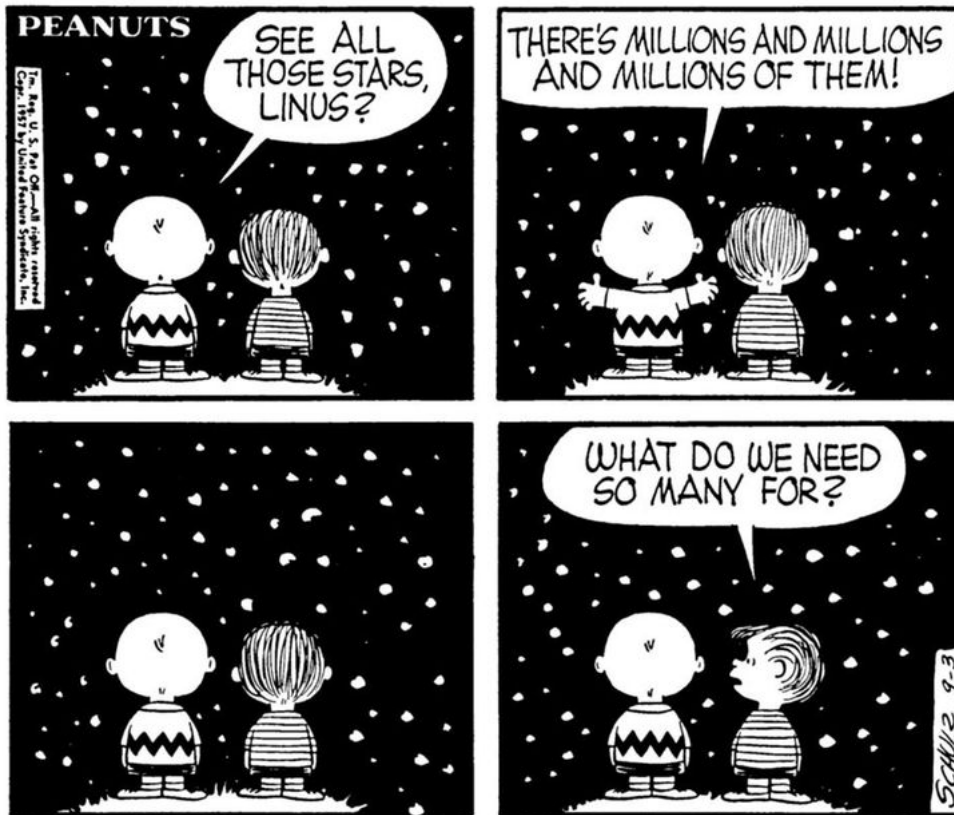
```

Feature	Value
PeriodLS	10.06780
Freq1_harmonics_amplitude_0	0.10253
Freq1_harmonics_rel_phase_0	0.00000
Freq1_harmonics_rel_phase_1	0.03540
Freq1_harmonics_rel_phase_2	-2.48334
Freq1_harmonics_rel_phase_3	-1.18346
Freq2_harmonics_rel_phase_0	0.00000
Freq2_harmonics_rel_phase_1	-1.58284
Freq2_harmonics_rel_phase_2	0.77825
Freq2_harmonics_rel_phase_3	-0.44295
Freq3_harmonics_rel_phase_0	0.00000
Freq3_harmonics_rel_phase_1	1.42454
Freq3_harmonics_rel_phase_2	0.67388
Freq3_harmonics_rel_phase_3	1.36163

## References

- [1] Falk, M., Marohn, F., Michel, R., Hofmann, D., Macke, M., Tewes, B., ... & Englert, S. (2011). A Course on Time Series Analysis: Examples with SAS. [An open source book on time series analysis SAS](#).
- [2] Kim, D. W., Protopapas, P., Alcock, C., Byun, Y. I., & Bianco, F. (2009). De-Trending Time Series Astronomical Variability Surveys. *Monthly Notices of the Royal Astronomical Society*, 397(1), 558-568. [Doi:10.1111/j.1365-2966.2009.14967.x](#).
- [3] Kim, D. W., Protopapas, P., Byun, Y. I., Alcock, C., Khardon, R., & Trichas, M. (2011). Quasi-stellar object selection algorithm using time variability and machine learning: Selection of 1620 quasi-stellar object candidates from MACHO Large Magellanic Cloud database. *The Astrophysical Journal*, 733(2), 68. [Doi:10.1088/0004-637X/735/2/68](#).
- [4] Kim, D. W., Protopapas, P., Bailer-Jones, C. A., Byun, Y. I., Chang, S. W., Marquette, J. B., & Shih, S. (2014). The EPOCH Project: I. Periodic Variable Stars in the EROS-2 LMC Database. *arXiv preprint* [Doi:10.1051/0004-6361/201323252](#).
- [5] Nun, I., Pichara, K., Protopapas, P., & Kim, D. W. (2014). Supervised Detection of Anomalous Light Curves in Massive Astronomical Catalogs. *The Astrophysical Journal*, 793(1), 23. [Doi:10.1088/0004-637X/793/1/23](#).
- [6] Pichara, K., Protopapas, P., Kim, D. W., Marquette, J. B., & Tisserand, P. (2012). An improved quasi-stellar object detection method in EROS-2 and MACHO LMC data sets. *Monthly Notices of the Royal Astronomical Society*, 427(2), 1284-1297. [Doi:10.1111/j.1365-2966.2012.22061.x](#).
- [7] Richards, J. W., Starr, D. L., Butler, N. R., Bloom, J. S., Brewer, J. M., Crellin-Quick, A., ... & Risch, M. (2011). On machine-learned classification of variable stars with sparse and noisy time-series data. *The Astrophysical Journal*, 733(1), 10. [Doi:10.1088/0004-637X/733/1/10](#).

```
picture = Image(filename='peanuts.jpg')
picture.size = (30, 30)
picture
```



## Appendix

The following section presents the tests performed to the features in order to check their invariance to unequal sampling. To do so, we take random observations of a light-curve and compare the resulting features with the ones obtained from the original data.

```

def Shuffle(mag, error, time, mag2, aligned_mag, aligned_mag2, aligned_time, aligned_error, aligned_error2):
    N = len(mag)
    shuffle = np.arange(0, N)
    index = np.random.permutation(shuffle)
    index = np.sort(index[0:N/2])

    mag_test = mag[index]
    time_test = time[index]
    error_test = error[index]

    N2 = len(mag2)
    shuffle2 = np.arange(0, N2)
    index2 = np.random.permutation(shuffle2)
    index2 = np.sort(index2[0:N2/2])

    mag2_test = mag2[index2]

    N3 = len(aligned_mag)
    shuffle3 = np.arange(0, N3)
    index3 = np.random.permutation(shuffle3)
    index3 = np.sort(index3[0:N3/2])

    aligned_mag_test = aligned_mag[index3]
    aligned_mag2_test = aligned_mag2[index3]
    aligned_time_test = aligned_time[index3]
    aligned_error_test = aligned_error[index3]
    aligned_error2_test = aligned_error2[index3]

    return mag_test, time_test, error_test, mag2_test, aligned_mag_test, aligned_mag2_test, aligned_time_test, aligned_error_test, aligned_error2_test

```

We calculate the features values for fifty random samples of the original light-curve:

```

features_value=[]
for i in xrange(50):
    [mag_test, time_test, error_test, mag2_test, aligned_mag_test, aligned_mag2_test, aligned_time_test, aligned_error_test, aligned_error2_test] = \
    Shuffle(mag, error, time, mag2, aligned_mag, aligned_mag2, aligned_time, aligned_error, aligned_error2)

    lc_test = np.array([mag_test, time_test, error_test, mag2_test, aligned_mag_test, aligned_mag2_test, aligned_time_test, aligned_error_test, aligned_error2_test])

    a = FATS.FeatureSpace(Data='all', featureList=None)
    a = a.calculateFeature(lc_test)
    features_value.append(a.result(method='array'))

```

We obtain the mean and standard deviation of each calculated feature:

```

features_value2 = np.asarray(features_value)
means=[]
stds=[]
for i in xrange(len(a.result(method='features'))):
    means.append(np.mean(features_value2[:,i]))
    stds.append(np.std(features_value2[:,i]))

```

Original light-curve:

```

a2 = FATS.FeatureSpace(Data='all', featureList=None)
a2 =a2.calculateFeature(lc)

```

Percentage difference between the original feature values and the ones obtained from the random sampling:

```

b = np.divide(np.abs((means - np.asarray(a2.result(method='array')))), stds)

```

```

import ipy_table

FeaturesList = [('Feature', 'Value')]

for i in xrange(len(b)):

    FeaturesList.append((a.result(method= 'features')[i], b[i]))

make_table(FeaturesList)
apply_theme('basic')
set_global_style(float_format='%0.3f')

```

Feature	Value
Amplitude	0.006
AndersonDarling	nan
Autocor_length	nan
Beyond1Std	0.089
CAR_mean	0.348
CAR_sigma	0.227
CAR_tau	0.311
Color	0.092
Con	0.478
Eta_color	0.742
Eta_e	0.354
FluxPercentileRatioMid20	0.285
FluxPercentileRatioMid35	0.218
FluxPercentileRatioMid50	0.112
FluxPercentileRatioMid65	0.233
FluxPercentileRatioMid80	0.083
Freq1_harmonics_amplitude_0	0.013
Freq1_harmonics_amplitude_1	0.109
Freq1_harmonics_amplitude_2	0.109
Freq1_harmonics_amplitude_3	0.081
Freq1_harmonics_rel_phase_0	nan
Freq1_harmonics_rel_phase_1	0.691
Freq1_harmonics_rel_phase_2	0.756
Freq1_harmonics_rel_phase_3	0.398
Freq2_harmonics_amplitude_0	2.020
Freq2_harmonics_amplitude_1	0.700
Freq2_harmonics_amplitude_2	0.514
Freq2_harmonics_amplitude_3	0.491
Freq2_harmonics_rel_phase_0	nan
Freq2_harmonics_rel_phase_1	2.156
Freq2_harmonics_rel_phase_2	0.393
Freq2_harmonics_rel_phase_3	1.259
Freq3_harmonics_amplitude_0	3.346
Freq3_harmonics_amplitude_1	1.286
Freq3_harmonics_amplitude_2	0.398
Freq3_harmonics_amplitude_3	0.561
Freq3_harmonics_rel_phase_0	nan



Freq3_harmonics_rel_phase_1	0.568
Freq3_harmonics_rel_phase_2	0.001
Freq3_harmonics_rel_phase_3	1.378
LinearTrend	0.140
MaxSlope	1.867
Mean	0.139
Meanvariance	0.065
MedianAbsDev	0.070
MedianBRP	1.015
PairSlopeTrend	0.607
PercentAmplitude	0.957
PercentDifferenceFluxPercentile	0.178
PeriodLS	0.174
Period_fit	0.143
Psi_CS	0.162
Psi_eta	0.436
Q31	0.174
Q31_color	0.151
Rcs	1.423
Skew	0.149
SlottedA_length	3.644
SmallKurtosis	0.003
Std	0.070
StetsonJ	0.053
StetsonK	0.102
StetsonK_AC	1.025
StetsonL	0.134

05819

Unique Total Visitors