Carpyncho-py

Release 0.3

Jun 13, 2022

Contents

1 Code **3**

2 License **5**

3 Citation **7**

4 Installation **9**
- 4.1 Installing with pip . . . 9
- 4.2 Installing the development version . . . 9

5 Documentation **11**

6 Contact **13**

7 Contents: **15**
- 7.1 Tutorials . . . 15
  - 7.1.1 Introduction . . . 15
  - 7.1.2 Command line interface (`CLI`) . . . 19
  - 7.1.3 Catalogs Tutorials . . . 21
- 7.2 API . . . 34
  - 7.2.1 carpyncho module . . . 34

8 Indices and tables **37**

Python Module Index **39**

Index **41**

Python client for Carpyncho VVV dataset collection.

This library access as a Pandas DataFrame all the data of the web version of Carpyncho https://carpyncho.github.io/.

CHAPTER 1

Code

The entire source code of is hosted in GitHub https://github.com/carpyncho/carpyncho-py/

CHAPTER 2

License

Carpyncho is under The BSD-3 License

The BSD 3-clause license allows you almost unlimited freedom with the software so long as you include the BSD copyright and license notice in it (found in Fulltext).

CHAPTER 3

Citation

If you use Carpyncho in a scientific publication, we would appreciate citations to the following paper:

> Cabral, J. B., Ramos, F., Gurovich, S., & Granitto, P. (2020). Automatic Catalog of RRLyrae from 14 million VVV Light Curves: How far can we go with traditional machine-learning? https://arxiv.org/abs/2005.00220

Bibtex entry

```
@ARTICLE{2020A&A...642A..58C,
       author = {{Cabral}, J.~B. and {Ramos}, F. and {Gurovich}, S. and {Granitto}, P.
↪~M.},
        title = "{Automatic catalog of RR Lyrae from {\ensuremath{\sim}}14 million␣
↪VVV light curves: How far can we go with traditional machine-learning?}",
      journal = {\aap},
     keywords = {methods: data analysis, methods: statistical, surveys, catalogs,␣
↪stars: variables: RR Lyrae, Galaxy: bulge, Astrophysics - Instrumentation and␣
↪Methods for Astrophysics, Astrophysics - Solar and Stellar Astrophysics, Computer␣
↪Science - Machine Learning, Statistics - Machine Learning},
         year = 2020,
        month = oct,
       volume = {642},
          eid = {A58},
        pages = {A58},
          doi = {10.1051/0004-6361/202038314},
archivePrefix = {arXiv},
       eprint = {2005.00220},
 primaryClass = {astro-ph.IM},
       adsurl = {https://ui.adsabs.harvard.edu/abs/2020A&A...642A..58C},
      adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

CHAPTER 4

Installation

This is the recommended way to install carpyncho.

4.1 Installing with pip

Make sure that the Python interpreter can load carpyncho code. The most convenient way to do this is to use virtualenv, virtualenvwrapper, and pip.

After setting up and activating the virtualenv, run the following command:

```
$ pip install carpyncho
...
```

That should be it all.

4.2 Installing the development version

If you'd like to be able to update your carpyncho code occasionally with the latest bug fixes and improvements, follow these instructions:

Make sure that you have Git installed and that you can run its commands from a shell. (Enter *git help* at a shell prompt to test this.)

Check out carpyncho main development branch like so:

```
$ git clone https://github.com/carpyncho/carpyncho-py.git carpyncho
...
```

This will create a directory *carpyncho* in your current directory.

Then you can proceed to install with the commands

```
$ cd carpyncho
$ pip install -e .
...
```

CHAPTER 5

Documentation

The full documentation of the project are available in https://carpyncho-py.readthedocs.io/

CHAPTER 6

Contact

For bugs or question please contact

Juan B. Cabral: jbcabral@unc.edu.ar

CHAPTER 7

Contents:

7.1 Tutorials

This section contains a step-by-step tutorial with examples for using the **carpyncho** tools, and how to understand several catalogs.

7.1.1 Introduction

This tutorial will show how to understand and manipilate the `carpyncho` Python Client.

First we need to import the module, and instantiate the client

[1]:
```
# import the module
import carpyncho

# instance the client
client = carpyncho.Carpyncho()
```

Firsts lets check which tiles have available catalogs to download.

[2]:
```
client.list_tiles()
```

[2]:
```
('others',
 'b206',
 'b214',
 'b216',
 'b220',
 'b228',
 'b234',
 'b247',
 'b248',
 'b261',
 'b262',
```

(continues on next page)

(continued from previous page)

```
 'b263',
 'b264',
 'b277',
 'b278',
 'b356',
 'b360',
 'b396')
```

Well lets asume we are interested in the tile `b216`, so we can check which catalogs are available in this tiles

```
[3]: client.list_catalogs("b216")
```

```
[3]: ('features', 'lc')
```

Well we see that catalogs with the light curves (`lc`), and the features of those curves (`features`) are available.

So for example we now can retrieve more info of any of this catalogs, for simplicity let's check the *b216 lc*

```
[4]: client.catalog_info("b216", "lc")
```

```
[4]: {'hname': 'Time-Series',
 'format': 'BZIP2-Parquet',
 'extension': '.parquet.bz2',
 'date': '2020-04-14',
 'md5sum': '236e126f82e80684f29247220470b831  lc_obs_b216.parquet.bz2',
 'filename': 'lc_obs_b216.parquet.bz2',
 'url': 'https://catalogs.iate.conicet.unc.edu.ar/carpyncho/lcurves/b216/lc_obs_b216.
↪parquet.bz2',
 'size': 369866999,
 'records': 37839384}
```

The attribute `hname` is a human readable version of the name of the catalog, the next two keys have information of format of the catalog (how is stored in the cloud), next are information about the date of publication of the file, check-sums and the cloud-ID (all of this is mostly for internal use).

Finally we have the two more important information: `size` is the size in bytes of the file (*352.7 MiB*) and the number of records stored in the file (more than 37 millons).

Ok... to big, lets check the *b278 features* catalog

```
[5]: client.catalog_info("b216", "features")
```

```
[5]: {'hname': 'Features',
 'format': 'BZIP2-Parquet',
 'extension': '.parquet.bz2',
 'date': '2020-04-14',
 'md5sum': '433aae05541a2f5b191aa95d717fa83c  features_b216.parquet.bz2',
 'filename': 'features_b216.parquet.bz2',
 'url': 'https://catalogs.iate.conicet.unc.edu.ar/carpyncho/lcurves/b216/features_
↪b216.parquet.bz2',
 'size': 149073679,
 'records': 334773}
```

In this case this file is only `142.2 MiB` of size, let's retrive it into a dataframe.

```
[6]: # the first time this can be slow
df = client.get_catalog("b278", "features")
df
```

```
b278-features: 349MB [02:23, 2.42MB/s]
```

[6]:

```
                     id  cnt        ra_k      dec_k vs_type vs_catalog  \
0       32780000001647   33  270.675437 -30.833556
1       32780000001722   32  270.601058 -30.797561
2       32780000001725   31  270.586525 -30.790697
3       32780000001764   35  270.533529 -30.764936
4       32780000001766   49  270.575246 -30.785039
...                ...  ...         ...        ...      ...         ...
866878  32780000865946   56  270.961067 -29.417511
866879  32780000879797   56  270.962108 -29.392694
866880  32780000894698   57  270.987062 -29.378722
866881  32780000894881   55  272.064012 -29.893197
866882  32780000900244   56  272.123942 -29.912258

        Amplitude  Autocor_length  Beyond1Std  Con  ...  c89_jk_color  \
0          0.4205             1.0    0.303030  0.0  ...      0.254668
1          0.2815             1.0    0.250000  0.0  ...      0.714901
2          0.7770             1.0    0.225806  0.0  ...      0.737901
3          0.6025             1.0    0.200000  0.0  ...      0.708924
4          0.4850             1.0    0.224490  0.0  ...      0.587902
...           ...             ...         ...  ...  ...           ...
866878     0.0230             1.0    0.357143  0.0  ...      0.687494
866879     0.0260             1.0    0.375000  0.0  ...      0.655494
866880     0.0310             1.0    0.350877  0.0  ...      0.743494
866881     0.0285             1.0    0.381818  0.0  ...      0.464515
866882     0.0280             1.0    0.267857  0.0  ...      0.706600

           c89_m2     c89_m4    n09_c3  n09_hk_color  n09_jh_color  \
0       14.690558  14.666523  0.183141      0.026877      0.229526
1       14.020039  13.975850  0.344830      0.136610      0.580695
2       12.123443  12.090208  0.228780      0.187609      0.552696
3       12.578053  12.528972  0.398169      0.114875      0.596295
4       13.324888  13.287657  0.285994      0.111611      0.478696
...           ...        ...       ...           ...           ...
866878  12.213607  12.171067  0.347818      0.125109      0.563573
866879  12.358619  12.329497  0.212287      0.163109      0.493573
866880  10.892837  10.854592  0.297559      0.164110      0.580572
866881  13.531632  13.505489  0.203314      0.096373      0.369172
866882  12.044150  12.008327  0.273388      0.158705      0.549095

        n09_jk_color     n09_m2     n09_m4      ppmb
0           0.256404  14.820497  14.825871  0.000044
1           0.717305  14.243584  14.253669  5.275137
2           0.740305  12.351745  12.358436  3.785030
3           0.711170  12.798067  12.809809  6.580914
4           0.590306  13.522170  13.530534  5.746016
...              ...        ...        ...       ...
866878      0.688682  12.398301  12.408567  2.734307
866879      0.656682  12.536672  12.542938  6.877899
866880      0.744682  11.089152  11.097934  5.883032
866881      0.465545  13.667806  13.673903  5.310308
866882      0.707800  12.229126  12.236734  0.922049

[866883 rows x 73 columns]
```

Well we have a lot of imformation to play here. Let's check if we have some multiple types of sources

[7]:
```
df.groupby("vs_type").id.count()
```

[7]:
```
vs_type
                857450
BLAP                 1
CV-DN                5
Cep-1                1
Cep-F                1
ECL-C              729
ECL-ELL            486
ECL-NC            3246
LPV-Mira           104
LPV-OSARG         3820
LPV-SRV            592
RRLyr-RRab         289
RRLyr-RRc          145
RRLyr-RRd            3
SP_ECL-C             1
SP_ECL-NC            1
T2Cep-BLHer          6
T2Cep-RVTau          2
T2Cep-WVir           1
Name: id, dtype: int64
```

Well 41 RRab stars (and more than 334K of unknow sources)

Well we have a lot to use here, lets make some plots.

Form now on, yo simple have a big pandas dataframe to manipulate.

All the methods of `carpyncho.Carpyncho` client are well documented and you can acces it whit the '?' command in Jupyter

[8]:
```
client.get_catalog?
```

```
Signature: client.get_catalog(tile, catalog, force=False)
Docstring:
Retrieve a catalog from the carpyncho dataset.

Parameters
----------
tile: str
    The name of the tile.
catalog:
    The name of the catalog.
force: bool (default=False)
    If its True, the cached version of the catalog is ignored and
    redownloaded. Try to always set force to False.

Returns
-------
pandas.DataFrame:
    The columns of the DataFrame changes between the different catalog.

Raises
------
ValueError:
    If the tile or the catalog is not found.
IOError:
```

(continues on next page)

(continued from previous page)

```
    If the checksum not match.
File:       ~/proyectos/carpyncho-py/src/carpyncho.py
Type:       method
```

[9]:
```
import datetime as dt
dt.datetime.now()
```

[9]:
```
datetime.datetime(2022, 6, 13, 0, 1, 52, 183122)
```

7.1.2 Command line interface (CLI)

After install Carpyncho you gonna have available command line app to download any dataset.

[1]:
```
carpyncho --help
```

```
Usage: carpyncho command [args...]

Carpyncho console client.

Explore and download the entire https://carpyncho.github.io/ catalogs from your
command line.

Commands:
  catalog-info       Retrieve the information about a given catalog.
  download-catalog   Retrives a catalog from th Carpyncho dataset collection.
  has-catalog        Check if a given catalog and tile exists.
  list-catalogs      Show the available catalogs for a given tile.
  list-tiles         Show available tiles.
  version            Print Carpyncho version.

This software is under the BSD 3-Clause License. Copyright (c) 2020, Juan
Cabral. For bug reporting or other instructions please check:
https://github.com/carpyncho/carpyncho-py
```

To list all availables tiles we can run

[2]:
```
carpyncho list-tiles
```

```
- b206
- b214
- b216
- b220
- b228
- b234
- b247
- b248
- b261
- b262
- b263
- b264
- b277
- b278
- b356
- b360
- b396
```

Then we can check all the available catalogs for a given tile (`b216` for example)

[3]:
```
carpyncho list-catalogs b216
```

```
Tile b216
    - features
    - lc
```

Lets asume we want to download the catalog *features* from the tile *b216*. First lets check how big is the catalog before download:

[4]:
```
carpyncho catalog-info b216 features
```

```
Catalog b216-features
    - hname: Features
    - format: BZIP2-Parquet
    - extension: .parquet.bz2
    - date: 2020-04-14
    - md5sum: 433aae05541a2f5b191aa95d717fa83c  features_b216.parquet.bz2
    - filename: features_b216.parquet.bz2
    - driveid: 1-t165sLjn0k507SFeW-A4p9wYVL9rP4B
    - size: 142.2 MiB
    - records: 334,773
```

Well `142 MiB` for `334773` rows in the table, lets download it and sotore it in `csv` format

[5]:
```
carpyncho download-catalog b216 features --out b216_features.csv
```

```
b216-features: 149MB [03:03, 811kB/s]
Writing b216_features.csv...
```

Now lets check the size and the checksum to see if it's correct (warning this is linux and mac only)

[7]:
```
cat b216_features.csv | wc -l
```

```
334774
```

The rows are ok, so it's done.

If you run the same command multiple times, the file will be cached.

All the commands support more options yo can check it with `carpyncho <command> --help`. For example

[11]:
```
carpyncho download-catalog --help
```

```
Usage: carpyncho download-catalog [OPTIONS] tile catalog

Retrives a catalog from th Carpyncho dataset collection.

Arguments:
  tile         The name of the tile.
  catalog      The name of the catalog.

Options:
  --out=STR    Path to store the catalog. The extension of the file detemines
               the format. Options are ".xlsx" (Excel), ".csv", ".pkl" (Python
               pickle) and ".parquet".
  --force      Force to ignore the cached value and redownload the catalog. Try
               to always set force to False.

Other actions:
  -h, --help   Show the help
```

[12]:
```
date
```

```
jue abr 23 22:38:42 -03 2020
```

7.1.3 Catalogs Tutorials

This section contains information on how to interpret and manipulate the catalogs offered by carpyncho.

Light-Curves catalogs tutorial (`lc`)

This notebook give some insights about the data stored in all the `lc` types catalogs.

[1]:
```
# import the module and instance the client
import carpyncho
client = carpyncho.Carpyncho()
```

[2]:
```
df = client.get_catalog("b214", "lc")
df.sample(3)
```

```
b214-lc: 222MB [01:33, 2.37MB/s]
```

[2]:
```
               bm_src_id  pwp_id  pwp_stack_src_id  pwp_stack_src_hjd  \
10436285  32140000106633    4707  3000470700094773       56110.056177
17718411  32140000156103    4719  3000471900157271       55435.200210
20185341  32140000399685    4726  3000472600347001       56214.078852

          pwp_stack_src_mag3  pwp_stack_src_mag_err3
10436285              15.774                   0.053
17718411              17.899                   0.277
20185341              16.240                   0.073
```

The columns of this catalog are

[3]:
```
print(list(df.columns))
```

```
['bm_src_id', 'pwp_id', 'pwp_stack_src_id', 'pwp_stack_src_hjd', 'pwp_stack_src_mag3',
↪ 'pwp_stack_src_mag_err3']
```

Where

- **bm_src_id** (Band-Merge Source ID): This is the unique identifier of every light curve. The records with the same *bm_src_id* are part of the same lc (This id is part of Carpyncho internal and is unique for every source).
- **pwp_id** (Pawprint Stack ID): The id of the pawprint where this point of the light curve is located (This id is part of Carpyncho internal database).
- **pwp_stack_src_id** (Pawprint Stack Source ID): The id of this particular observation inside the pawprint where this point (This id are part of Carpyncho internal database)
- **pwp_stack_src_hjd** (Pawprint Stack Source HJD): The Heliocentric-Julian-Date of this particular observation.
- **pwp_stack_src_mag3** (Pawprint Stack Source Magnitude of the 3rd Aperture): The magnitude (of the 3rd aperture) of this particular observation.
- **pwp_stack_src_mag_err3** (Pawprint Stack Source Magnitude Error of the 3rd Aperture): The magnitude error (of the 3rd aperture) of this particular observation.

Retrieve a single light-curve

Lets, for example, retrieve the LC with the ID `32140000349109` and sort by time

[4]:
```
lc = df[df.bm_src_id == 32140000349109]
lc = lc.sort_values("pwp_stack_src_hjd")
lc
```

[4]:
```
               bm_src_id  pwp_id  pwp_stack_src_id  pwp_stack_src_hjd  \
9824315   32140000349109    4705  3000470500316153        55301.355623
15823573  32140000349109    4713  3000471300371137        55404.204420
17889825  32140000349109    4719  3000471900344310        55435.200224
15383198  32140000349109    4711  3000471100264253        55497.035605
7230797   32140000349109    4694  3000469400252478        55806.201062
...                  ...     ...               ...                ...
17498325  32140000349109    4718  3000471800253351        57248.183650
20689532  32140000349109    4728  3000472800102886        57251.230468
7873271   32140000349109    4697  3000469700250665        57252.177815
12365892  32140000349109    4677  3000467700294240        57265.086562
15027768  32140000349109    4686  3000468600357522        57282.060899

          pwp_stack_src_mag3  pwp_stack_src_mag_err3
9824315               15.736                   0.045
15823573              15.705                   0.040
17889825              15.734                   0.042
15383198              15.868                   0.061
7230797               15.795                   0.060
...                      ...                     ...
17498325              15.750                   0.050
20689532              15.828                   0.099
7873271               15.727                   0.050
12365892              15.773                   0.055
15027768              15.748                   0.045

[67 rows x 6 columns]
```

Great, 67 epochs. Let's check the average and dispersion of the magnitudes and the error

[5]:
```
lc[['pwp_stack_src_mag3', 'pwp_stack_src_mag_err3']].mean()
```

[5]:
```
pwp_stack_src_mag3        15.759224
pwp_stack_src_mag_err3     0.051299
dtype: float64
```

[6]:
```
lc[['pwp_stack_src_mag3', 'pwp_stack_src_mag_err3']].std()
```

[6]:
```
pwp_stack_src_mag3        0.044732
pwp_stack_src_mag_err3    0.009130
dtype: float64
```

The source is stable, now check the observation range

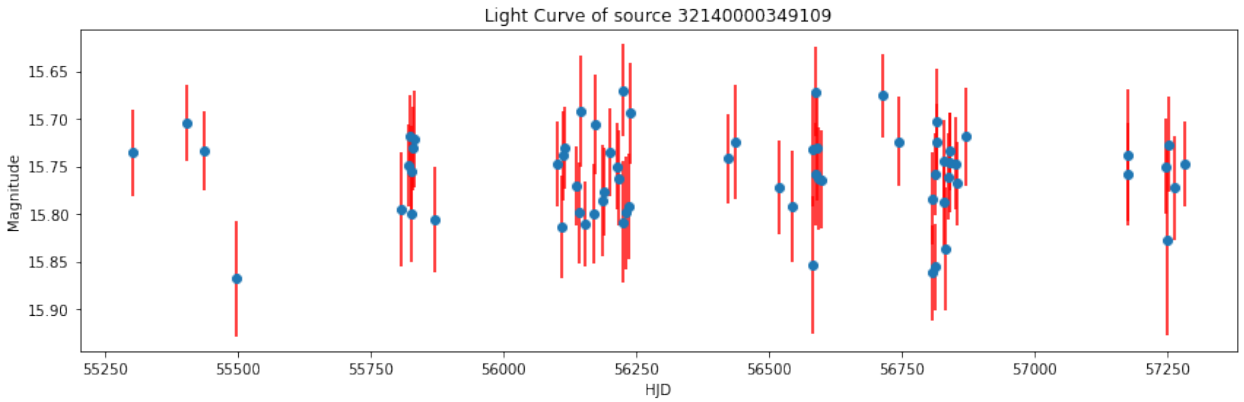
[7]:
```
print((lc.pwp_stack_src_hjd.max() - lc.pwp_stack_src_hjd.min()) / 365, "Years")
```

```
5.426589796388058 Years
```

Finally we can plot the entire LC

[8]:
```
import matplotlib.pyplot as plt
```

[9]:
```
fig, ax = plt.subplots(figsize=(12, 4))

ax.errorbar(
    lc.pwp_stack_src_hjd,
    lc.pwp_stack_src_mag3,
    lc.pwp_stack_src_mag_err3,
    ls="", marker="o", ecolor="red")

ax.set_title(f"Light Curve of source 32140000349109")
ax.set_ylabel("Magnitude")
ax.set_xlabel("HJD")

ax.invert_yaxis()
fig.tight_layout()
```



[10]:
```
import datetime as dt
dt.datetime.now()
```

[10]:
```
datetime.datetime(2022, 6, 13, 0, 7, 52, 200684)
```

[ ]:
```

```

Features catalogs tutorial (`features`)

This notebook give some insights about the data stored in all the `features` types catalogs.

[1]:
```
# import the module and instance the client
import carpyncho
client = carpyncho.Carpyncho()
```

Now we download one features catalog

[2]:
```
df = client.get_catalog("b214", "features")
df.sample(3)
```

```
b214-features: 159MB [01:06, 2.40MB/s]
```

[2]:
```
                     id  cnt        ra_k      dec_k vs_type vs_catalog  \
246230  32140000304888   66  281.886621 -24.814519
```

(continues on next page)

(continued from previous page)

```
375156  32140000459091   44  281.626075 -24.136333
264628  32140000328959   39  281.817479 -24.701575

        Amplitude  Autocor_length  Beyond1Std  Con  ...  c89_jk_color  \
246230    0.18675             1.0    0.287879  0.0  ...      0.790576
375156    0.27050             1.0    0.431818  0.0  ...      0.475020
264628    0.28200             1.0    0.333333  0.0  ...      0.498984

           c89_m2     c89_m4    n09_c3  n09_hk_color  n09_jh_color  \
246230  15.292760  15.256852  0.274868      0.188890      0.602440
375156  16.945470  16.877186  0.682651     -0.076031      0.551810
264628  16.798042  16.762198  0.305437      0.071062      0.428715

        n09_jk_color     n09_m2     n09_m4      ppmb
246230      0.791330  15.479110  15.486869  2.341868
375156      0.475779  17.051519  17.072001  4.627671
264628      0.499777  16.923741  16.932339  0.928964

[3 rows x 73 columns]
```

The columns of this catalog are

[3]:
```
print(list(df.columns))
```

```
['id', 'cnt', 'ra_k', 'dec_k', 'vs_type', 'vs_catalog', 'Amplitude', 'Autocor_length',
↪ 'Beyond1Std', 'Con', 'Eta_e', 'FluxPercentileRatioMid20', 'FluxPercentileRatioMid35
↪', 'FluxPercentileRatioMid50', 'FluxPercentileRatioMid65', 'FluxPercentileRatioMid80
↪', 'Freq1_harmonics_amplitude_0', 'Freq1_harmonics_amplitude_1', 'Freq1_harmonics_
↪amplitude_2', 'Freq1_harmonics_amplitude_3', 'Freq1_harmonics_rel_phase_0', 'Freq1_
↪harmonics_rel_phase_1', 'Freq1_harmonics_rel_phase_2', 'Freq1_harmonics_rel_phase_3
↪', 'Freq2_harmonics_amplitude_0', 'Freq2_harmonics_amplitude_1', 'Freq2_harmonics_
↪amplitude_2', 'Freq2_harmonics_amplitude_3', 'Freq2_harmonics_rel_phase_0', 'Freq2_
↪harmonics_rel_phase_1', 'Freq2_harmonics_rel_phase_2', 'Freq2_harmonics_rel_phase_3
↪', 'Freq3_harmonics_amplitude_0', 'Freq3_harmonics_amplitude_1', 'Freq3_harmonics_
↪amplitude_2', 'Freq3_harmonics_amplitude_3', 'Freq3_harmonics_rel_phase_0', 'Freq3_
↪harmonics_rel_phase_1', 'Freq3_harmonics_rel_phase_2', 'Freq3_harmonics_rel_phase_3
↪', 'Gskew', 'LinearTrend', 'MaxSlope', 'Mean', 'Meanvariance', 'MedianAbsDev',
↪'MedianBRP', 'PairSlopeTrend', 'PercentAmplitude', 'PercentDifferenceFluxPercentile
↪', 'PeriodLS', 'Period_fit', 'Psi_CS', 'Psi_eta', 'Q31', 'Rcs', 'Skew',
↪'SmallKurtosis', 'Std', 'StetsonK', 'c89_c3', 'c89_hk_color', 'c89_jh_color', 'c89_
↪jk_color', 'c89_m2', 'c89_m4', 'n09_c3', 'n09_hk_color', 'n09_jh_color', 'n09_jk_
↪color', 'n09_m2', 'n09_m4', 'ppmb']
```

Where

- **id** (ID): This is the unique identifier of every light curve. If you want to access all the points of the lightcurve of a source wiht any *id*, you can search for the same value of a `bm_src_id`in the `lc` of the same tile of the features catalog.
- **cnt** (Count): How many epochs has the lightcurve.
- **ra_k**: Right Ascension in band K_s of the source in the first epoch.
- **dec_k**: Declination in band K_s of the source in the first epoch.
- **vs_type** (Variable Star Type): The type of the source if is a variable star tagged with the OGLE-III, OGLE-IV and VIZIER catalogs; or empty if the source has no type.
- **vs_catalog** (Variable Star Catalog): From which catalog the *vs_type* was extracted.

All the other columns (Except the last 13) are the features itself And can be consulted here https://feets.readthedocs.io/en/latest/tutorial.html#The-Features

Finally the reddening free features are:

- **c89_c3**: $C3$ Pseudo-color using the cardelli-89 extinction law.
- **c89_ab_color**: Magnitude difference in the first epoch between the band a and the band b using the Cardelli-89 extinction law. Where a and b can be the bands H, J and K_s.
- **c89_m2** and **c89_m4**: $m2$ and $m4$ pseudo-magnitudes using the Cardelli-89 extinction law.
- **n09_c3**: $C3$ Pseudo-color using the Nishiyama-09 extinction law.
- **n09_ab_color**: Magnitude difference in the first epoch between the band a and the band b using the nishiyama-09 extinction law. Where a and b can be the bands H, J and K_s
- **n09_m2** and **n09_m4**: $m2$ and $m4$ pseudo-magnitudes using the nishiyama-09 extinction law.
- **ppmb** (Pseudo-Phase Multi-Band): This index sets the first time in phase with respect to the average time in all bands, using the period calculated by *feets*.

$$PPMB = frac(\frac{|mean(HJD_H, HJD_J, HJD_{K_s}) - T_0|}{P})$$

Where HJD_H, HJD_J and HJD_{K_s} are the time of observations in the band H, J and K_s; T_0 is the time of observation of maximum magnitude in K_s band; $mean$ calculate the mean of the three times, $frac$ returns only the decimal part of the number, and P is the extracted period.

For more information about the extintion laws and pseudo colors/magnitudes:

Cardelli-89 Extinction law:

Cardelli, J. A., Clayton, G. C., & Mathis, J. S. (1989). The relationship between infrared, optical, and ultraviolet extinction. The Astrophysical Journal, 345, 245-256.

Nishiyama-09 Extinction law:

Nishiyama, S., Tamura, M., Hatano, H., Kato, D., Tanabé, T., Sugitani, K., & Nagata, T. (2009). Interstellar extinction law toward the galactic center III: J, H, KS bands in the 2MASS and the MKO systems, and 3.6, 4.5, 5.8, 8.0 μm in the Spitzer/IRAC system. The Astrophysical Journal, 696(2), 1407.

Pseudo colors/magnitudes:

Catelan, M., Minniti, D., Lucas, P. W., Alonso-Garcia, J., Angeloni, R., Beamin, J. C., ... & Dekany, I. (2011). The Vista Variables in the Via Lactea (VVV) ESO Public Survey: Current Status and First Results. arXiv preprint arXiv:1105.1119.

Well lets play with the data of 3 Variable star

[4]:
```
rrs = df[df.vs_type == "RRLyr-RRab"][:3]
rrs
```

[4]:
```
                   id  cnt        ra_k      dec_k     vs_type vs_catalog  \
4456   32140000002913   68  280.582129 -25.299033  RRLyr-RRab     vizier
21827  32140000030432   68  280.482488 -25.156328  RRLyr-RRab     vizier
21929  32140000030564   68  281.279492 -25.499744  RRLyr-RRab     vizier

       Amplitude  Autocor_length  Beyond1Std  Con  ...  c89_jk_color  \
4456     0.12600             1.0    0.279412  0.0  ...      0.162897
21827    0.17275             1.0    0.235294  0.0  ...      0.221322
21929    0.18400             1.0    0.308824  0.0  ...      0.314535
```

(continues on next page)

(continued from previous page)

```
          c89_m2     c89_m4    n09_c3  n09_hk_color  n09_jh_color  \
4456   13.827993  13.819663  0.053522      0.040231      0.123061
21827  13.112390  13.104045  0.049464      0.062893      0.158674
21929  14.001622  13.984766  0.128221      0.068086      0.246717

       n09_jk_color     n09_m2     n09_m4      ppmb
4456       0.163291  13.893530  13.895081  1.629816
21827      0.221566  13.185733  13.187114  1.672651
21929      0.314803  14.087323  14.090839  2.006219

[3 rows x 73 columns]
```

We can check their mean of magnitudes to check if the source is not saturated or diffuse.

[5]:
```
rrs.Mean
```

[5]:
```
4456     14.086691
21827    13.488265
21929    14.412221
Name: Mean, dtype: float64
```

The three are between 12 an 16.5, so they are ok, and their pulsation?

[6]:
```
rrs.Std
```

[6]:
```
4456     0.071391
21827    0.090222
21929    0.120290
Name: Std, dtype: float64
```

Plotting time: we need the lc catalog to show the phased-folded light curve.

[7]:
```
lcs = client.get_catalog("b214", "lc")
```

Now to reduce the memory footprint we can retrieve the lc for only our 3 selected stars

[8]:
```
lcs = lcs[lcs.bm_src_id.isin(rrs.id)]
```

For make our code simple we can use to folde the light curve the PyAstronomy and numpy library

[9]:
```
from PyAstronomy.pyasl import foldAt
import numpy as np
```

[10]:
```
%matplotlib inline
import matplotlib.pyplot as plt
```

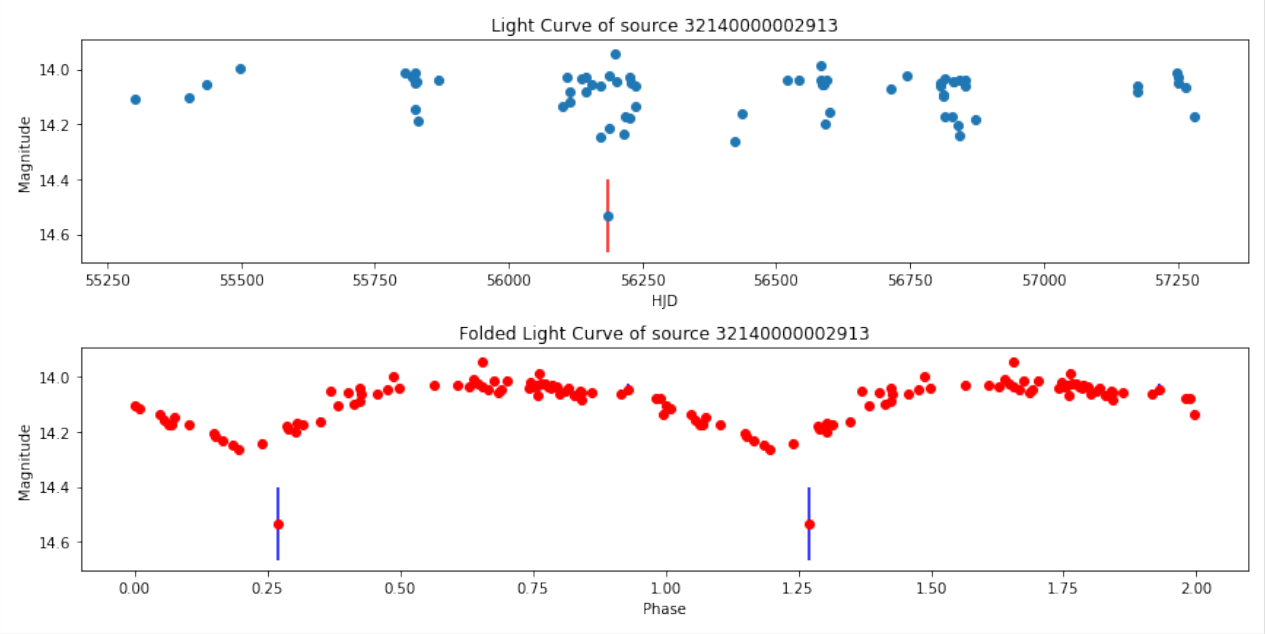
now we can plot the folded and unfolded lightcurves

[11]:
```
# get one ot the 3 sources
rr = rrs.iloc[0]

# retrieve the lightcurve for this rr
lc = lcs[lcs.bm_src_id == rr.id]

# sort by time
lc = lc.sort_values("pwp_stack_src_hjd")
```

(continues on next page)

(continued from previous page)

```
# split in time, magnitude and error
time, mag, err = (
        lc.pwp_stack_src_hjd.values,
        lc.pwp_stack_src_mag3.values,
        lc.pwp_stack_src_mag_err3.values)

# t0 is the first time
t0 = time[0]

# fold
phases = foldAt(time, rr.PeriodLS, T0=t0)
sort = np.argsort(phases)
phases, pmag, perr = phases[sort], mag[sort], err[sort]

# duplicate the values in two phases
phases = np.hstack((phases, phases + 1))
pmag = np.hstack((pmag, pmag))
perr = np.hstack((perr, perr))

# now create two plot for the folded and the unfolde LC
fig, axes = plt.subplots(2, 1, figsize=(12, 6))

# first lets plot the unfolded lc
ax = axes[0]
ax.errorbar(time, mag, err, ls="", marker="o", ecolor="red")
ax.set_title(f"Light Curve of source {rr.id}")
ax.set_ylabel("Magnitude")
ax.set_xlabel("HJD")
ax.invert_yaxis()

# now the folded lc
ax = axes[1]
ax.errorbar(phases, pmag, perr, ls="", marker="o", ecolor="blue", color="red")
ax.set_title(f"Folded Light Curve of source {rr.id}")
ax.set_ylabel("Magnitude")
ax.set_xlabel("Phase")
ax.invert_yaxis()

fig.tight_layout()
```



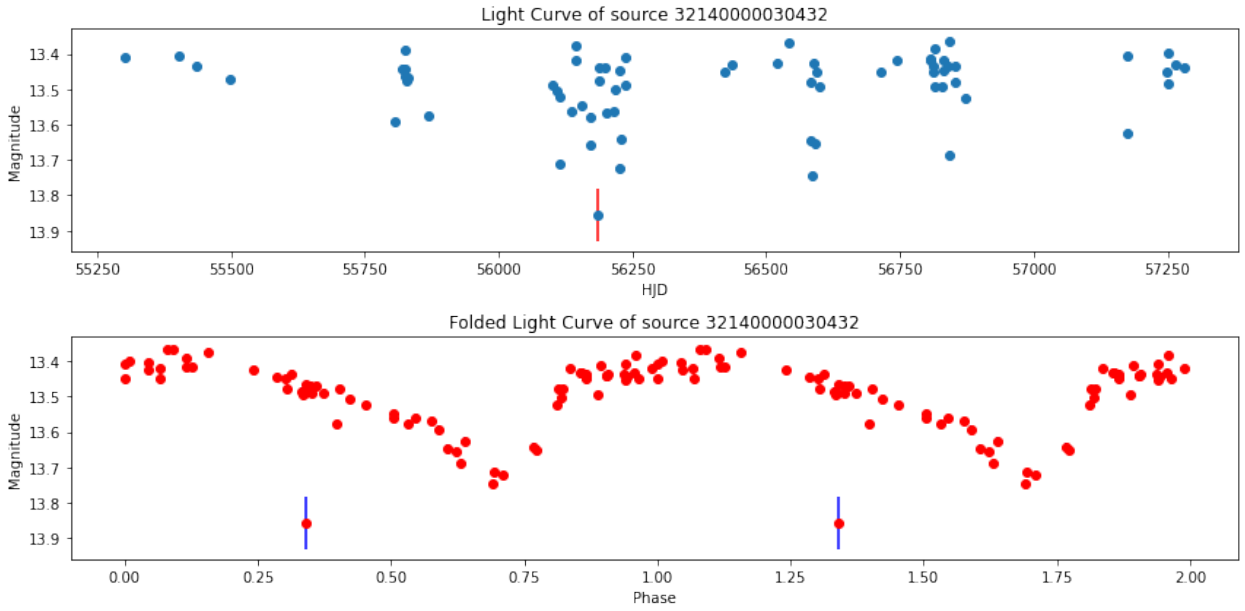
The next light curve

[12]:
```
rr = rrs.iloc[1]
lc = lcs[lcs.bm_src_id == rr.id]
lc = lc.sort_values("pwp_stack_src_hjd")

time, mag, err = (
        lc.pwp_stack_src_hjd.values,
        lc.pwp_stack_src_mag3.values,
        lc.pwp_stack_src_mag_err3.values)

t0 = time[0]

phases = foldAt(time, rr.PeriodLS, T0=t0)
sort = np.argsort(phases)
phases, pmag, perr = phases[sort], mag[sort], err[sort]

phases = np.hstack((phases, phases + 1))
pmag = np.hstack((pmag, pmag))
perr = np.hstack((perr, perr))


fig, axes = plt.subplots(2, 1, figsize=(12, 6))

ax = axes[0]
ax.errorbar(time, mag, err, ls="", marker="o", ecolor="red")
ax.set_title(f"Light Curve of source {rr.id}")
ax.set_ylabel("Magnitude")
ax.set_xlabel("HJD")
ax.invert_yaxis()

ax = axes[1]
ax.errorbar(phases, pmag, perr, ls="", marker="o", ecolor="blue", color="red")
ax.set_title(f"Folded Light Curve of source {rr.id}")
ax.set_ylabel("Magnitude")
```

(continues on next page)

(continued from previous page)

```
ax.set_xlabel("Phase")
ax.invert_yaxis()

fig.tight_layout()
```



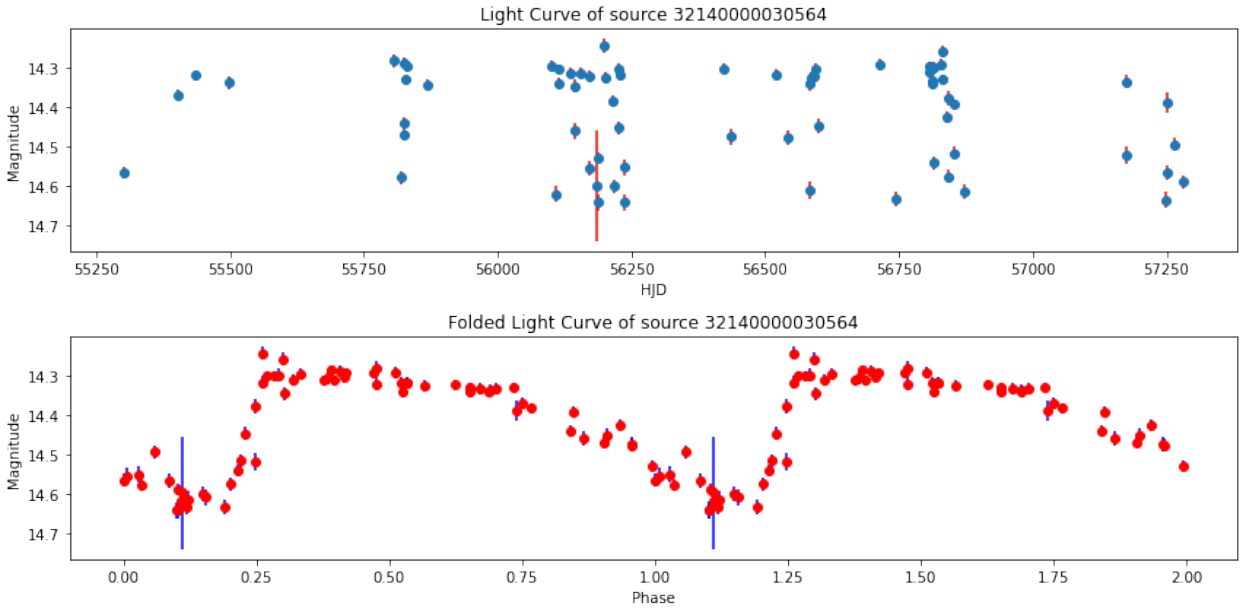
And the final one

[13]:
```
rr = rrs.iloc[2]
lc = lcs[lcs.bm_src_id == rr.id]
lc = lc.sort_values("pwp_stack_src_hjd")

time, mag, err = (
        lc.pwp_stack_src_hjd.values,
        lc.pwp_stack_src_mag3.values,
        lc.pwp_stack_src_mag_err3.values)

t0 = time[0]

phases = foldAt(time, rr.PeriodLS, T0=t0)
sort = np.argsort(phases)
phases, pmag, perr = phases[sort], mag[sort], err[sort]

phases = np.hstack((phases, phases + 1))
pmag = np.hstack((pmag, pmag))
perr = np.hstack((perr, perr))


fig, axes = plt.subplots(2, 1, figsize=(12, 6))

ax = axes[0]
ax.errorbar(time, mag, err, ls="", marker="o", ecolor="red")
ax.set_title(f"Light Curve of source {rr.id}")
ax.set_ylabel("Magnitude")
ax.set_xlabel("HJD")
ax.invert_yaxis()
```

(continues on next page)

(continued from previous page)

```
ax = axes[1]
ax.errorbar(phases, pmag, perr, ls="", marker="o", ecolor="blue", color="red")
ax.set_title(f"Folded Light Curve of source {rr.id}")
ax.set_ylabel("Magnitude")
ax.set_xlabel("Phase")
ax.invert_yaxis()

fig.tight_layout()
```



[14]:
```
import datetime as dt
dt.datetime.now()
```

[14]:
```
datetime.datetime(2022, 6, 13, 0, 13, 25, 563015)
```

[ ]:
```

```

Carpyncho RR-Lyrae V.1.0 Catalogs(`cp_rr_v1`)

This notebook give some insights about the data stored in all the `features` types catalogs.

[1]:
```
# import the module and instance the client
import carpyncho
client = carpyncho.Carpyncho()
```

Now we download te Carpyncho RR-Lyrae V1 Catalog

[2]:
```
df = client.get_catalog("others", "cpy_rr_v1")
df
```

```
others-cpy_rr_v1: 32.8kB [00:00, 1.80MB/s]
```

[2]:
```
                      id  tile  cnt        ra_k      dec_k      prob  \
6799063  33960000211620  b396  131  267.549917 -18.699892  0.852000
```

(continues on next page)

(continued from previous page)

```
7153972  33960000942530  b396  130  268.118017 -17.763556  0.849467
6971905  33960000566886  b396  131  267.536346 -18.093889  0.837733
6801853  33960000220135  b396  131  267.637892 -18.734767  0.837333
6747151  33960000105195  b396  131  267.487762 -18.848939  0.828400
...                  ...   ...  ...         ...        ...       ...
6457949  33600000787886  b360  139  263.635417 -29.303825  0.462667
6926060  33960000470380  b396  131  267.525504 -18.250553  0.462000
6874859  33960000359818  b396  131  267.490592 -18.416206  0.460933
682776   32200000656825  b220  123  274.725521 -34.229878  0.460933
6617136  33600000965623  b360  204  263.473292 -28.911094  0.460533

                                                    tsample
6799063  b206, b214, b216, b220, b228, b234, b247, b248...
7153972  b206, b214, b216, b220, b228, b234, b247, b248...
6971905  b206, b214, b216, b220, b228, b234, b247, b248...
6801853  b206, b214, b216, b220, b228, b234, b247, b248...
6747151  b206, b214, b216, b220, b228, b234, b247, b248...
...                                                    ...
6457949  b206, b214, b216, b220, b228, b234, b247, b248...
6926060  b206, b214, b216, b220, b228, b234, b247, b248...
6874859  b206, b214, b216, b220, b228, b234, b247, b248...
682776   b206, b214, b216, b228, b234, b247, b248, b261...
6617136  b206, b214, b216, b220, b228, b234, b247, b248...

[242 rows x 7 columns]
```

The columns of this catalog are

[3]:
```
print(list(df.columns))
```

```
['id', 'tile', 'cnt', 'ra_k', 'dec_k', 'prob', 'tsample']
```

Where

- **id** (ID): This is the unique identifier of every light curve. If you want to access all the points of the lightcurve of a source wiht any *id*, you can search for the same value of a `bm_src_id`in the `lc`, or `id` in the `features` catalog of the same tile indicated in the column `tile`.
- **tile**: The name of the tile where the candidate is located.
- **cnt** (Count): How many epochs has the lightcurve.
- **ra_k**: Right Ascension in band K_s of the source in the first epoch.
- **dec_k**: Declination in band K_s of the source in the first epoch.
- **prob** (Probability): The probability of this source to be a RR-Lyrae star [1].
- **tsample** (Tiles-Sample): Which tiles was used to create the ensemble select this source as a candidate [1].

[1] To more insights about this feature please chek our work

> Not ready

Well lets play with a candidate

[4]:
```
rr = df.iloc[0]
rr
```

[4]:
```
id                                    33960000211620
tile                                            b396
```

(continues on next page)

(continued from previous page)

```
cnt                                                       131
ra_k                                                267.549917
dec_k                                               -18.699892
prob                                                     0.852
tsample    b206, b214, b216, b220, b228, b234, b247, b248...
Name: 6799063, dtype: object
```

We can check their mean of magnitudes to check if the source is not saturated or diffuse.

Now we know id from the tile b396 we can retrieve the entire collection of features and the light-curve

[5]:
```
feats = client.get_catalog("b396", "features")
lc = client.get_catalog("b396", "lc")

# retrieve the features of the selected source
feats = feats[feats.id == rr.id]
lc = lc[lc.bm_src_id == rr.id]
```

```
b396-features: 303MB [02:10, 2.32MB/s]
b396-lc: 829MB [05:58, 2.32MB/s]
```

No we have the features

[6]:
```
feats
```

[6]:
```
                     id  cnt        ra_k      dec_k vs_type vs_catalog  \
144267  33960000211620  131  267.549917 -18.699892

        Amplitude  Autocor_length  Beyond1Std  Con  ...  c89_jk_color  \
144267      0.178             2.0    0.274809  0.0  ...      0.139307

           c89_m2     c89_m4    n09_c3  n09_hk_color  n09_jh_color  \
144267  13.758278  13.748731  0.039034      0.037559      0.103765

        n09_jk_color   n09_m2     n09_m4     ppmb
144267      0.141323  13.8797  13.880858  1.49232

[1 rows x 73 columns]
```

and the entire lc

[7]:
```
lc
```

[7]:
```
               bm_src_id  pwp_id  pwp_stack_src_id  pwp_stack_src_hjd  \
752483    33960000211620    2753  3000275300031172       56152.044089
926175    33960000211620    2754  3000275400026402       56152.044623
1745826   33960000211620    2759  3000275900028136       56156.014732
1922566   33960000211620    2760  3000276000029792       56156.015266
2959474   33960000211620    2765  3000276500036874       56167.994914
...                  ...     ...               ...                ...
82226182  33960000211620    2711  3000271100035211       56075.305061
82456417  33960000211620    2712  3000271200042427       56075.305558
83636862  33960000211620    2717  3000271700036927       56078.297314
83884497  33960000211620    2718  3000271800042962       56078.297828
85016389  33960000211620    2723  3000272300029811       56099.295536

          pwp_stack_src_mag3  pwp_stack_src_mag_err3
752483                14.230                   0.028
```

(continues on next page)

(continued from previous page)

```
926175                14.198                    0.030
1745826               14.282                    0.030
1922566               14.342                    0.032
2959474               14.304                    0.029
...                      ...                      ...
82226182              14.265                    0.031
82456417              14.242                    0.027
83636862              14.206                    0.029
83884497              14.205                    0.026
85016389              14.295                    0.031

[131 rows x 6 columns]
```

We can phase the light curve now

For make our code simple we can use to folde the light curve the PyAstronomy and numpy library

[8]:
```
from PyAstronomy.pyasl import foldAt
import numpy as np
```

[9]:
```
%matplotlib inline
import matplotlib.pyplot as plt
```

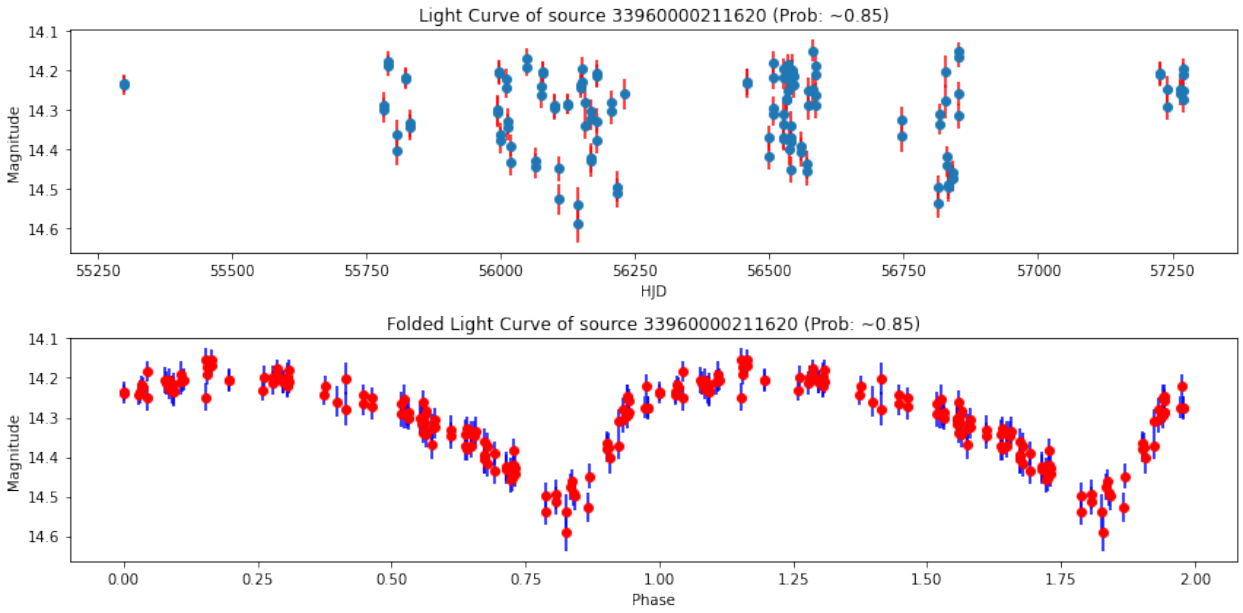
now we can plot the folded and unfolded lightcurves

[10]:
```
lc = lc.sort_values("pwp_stack_src_hjd")

time, mag, err = (
        lc.pwp_stack_src_hjd.values,
        lc.pwp_stack_src_mag3.values,
        lc.pwp_stack_src_mag_err3.values)

t0 = time[0]

phases = foldAt(time, feats.PeriodLS.values, T0=t0)
sort = np.argsort(phases)
phases, pmag, perr = phases[sort], mag[sort], err[sort]

phases = np.hstack((phases, phases + 1))
pmag = np.hstack((pmag, pmag))
perr = np.hstack((perr, perr))


fig, axes = plt.subplots(2, 1, figsize=(12, 6))

ax = axes[0]
ax.errorbar(time, mag, err, ls="", marker="o", ecolor="red")
ax.set_title(f"Light Curve of source {rr.id} (Prob: ~{rr.prob:.2f})")
ax.set_ylabel("Magnitude")
ax.set_xlabel("HJD")
ax.invert_yaxis()

ax = axes[1]
ax.errorbar(phases, pmag, perr, ls="", marker="o", ecolor="blue", color="red")
ax.set_title(f"Folded Light Curve of source {rr.id} (Prob: ~{rr.prob:.2f})")
ax.set_ylabel("Magnitude")
ax.set_xlabel("Phase")
```

(continues on next page)

(continued from previous page)

```
ax.invert_yaxis()

fig.tight_layout()
```



[11]:
```
import datetime as dt
dt.datetime.now()
```

[11]:
```
datetime.datetime(2022, 6, 13, 0, 25, 59, 228809)
```

7.2 API

7.2.1 carpyncho module

Python client for Carpyncho VVV dataset collection.

This code access as a Pandas DataFrame all the data of the web version of Carpyncho https://carpyncho.github.io/.

class carpyncho.**Carpyncho**(*cache_path: str = PosixPath('/home/docs/carpyncho_py_data/_cache_')*, *cache_expire: float = None*, *parquet_engine: str = 'auto'*, *index_url: str = 'https://raw.githubusercontent.com/carpyncho/carpyncho-py/master/data/index.json'*)

Bases: `object`

Client to access the *Carpyncho VVV dataset collection*.

This code access as a Pandas Dataframe all the data of the web version of Carpyncho. https://carpyncho.github.io/.

Parameters

- **cache** (`diskcache.Cache`, `diskcache.Fanout`,) – or `None` (default: `None`) Any instance of `diskcache.Cache`, `diskcache.Fanout` or `None` (Default). If it's `None` a `diskcache.Cache` istance is created with the parameter `directory =`

carpyncho.DEFAULT_CACHE_DIR. More information: http://www.grantjenks.com/docs/diskcache

- **cache_expire** (float or None (default=''None'')) – Seconds until item expires (default None, no expiry) More information: http://www.grantjenks.com/docs/diskcache
- **parquet_engine** (str (default="auto")) – Default Parquet library to use. Remotely carpyncho stores all the data as compresses parquet files; When the download happend a this must be parsed. If 'auto', then the option io.parquet.engine is used. The default io.parquet.engine behavior is to try 'pyarrow', falling back to 'fastparquet' if 'pyarrow' is unavailable.

cache
Return the internal cache of the client the internal cache.

cache_expire = None
Default timeout of the catalog-cache. Try to always set to None (default), the catalogs are big and mostly never change.

cache_path = None
Location of the catalog cache

catalog_info(*tile*, *catalog*)
Retrieve the information about a given catalog.

Parameters

- **tile** (*str*) – The name of the tile.
- **catalog** – The name of the catalog.

Returns The entire information of the given catalog file. This include url, md5 checksum, size in bytes, number of total records, etc.

Return type dict

Raises ValueError: – If the tile or the catalog is not found.

get_catalog(*tile*, *catalog*, *force=False*)
Retrieve a catalog from the carpyncho dataset.

Parameters

- **tile** (*str*) – The name of the tile.
- **catalog** – The name of the catalog.
- **force** (*bool (default=False)*) – If its True, the cached version of the catalog is ignored and redownloaded. Try to always set force to False.

Returns The columns of the DataFrame changes between the different catalog.

Return type pandas.DataFrame

Raises

- ValueError: – If the tile or the catalog is not found.
- IOError: – If the checksum not match.

has_catalog(*tile*, *catalog*)
Check if a given catalog and tile exists.

Parameters

- **tile** (*str*) – The name of the tile.

- **catalog** – The name of the catalog.

Returns True if the convination tile+catalog exists.

Return type bool

index_
Structure of the Carpyncho dataset information as a Python-dict.

index_url = None
Location of the carpyncho index (usefull for development)

list_catalogs(*tile*)
Retrieve the available catalogs for a given tile.

Parameters **tile** (*str*) – The name of the tile to retrieve the catalogs.

Returns The names of available catalogs in the given tile.

Return type tuple of str

Raises ValueError: – If the tile is not found.

list_tiles()
Retrieve available tiles with catalogs as a tuple of str.

parquet_engine = None
Default Parquet library to use.

retrieve_index(*reset*)
Access the remote index of the Carpyncho-Dataset.

The index is stored internally for 1 hr.

Parameters **reset** (*bool*) – If its True the entire cache is ignored and a new index is donwloaded and cached.

Returns

Return type dict with the index structure.

carpyncho.**CARPYNCHOPY_DATA_PATH = PosixPath('/home/docs/carpyncho_py_data')**
Where carpyncho gonna store the entire data.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

carpyncho, 34

Index

C

cache (*carpyncho.Carpyncho attribute*), 35
cache_expire (*carpyncho.Carpyncho attribute*), 35
cache_path (*carpyncho.Carpyncho attribute*), 35
Carpyncho (*class in carpyncho*), 34
carpyncho (*module*), 34
CARPYNCHOPY_DATA_PATH (*in module carpyncho*), 36
catalog_info() (*carpyncho.Carpyncho method*), 35

G

get_catalog() (*carpyncho.Carpyncho method*), 35

H

has_catalog() (*carpyncho.Carpyncho method*), 35

I

index_ (*carpyncho.Carpyncho attribute*), 36
index_url (*carpyncho.Carpyncho attribute*), 36

L

list_catalogs() (*carpyncho.Carpyncho method*), 36
list_tiles() (*carpyncho.Carpyncho method*), 36

P

parquet_engine (*carpyncho.Carpyncho attribute*), 36

R

retrieve_index() (*carpyncho.Carpyncho method*), 36