pygcvs Documentation

Release 1.1

Author

Nov 01, 2017

Contents

A Python library for reading variable star data from the General Catalog of Variable Stars.

CHAPTER 1

Installation

Use `pip` to install latest release available at PyPI:

```
pip install pygcvs
```

CHAPTER 2

Usage

Download the `iii.dat` file from GCVS and point the `read_gcvs` function at its location. The function returns a generator which yields a single star data dictionary at a time. See below:

```
>>> import pygcvs
>>> for star in pygcvs.read_gcvs('iii.dat'):
...     print(star['name'])
R AND
S AND
#...
V0515 VUL
V0516 VUL
```

CHAPTER 3

Resources

- Documentation
- Issue tracker
- CI server

CHAPTER 4

Author

- Zbigniew Siciarz (zbigniew at siciarz dot net)

CHAPTER 5

License

pygcvs is free software, licensed under the MIT/X11 License. A copy of the license is provided with the source code in the LICENSE file.

5.1 Documentation

5.1.1 Usage

Basic usage

The function *read_gcvs()* is the main entry point to the library. When called, it will return a generator yielding a single dictionary at a time. The dictionary fields correspond to variable star attributes such as name, location, min/max brightness, variability type etc.

The following fields are available:

constellation A three-letter abbreviation of constellation in which the star can be found, normalized to upper case. For example *OPH*, *CRB* etc.

name Variable star designation, such as *RR LYR*, *V838 MON* etc. The name is always normalized to upper case.

ra Right ascension of the variable star in J2000.0 epoch, eg. `'20:50:12.7'`.

dec Declination of the variable star in J2000.0 epoch, eg. `'+03:39:08'`.

Note: The coordinates are returned as strings compatible with the popular PyEphem astronomy package. For example, `ra` field can be passed directly to `pyephem.hours` function.

---

variable_type Type of variability, for example *M* for Mira-like pulsating red giants. This value is literal as it appears in the GCVS, meaning that uncertain types contain *:* (a colon), etc.

max_magnitude Maximum brightness of a star as a floating point number. Can be *None* if unspecified in GCVS.

min_magnitude Brightness of the variable star at minimum, as a floating point number. *None* if unspecified in GCVS.

Note: At the moment some special characters are stripped from both maximum and minimum magnitude. For example, if GCVS denotes the minimum brightness as <16.0 (fainter than 16.0), the value of `min_magnitude` will be 16.0 and the *fainter than* information will be lost. However, if the field in GCVS represents an amplitude instead of minimum magnitude, `min_magnitude` field will contain a correctly calculated absolute value.

---

epoch Julian Day epoch of the maximum brightness as a float. *None* if the star is not periodic or there is no epoch data in GCVS.

period Variability period in days, or *None* if unspecified or not periodic.

PyEphem compatibility

If you have PyEphem installed, you can use the *dict_to_body()* function to convert star data to a *Body* instance more suitable for various astronomical computations. See the example below, which checks if **TX Del** (a Type II cepheid variable) is currently visible in Warsaw.‘

```
"""
Check if TX Del is visible in Warsaw at the moment.
"""

import sys
import ephem

from pygcvs import read_gcvs, dict_to_body


if __name__ == '__main__':
    try:
        gcvs_file = sys.argv[1]
    except IndexError:
        print('Usage: python check_visibility.py <path to iii.dat>')
    else:
        city = ephem.city('Warsaw')
        stars = {star['name']: star for star in read_gcvs(gcvs_file)}
        body = dict_to_body(stars['TX DEL'])
        body.compute(city)
        if body.alt > 0:
            print('TX Del is visible in Warsaw right now, go observe!')
        else:
            print('TX Del is not visible in Warsaw right now. Try later.')
```

Using GcvsParser directly

Most of the time, the *read_gcvs()* function works just fine. However, it’s limited (on purpose!) to read GCVS data from a file on disk. If you have the data in some other form, you can pass it directly to the *GcvsParser*, provided the data source supports the iterator protocol.

For example, when using requests, the response object has a `iter_lines` method which, well, iterates over the lines of the response.

```
"""
An example of directly using the parser to read downloaded GCVS data.
"""

import requests

from pygcvs.parser import GcvsParser


if __name__ == '__main__':
    url = 'http://www.sai.msu.su/gcvs/gcvs/iii/iii.dat'
    response = requests.get(url, stream=True)
    parser = GcvsParser(response.iter_lines(decode_unicode=True))
    for star in parser:
        print(star)
```

Visualisations with matplotlib

You can use matplotlib to visualise various aspects of GCVS variable star data.

Maximum vs minimum magnitude

```
"""
Visualisation of maximum/minimum magnitude for GCVS stars.
"""

import sys

import matplotlib.pyplot as plot

from pygcvs import read_gcvs


if __name__ == '__main__':
    try:
        gcvs_file = sys.argv[1]
    except IndexError:
        print('Usage: python plot_magnitudes.py <path to iii.dat>')
    else:
        min_magnitudes = []
        max_magnitudes = []
        for star in read_gcvs(gcvs_file):
            if star['min_magnitude'] and star['max_magnitude']:
                min_magnitudes.append(star['min_magnitude'])
                max_magnitudes.append(star['max_magnitude'])
        plot.title('GCVS variable star magnitudes')
        plot.plot(min_magnitudes, max_magnitudes, 'ro')
        plot.xlabel('Min magnitude')
        plot.ylabel('Max magnitude')
        # invert axes because brightest stars have lowest magnitude value
        plot.gca().invert_xaxis()
        plot.gca().invert_yaxis()
        plot.savefig('magnitudes.png')
```

Brightness amplitude vs period

```
"""
Visualisation of brightness amplitude vs variability period.
"""

import sys

import matplotlib.pyplot as plot

from pygcvs import read_gcvs


if __name__ == '__main__':
    try:
        gcvs_file = sys.argv[1]
    except IndexError:
        print('Usage: python plot_amplitude_vs_period.py <path to iii.dat>')
    else:
        periods = []
        amplitudes = []
        for star in read_gcvs(gcvs_file):
            if star['period'] and star['min_magnitude'] and star['max_magnitude']:
                periods.append(star['period'])
                amplitudes.append(star['min_magnitude'] - star['max_magnitude'])
        plot.title('GCVS variable stars amplitudes')
        plot.semilogx(periods, amplitudes, 'ro')
        plot.xlabel('Period [days]')
        plot.ylabel('Brightness amplitude [mag]')
        plot.savefig('amplitude_vs_period.png')
```

5.1.2 API reference

pygcvs Package

class pygcvs.__init__.**GcvsParser**(*fp*)

A parser for GCVS data format.

Example usage:

```
>>> with open('iii.dat', 'rb') as fp:
...     parser = GcvsParser(fp)
...     for star in parser:
...         print(star['name'])
R AND
S AND
#...
V0515 VUL
V0516 VUL
```

Creates the parser and feeds it a file-like object.

Parameters **fp** – a file-like object or a generator yielding strings

parse_coordinates(*coords_str*)

Returns a pair of PyEphem-compatible coordinate strings (Ra, Dec).

If the star has no coordinates in GCVS (there are such cases), a pair of None values is returned.

parse_epoch(*epoch_str*)
Converts epoch field to a float value (adding 24... prefix), or `None` if there is no epoch in GCVS record.

parse_magnitude(*magnitude_str*)
Converts magnitude field to a float value, or `None` if GCVS does not list the magnitude.

Returns a tuple (magnitude, symbol), where symbol can be either an empty string or a single character - one of '<', '>', '('.

parse_name(*name_str*)
Normalizes variable star designation (name).

parse_period(*period_str*)
Converts period field to a float value or `None` if there is no period in GCVS record.

row_to_dict(*row*)
Converts a raw GCVS record to a dictionary of star data.

pygcvs.__init__.**dict_to_body**(*star_dict*)
Converts a dictionary of variable star data to a *Body* instance.

Requires PyEphem to be installed.

pygcvs.__init__.**read_gcvs**(*filename*)
Reads variable star data in GCVS format.

Parameters **filename** – path to GCVS data file (usually `iii.dat`)

pygcvs.parser Module

class pygcvs.parser.**GcvsParser**(*fp*)
A parser for GCVS data format.

Example usage:

```
>>> with open('iii.dat', 'rb') as fp:
...     parser = GcvsParser(fp)
...     for star in parser:
...         print(star['name'])
R AND
S AND
#...
V0515 VUL
V0516 VUL
```

Creates the parser and feeds it a file-like object.

Parameters **fp** – a file-like object or a generator yielding strings

parse_constellation(*constellation_str*)

parse_coordinates(*coords_str*)
Returns a pair of PyEphem-compatible coordinate strings (Ra, Dec).

If the star has no coordinates in GCVS (there are such cases), a pair of None values is returned.

parse_epoch(*epoch_str*)
Converts epoch field to a float value (adding 24... prefix), or `None` if there is no epoch in GCVS record.

parse_magnitude(*magnitude_str*)
Converts magnitude field to a float value, or `None` if GCVS does not list the magnitude.

Returns a tuple (magnitude, symbol), where symbol can be either an empty string or a single character - one of '<', '>', '('.

parse_name(*name_str*)
Normalizes variable star designation (name).

parse_period(*period_str*)
Converts period field to a float value or `None` if there is no period in GCVS record.

row_to_dict(*row*)
Converts a raw GCVS record to a dictionary of star data.

5.1.3 Development

Contributing

Looking to improve pygcvs? Here's how you can help.

Report issues

If you think you found a **bug** in pygcvs or have a **feature request**, feel free to file an issue. We rely on GitHub for issue tracking. Please, search through existing issues before you report a new one; perhaps your problem was already discussed or fixed.

When submitting an issue, please include the following:

- problem description
- steps to reproduce (a smallest possible code example that reproduces the issue would be most welcome!)
- expected outcome
- actual outcome
- platform information: your operating system, Python version, etc.
- any other relevant information

Contribute code

Contributions to pygcvs source code are accepted as **pull requests** on GitHub. Fork the project, work on it in your repository and when you think your patch is ready, send us a pull request.

License

By contributing your code, you agree to license your contribution under the terms of MIT license (see `LICENSE` file for details).

5.1.4 Changelog

Unreleased

- none yet

1.1.0

- Python 3.5 and 3.6 support

1.0.0

- first **Python 3-only** release!
- moved helper functions to pygcvs.helpers module
- fixed parsing uncertain epochs and periods

0.1.4

- Python 3.4 compatibility
- more specific Python version classifiers in setup.py

0.1.3

- fixed reading empty coordinates

0.1.2

- added wheel distribution

0.1.1

- added constellation abbreviation to star data dictionary

0.1.0

- initial release

5.2 Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index