Easy asteroid phase curve fitting for the Python ecosystem: Pyedra

Milagros R. Colazo[a,b], Juan B. Cabral[c,a], Martín Chalela[a,1], Bruno O. Sánchez[d]

[a] *Instituto de Astronomía Teórica y Experimental - Observatorio Astronómico de Córdoba (IATE, UNC–CONICET), Córdoba, Argentina.*
[b] *Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba (FaMAF–UNC) Bvd. Medina Allende s/n, Ciudad Universitaria, X5000HUA, Córdoba, Argentina*
[c] *Centro Internacional Franco Argentino de Ciencias de la Información y de Sistemas (CIFASIS, CONICET–UNR), Ocampo y Esmeralda, S2000EZP, Rosario, Argentina.*
[d] *Department of Physics, Duke University, 120 Science Drive, Durham, NC, 27708, USA*

Abstract

A trending astronomical phenomenon to study is the the variation in brightness of asteroids, caused by its rotation on its own axis, non-spherical shapes, changes of albedo along its surface and its position relative to the sun. The latter behavior can be visualized on a "Phase Curve" (phase angle vs. reduced magnitude). To enable the comparison between several models proposed for this curve we present a Python package called Pyedra. Pyedra implements three phase-curve-models, and also providing capabilities for visualization as well as integration with external datasets. The package is fully documented and tested following a strict quality-assurance workflow, whit a user-friendly programmatic interface. In future versions, we will include more models, and additional estimation of quantities derived from parameters like diameter, and types of albedo; as well as enabling correlation of information of physical and orbital parameters.

Keywords: minor planets, asteroids: general ; planets and satellites: fundamental parameters ; Python Package

1. Introduction

The brightness variation of asteroids is a fascinating astronomical phenomenon to study. One cause of the object's varying magnitude is its rotation about its own axis. This is because asteroids have non-spherical shapes and albedo differences along their surface. On the other hand, the brightness of an asteroid will also vary just by moving in its orbit around the Sun. When the object is close to opposition, i.e. at angles close to 0°, sunlight hitting the asteroid and light reflected from the object's surface will come from the same direction, causing the object to have a maximum in apparent brightness. As it moves in its orbit and its phase angle increases, the Sun's light will begin to cast shadows on the asteroid's surface causing a decrease in its brightness. In summary, the magnitude of an asteroid drops as it approaches the opposition and as it moves away the magnitude will begin to grow again. This behavior can be visualized on a phase angle (α) vs. reduced magnitude V diagram, known as "Phase Curve". Although several models have been proposed to describe this curve, there is no comprehensive tool available that provides with the necessary fitting procedures and enables a reliable comparison between these models.

In 1989, Bowell et al. proposed the G model (also known as H, G model), a semi-empirical model derived from the basic principles of radiative transfer theory with some assumptions (Waszczak et al. 2015). Shevchenko (1996) proposed an empirical tri-parametric phase function model valid for phase angles in the range of $0° − 40°$. There is a third model for the phase function proposed by Muinonen et al. in 2010. It is a model similar to Bowell's but replaces the G parameter with two parameters G_1 and G_2, making it also a tri-parametric model. According to Muinonen et al., the G model is a good approximation in the region of $10°\sim60°$, while the G_1 and G_2 model works well also for angles close to the opposition ($\sim0°$).

Many large sky surveys are currently in operation. Thousands of asteroids are observed by these telescopes, providing a unique opportunity to study them. Some of these large sky surveys are Gaia (Gaia Collaboration et al. 2018), TESS (Ricker et al. 2015), in the near future the Vera Rubin's Legacy Survey of Space and Time (LSST, Schwamb et al. 2019). This poses the opportunity to characterize the phase curve of a large numbers of asteroids. We must be prepared to take full advantage of this information. One of the main analysis with these datasets would be the calculation of the absolute magnitude H of hundreds or thousands of these objects, enabling also the estimation of their diameters. The parameters G, G_1, G_2, b can provide a good estimate of the albedo of the asteroids and, even more, can help in the taxonomic classification (Shevchenko 1996; Belskaya & Shevchenko 2000; Carbognani et al. 2019).

In this context of "big data for asteroids" we developed Pyedra . Pyedra enables the analysis of large amounts of data, it provides the parameters of the selected phase function model that best fits the observations. Thus, it can quickly create parameter catalogs for large databases as well as providing the possibility of working with non-survey data, i.e. personal observations.

This paper is organized as follows: in Section 2 we provide a brief description of the algorithm. In Section 3 we introduce technical details about the Pyedra package. In Section 4 we present the conclusions and future perspectives.

2. The Algorithm

In this section, we present the three phase function models implemented in Pyedra and for each one of them, we provide details on the method used for parameter estimation. In general we adopt the procedure proposed by Muinonen et al. (2010), hereafter M10.

2.1. H, G model

The H, G phase function model for asteroids can be described analytically through the following equation (Muinonen et al. 2010):

$$V(\alpha) = H - 2.5 \log_{10}[(1 - G)\Phi_1(\alpha) + G\Phi_2(\alpha)], \quad (1)$$

where H and G are the two free parameters of the model, α is the phase angle, $V(\alpha)$ is the reduced V magnitude (brightness on Johnson's filter V normalized at 1 AU from the Sun and the observer), $\Phi_1(\alpha)$ and $\Phi_2(\alpha)$ are two basis function normalized at unity for $\alpha = 0°$. The base functions can be accurately approximated by:

$$\Phi_1(\alpha) = \exp\left(-3.33 \tan^{0.63}\frac{1}{2}\alpha\right),$$
$$\Phi_2(\alpha) = \exp\left(-1.87 \tan^{1.22}\frac{1}{2}\alpha\right). \quad (2)$$

To obtain the value of H and G, M10 proposes to write to the reduced magnitude as:

$$10^{-0.4V(\alpha)} = a_1\Phi_1(\alpha) + a_2\Phi_2(\alpha), \quad (3)$$

then we can write the absolute magnitude H and the coefficient G as:

$$H = -2.5 \log_{10}(a_1 + a_2),$$
$$G = \frac{a_2}{a_1 + a_2}. \quad (4)$$

The coefficients a_1 and a_2 are estimated from the observations using the standard method of least squares.

2.2. H, G₁, G₂ model

This three parameter magnitude phase function can be described as in M10:

$$V(\alpha) = H - 2.5 \log_{10}[G_1\Phi_1(\alpha) + G_2\Phi_2(\alpha) + (1 - G_1 - G_2)\Phi_3(\alpha)], \quad (5)$$

where $\Phi_1(0°)=\Phi_2(0°)=\Phi_3(0°)=1$. H, G_1 and G_2 are the parameters of the model, α is the phase angle, $V(\alpha)$ is the reduced magnitude and Φ_1, Φ_2, Φ_3 are basis functions.

These basis are defined piecewise using linear terms as well as cubic splines (Penttilä et al. 2016) along the orbit.

In this case we write the reduced magnitude as:

$$10^{-0.4V(\alpha)} = a_1\Phi_1(\alpha) + a_2\Phi_2(\alpha) + a_3\Phi_3(\alpha). \quad (6)$$

For this calculation we use the tabulated values for the base functions presented in Penttilä et al. (2016). The model free parameters can be obtained from:

$$H = -2.5 \log_{10}(a_1 + a_2 + a_3),$$
$$G_1 = \frac{a_1}{a_1 + a_2 + a_3},$$
$$G_2 = \frac{a_2}{a_1 + a_2 + a_3}. \quad (7)$$

The coefficients a_1, a_2 and a_3 are estimated from observations using the method of least squares.

2.3. Shevchenko model

This model is described in the following equation (Shevchenko 1996):

$$V(1, \alpha) = V(1, 0) - \frac{a}{1 + \alpha} + b \cdot \alpha, \quad (8)$$

where a characterizes the amplitude of the so-called "opposition effect", b is a parameter describing the linear term of the phase-magnitude relationship, α is the phase angle and $V(1, 0)$ is the absolute magnitude.

Although M10 does not present this model in its work, we have extended the use of least squares for this case as well given that Shevchenko's formula is already written in the form of a linear equation.

3. Technical details about the Pyedra package

3.1. User functionalities and application example

The Pyedra package consists of 3 main functions to perform the fitting of observations. Each of these functions corresponds to one of the models mentioned in Section 2. These functions are:

- `HG_fit()`: fits the 2.1 model to the observations.

- `HG1G2_fit()`: fists the 2.2 model to the observations.

- `Shev_fit()`: fits the 2.3 model to the observations.

All these functions return an object that we call `PyedraFitDataFrame`, containing the parameters obtained after the fit with a format that is analogous to a pandas Dataframe. The `plot()` method of this object returns:

- a graph of our observations in the plane (α, V) together with the fitted model.

- any of the graphics that pandas allows to create.

Pyedra also offers the possibility to add Gaia observations to the user's sample. This is done by using:

1. `.load_gaia()` to read the files containing Gaia observations.

2. `.merge_obs()` to merge the user and Gaia tables.

3. One can apply any of the above functions to this new dataframe.

This is a very interesting feature because, in general, observations from the ground correspond to small phase angles. In contrast, Gaia can only observe for phase angles $\alpha > 10°$ (Gaia Collaboration et al. 2018). Both sets of data are complementary, thus achieving a more complete coverage of the phase angle space. This also leads to a better determination of the phase function parameters.

As a simple usage application we show how to calculate the parameters of the H, G model and how to plot the observations with the fit obtained using the dataset of Carbognani et al. (2019). The respective plots are shown in Figs. 1 & 2. This dataset is provided with Pyedra for the user to test the functionalities.

```
>>> import pyedra
>>> import pandas as pd
>>> import matplotlib.pyplot as plt

# load the data
>>> df = pyedra.datasets.load_carbognani2019()

# fit the data
>>> HG = pyedra.HG_fit(df)
    id         H     error_H         G     error_G
         R
0   85   7.492423   0.070257   0.043400   0.035114
     0.991422
1  208   9.153433   0.217270   0.219822   0.097057
     0.899388
2  236   8.059719   0.202373   0.104392   0.094382
     0.914150
3  306   8.816185   0.122374   0.306459   0.048506
     0.970628
4  313   8.860208   0.098102   0.170928   0.044624
     0.982924
5  338   8.465495   0.087252  −0.121937   0.048183
     0.992949
6  522   8.992164   0.063690   0.120200   0.028878
     0.991757
PyedraFitDataFrame − 7 rows x 6 columns

# take the mean value of H
>>> HG.H.mean()
>>> 8.54851801238607

# plot the data and the fit
>>> HG.plot(df=df, ax=None)
>>> plt.show()
<AxesSubplot: title={'center':'Phase curves'},
xlabel='Phase angle', ylabel='V'>

# scatter plot of G vs H
>>> HG.plot(x='G', y='H', kind='scatter')
>>> plt.show()
<AxesSubplot: xlabel='G', ylabel='H'>
```

3.2. Quality assurance

Software quality assurance refers to the set of standards and procedures that must be used in order to verify that the software meets certain subjective quality criteria. The most common procedures to carry out this task are *unit-testing* and *code-coverage*.
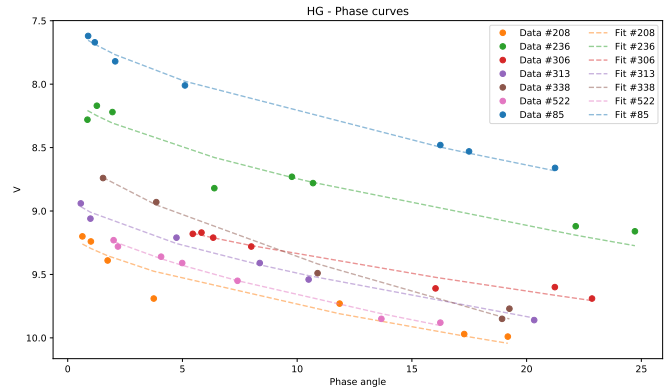

Figure 1: Example of the figure obtained for the model H, G. The points correspond to the observations and the dotted lines to the best fit. Points of the same color correspond to the same object.
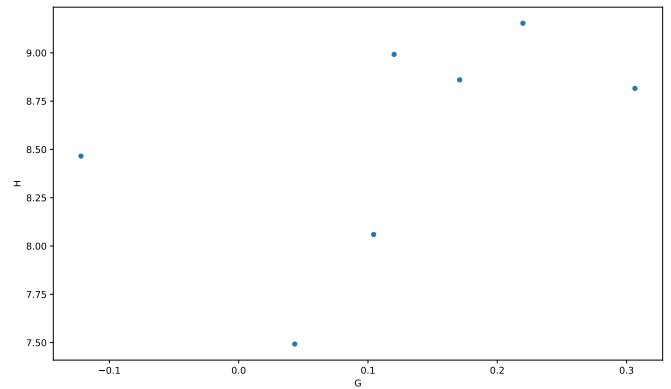

Figure 2: Example of the 'scatter' graphic of pandas dataframe, also available for PyedraDataFrame.

The purpose of *unit-testing* is to check that each of the individual components of the software works as expected (Jazayeri 2007). That is, we isolate a function from our code and verify that it works correctly. On the other hand, *code-coverage* is a measure of how much of our software has been tested (Miller & Maloney 1963). In this way, we can identify parts of the code that we have not verified. In the Pyedra package we provide five suites of unit-tests that evaluate different sections of the code, reaching 99% of code-coverage. The testing suites are tested for Python versions 3.7, 3.8 and 3.9. We are also interested in the maintainability of Pyedra, therefore we have adopted PEP 8 – Style Guide for Python Code (Van Rossum et al. 2001) in such a way that our project meets current code standard and readability. For this purpose, we use the *flake8*[1] tool hat automatically detects any case where we are not respecting the style imposed by PEP 8 as well as programming errors, such as: "library imported but unused".

Finally, the entire source code is MIT-licensed and available in a public repository[2]. All changes and new versions of the package committed to this repository are automatically tested

---

[1] https://flake8.pycqa.org/en/latest/
[2] https://github.com/milicolazo/Pyedra

with continuous-integration services [3, 4]. Documentation is automatically generated from Pyedra docstrings and made public in the read-the-docs service[5].

Finally, Pyedra is available for installation on the PythonPackage-Index (PyPI)[6]; and is currently going through registration process to appear in the Astrophysics Source Code Library (ASCL.net, Grosbol & Tody 2010)

3.3. Integration with the Python scientific–stack

Python has become an important programming language within the astronomical community (Stansby et al. 2020). This is mainly because it is a simple to use, free and versatile language for manipulating and visualizing data (Faes 2012).

Pyedra is built on top of the Python scientific stack: *Pandas* (McKinney et al. 2010) since the main object on which Pyedra operates is a dataframe; *Scipy* (Virtanen et al. 2020) for function interpolation and fit of least squares optimization; *Numpy* (Walt et al. 2011) to manipulate arrays; *Matplotlib* (Hunter 2007) for the data visualization; and attrs[7] to facilitate the implementation of classes.

3.3.1. Short comparison with other similar packages

Pyedra's main objective is to calculate the parameters of different phase function models for large and small volumes of data. The `sbpy`[8] (Mommert et al. 2019) package offers the possibility to model phase curves. In this subsection, we will make a brief contrast between both projects.

Regarding the available models, `sbpy` and Pyedra share the HG and HG1G2 model. In the case of `sbpy`, the models HG12 (Revised H, G12 model by Penttilä et al.) and a linear model are also available. Although these models were not considered in Pyedra (but will be implemented in the next release), we have included Shevchenko's model which is not present in `sbpy`.

On the other hand, `sbpy` does not provide the functionality to estimate the best fit model parameters (as Pyedra does) but returns other quantities derived from these parameters. In addition, Pyedra has an error estimate for each calculated value, something that is not present in `sbpy`.

Finally, Pyedra's main strength against `sbpy` is its simplicity of use. With sbpy we have not found a quick way to get phase function parameter catalogs for databases with large numbers of entries. With Pyedra , the user can accomplish this task by just writing one line of code. The same is true for graphic capabilities: since plotting phase functions is one of Pyedra 's features, one single method call allows to obtain a visualization of the phase function. It is also worth noticing that with Pyedra not only phase curve plots can be easily obtained, all pandas visualization tools are also available enabling a more comprehensive analysis of the resulting catalog. Moreover, as it is based on pandas' dataframe manipulation, the output catalog is simple to visualize, modify and to carry out different calculations from it.

---

[3]`https://travis-ci.com/milicolazo/Pyedra`
[4]`https://github.com/milicolazo/Pyedra/actions`
[5]`https://pyedra.readthedocs.io/en/latest/?badge=latest`
[6]`https://pypi.org/project/Pyedra/`
[7]`https://www.attrs.org`
[8]`https://sbpy.org/`

4. Conclusions

In this paper, we present Pyedra, a python implementation for asteroid phase curve fitting. This package allows the user to fit three different models of phase functions to observations of asteroid phase angle and photometry.

Pyedra is suitable for analysis of private datasets, of one or more asteroids, as well as large volumes of information from any public survey data release such as TESS, Gaia, K2, among others. SiConsequently Pyedra is a tool that will enable the creation of phase curve model parameter catalogs for hundreds of thousands of asteroids.

Pyedra also offers the possibility of producing numerous visualization plots. Not only it can produce a graph of the phase functions but it makes available all the graphs natively offered for pandas dataframes. In this way, we provide the possibility of a complete analysis of the results obtained.

As we have already mentioned, we are living in an era of big surveys. We must be able to have tools capable of processing the vast amount of data that these surveys constantly provide to the scientific community.

4.1. Future Work

Pyedra is still in development process, so there are still topics to be improved.

The first thing to consider would be to have more phase function models added to those already offered by the package. In addition, it would be convenient to be able to estimate certain quantities derived from the parameters obtained, such as the diameter, the integral function, the different types of albedo, etc. It would be interesting to have a tool that (in the case of a database containing several asteroids) allows combining information on physical parameters with orbital parameters. For example, the possibility of studying which *G* values the asteroids have for different semi-axes *a*.

Finally, we intend to add more large survey data for the user to combine with their observations, such as TESS, SDSS, etc.

5. Acknowledgments

References

Belskaya, I. N. & Shevchenko, V. G. 2000, Icarus, 147, 94

Bowell, E., Hapke, B., Domingue, D., et al. 1989, in Asteroids II, ed. R. P. Binzel, T. Gehrels, & M. S. Matthews, 524–556

Carbognani, A., Cellino, A., & Caminiti, S. 2019, Plan. Space Sci., 169, 15

Faes, D. 2012, Journal of Colloid and Interface Science, 3, E1

Gaia Collaboration, Spoto, F., Tanga, P., et al. 2018, A&A, 616, A13

Grosbol, P. & Tody, D. 2010, arXiv preprint arXiv:1004.4430

Hunter, J. D. 2007, Computing in science & engineering, 9, 90

Jazayeri, M. 2007, in Future of Software Engineering (FOSE'07), IEEE, 199–213

McKinney, W. et al. 2010, in Proceedings of the 9th Python in Science Conference, Vol. 445, Austin, TX, 51–56

Miller, J. C. & Maloney, C. J. 1963, Communications of the ACM, 6, 58

Mommert, M., Kelley, M. S., Val-Borro, M., et al. 2019, Journal of open source software

Muinonen, K., Belskaya, I. N., Cellino, A., et al. 2010, Icarus, 209, 542

Penttilä, A., Shevchenko, V. G., Wilkman, O., & Muinonen, K. 2016, Plan. Space Sci., 123, 117

Ricker, G. R., Winn, J. N., Vanderspek, R., et al. 2015, Journal of Astronomical Telescopes, Instruments, and Systems, 1, 014003

Schwamb, M. E., Hsieh, H., Bannister, M. T., et al. 2019, Research Notes of the AAS, 3, 51

Shevchenko, V. G. 1996, in Lunar and Planetary Science Conference, Vol. 27, Lunar and Planetary Science Conference, 1193

Stansby, D., Yeates, A., & Badman, S. 2020, The Journal of Open Source Software, 5, 2732

Van Rossum, G., Warsaw, B., & Coghlan, N. 2001, Python. org, 1565

Virtanen, P., Gommers, R., Oliphant, T. E., et al. 2020, Nature Methods, 17, 261

Walt, S. v. d., Colbert, S. C., & Varoquaux, G. 2011, Computing in science & engineering, 13, 22

Waszczak, A., Chang, C.-K., Ofek, E. O., et al. 2015, AJ, 150, 75