The Journal of Open Source Software

`lhorizon`: geometry and targeting via JPL *Horizons*

Michael St. Clair[1] **and Matthew Siegler**[2]

1 Chief Technical Officer, Million Concepts **2** Research Scientist, Planetary Science Institute

DOI: 10.21105/joss.03495

Software

- Review
- Repository
- Archive

Editor: Arfon Smith

Reviewers:

- @malmans2
- @steo85it

Submitted: 06 July 2021
Published: 13 September 2021

Summary

`lhorizon` helps you locate natural and artificial bodies and features in the Solar System. It is built around a thick Python wrapper for the Jet Propulsion Laboratory (JPL) Solar System Dynamics Group *Horizons* service (Jon D. Giorgini, 2015). *Horizons* is one of the only providers of ready-to-go, no-assembly-required geometry data for almost every object in the Solar System. Other interfaces to *Horizons* exist (see 'Other Related Work' below for several examples), but `lhorizon` is particularly optimized for stability and time-to-value for large queries and for processing results with reference to arbitrary topocentric coordinate systems.

`lhorizon` offers a flexible, idiomatic, highly performant pseudo-API to *Horizons* that returns data as standard scientific Python objects (*NumPy*, Harris et al. (2020) `ndarrays` and *pandas* DataFrames, The pandas development team (2020)). It provides special handling functions for bulk, chunked, and body-listing queries; it also includes an ancillary module, `lhorizon.targeter`, for finding the footprint of an observer's boresight or field of view in topocentric coordinates on target bodies.

We wrote `lhorizon` in support of a research effort to use Earth-based radio telescopes, including Arecibo and the Very Large Array (VLA), to perform heat flow mapping of the Moon. We needed to coregister these data with existing models, so we were specifically interested in answering questions like: "in lunar latitude and longitude, where is Arecibo pointing?" As Earth-based radio telescopes are not primarily designed to answer questions about nearby bodies, their processing pipelines aren't readily suited for goals like producing lunar maps. Using these instruments in unusual ways also resulted in additional measurement uncertainties (*e.g.*, geometric uncertainty) that we wanted to minimize. *Horizons* was an appealing data source due to its high precision (up to microarcseconds for Moon positions relative to Earth in the relevant time frame, well above the limits of precision introduced by our other constraints) and its ability to deliver tables of positions relative to arbitrary topocentric points, natively referenced to the geodetic datums of their host bodies (*e.g.*, WGS84 for Earth), with robust corrections for light-time, gravitational delays, and aberration.

However, with millions of data points widely dispersed across times and observing locations, we needed a highly performant programmatic interface to achieve our task. We were pleased to discover that the *astroquery* (Ginsburg et al., 2019) project included a module for querying *Horizons* called `jplhorizons`. This module, written primarily by Michael Mommert around 2016, is tightly integrated with *astroquery*. It uses *astroquery*'s session handlers and parsing system, and returns results in *astropy* tables. Unfortunately, we discovered that due to changes in the behavior of the Horizons CGI endpoint, parts of `jplhorizons` that probably worked very well in 2016 no longer worked in 2019. We implemented workarounds, but discovered that the performance of `jplhorizons` was inadequate for our use case. *astroquery*'s parsers and *astropy* tables are powerful, but this power comes at a performance cost. The cost is irrelevant for many applications but quite relevant for use cases with tens to hundreds of thousands of data points per analysis. We wrote an entirely new response parser using only builtins, *NumPy*, and *pandas*, resulting in performance improvements of a factor of 10-100x. (Since then, there have been significant backend improvements in *astropy* tables, and `lhorizon` typically offers

only about 10x speed and 50% memory reduction over the latest version of `jplhorizons`. Benchmark notebooks are available in our GitHub repository.)

We submitted minimal workarounds for the API issues to *astroquery*, but the changes we made in our fork were too extensive to be folded into *astroquery* via a PR – especially because removing *astropy* objects and idioms was one of our major design goals. We named this fork `lhorizon` and have continued developing it as a distinct project.

Statement of Need

JPL's geometry products are essential elements of academic and industrial projects related to planetary science, astronomy, geosciences, and many other fields. They are useful for any application involving artificial satellites (from data analysis to mission design) or the position of Solar System bodies (even "simple" quantities like solar angles at arbitrary Earth locations). They are thus invaluable public resources. Unfortunately, they are not always easy to use.

JPL offers two automated interfaces to its geometry products: the SPICE toolkit (NASA NAIF, 2021), developed by NAIF, NASA's Navigation and Ancillary Information Facility, and *Horizons*. SPICE is very powerful but presents a high barrier to entry. Using SPICE requires not only acquiring and configuring software, but also collecting the appropriate data files, called "kernels." There is no central repository for kernels – NAIF's website comes closest, but crucial kernels are scattered across hundreds of other Planetary Data System (PDS) archives. Loading a consistent kernel pool is challenging and requires scripting in a domain-specific markup language. Learning the SPICE toolkit can be challenging for both specialists and for general users who need quick access to geometry data. SPICE implementations exist in several languages, and excellent wrappers exist (notably the idiomatic Python wrapper *SpiceyPy* (Annex et al., 2020)), but they do not solve the conceptual and data access difficulties of SPICE.

Horizons is, by comparison, user-friendly. While it does not implement all of the utilities of the SPICE toolkit, it offers flexibility SPICE does not and contains state information for many sites and bodies for which no SPICE kernels exist. *Horizons* offers several interface methods: interactive web, telnet, email, and web CGI. Because bulk queries to the CGI endpoint are not easy to compose and parsing its responses is not straightforward, simply building URLs for this interface and
parsing the returned text significantly improves access to Solar System geometry data – `lhorizon` does that and more. Some likely use cases include calculating solar angles on Mars, determining the precise distance from the Solar System barycenter to an artificial satellite, and finding selenodetic coordinates for pixels within the field of view of a terrestrial telescope pointed at the Moon. We include Jupyter Notebooks in our repository that illustrate these uses.

An official REST API to *Horizons* is forthcoming but not yet available, and the details of its capabilities have not been publicly released (J. D. Giorgini, 2020). It is likely that high-level wrappers for this API will be useful, and we plan to update `lhorizon` to fill this role.

Other Related Work

Many wrappers, helpers, and interfaces for *Horizons* have been developed, though most are incomplete, defunct, or encapsulated in other applications. They include:

- *py-NASA-horizons*, a vectors query wrapper; abandoned since 2013 and no longer functional (tpltnt (pseudonymous), 2013).

- Mihok's *HORIZON-JPL*, a REST API; abandoned since 2014 and no longer functional (Mihok, 2014).
- Fejes' *JS-HORIZONS*, a JavaScript library focused on physical rather than geometry data (Fejes, 2020).

More broadly, libraries like `astropy.coordinates` (A. M. Price-Whelan et al., 2018) and *Skyfield* (Rhodes, 2019) that perform calculations based on JPL ephemerides are similar in application to `lhorizon` and should be considered by potential users.

This application space also includes lower-level ephemeris toolkits other than SPICE that may be preferable for some applications. For instance, the CALCEPH (M. Gastineau, 2021) library, developed by the IMCCE of the Observatoire de Paris, offers interfaces to many programming languages and is compatible with a wider variety of ephemeris formats than SPICE.

Acknowledgements

This work was supported by a NASA Solar System Workings grant, #NNX16AQ10G, and a NASA Solar System Observations grant, #NNX17AF12G.

References

A. M. Price-Whelan, and, Sipőcz, B. M., Günther, H. M., Lim, P. L., Crawford, S. M., Conseil, S., Shupe, D. L., Craig, M. W., Dencheva, N., Ginsburg, A., VanderPlas, J. T., Bradley, L. D., Pérez-Suárez, D., Val-Borro, M. de, Aldcroft, T. L., Cruz, K. L., Robitaille, T. P., Tollerud, E. J., Ardelean, C., … and, and. (2018). The astropy project: Building an open-science project and status of the v2.0 core package. *The Astronomical Journal*, *156*(3), 123. https://doi.org/10.3847/1538-3881/aabc4f

Annex, A., Pearson, B., Seignovert, B., Carcich, B., Eichhorn, H., Mapel, J., Forstner, J. von, McAuliffe, J., Rio, J. del, Berry, K., Aye, K.-M., Stefko, M., Val-Borro, M. de, Kulumani, S., & Murakami, S. (2020). SpiceyPy: A pythonic wrapper for the SPICE toolkit. *Journal of Open Source Software*, *5*(46), 2050. https://doi.org/10.21105/joss.02050

Fejes, Z. (2020). *JS-HORIZONS*. GitHub. https://github.com/zachfejes/js-horizons

Ginsburg, A., Sipőcz, B. M., Brasseur, C. E., Cowperthwaite, P. S., Craig, M. W., Deil, C., Guillochon, J., Guzman, G., Liedtke, S., Lim, P. L., Lockhart, K. E., Mommert, M., Morris, B. M., Norman, H., Parikh, M., Persson, M. V., Robitaille, T. P., Segovia, J.-C., Singer, L. P., … and, J. W. (2019). Astroquery: An astronomical web-querying package in python. *The Astronomical Journal*, *157*(3), 98. https://doi.org/10.3847/1538-3881/aafc33

Giorgini, Jon D. (2015). Status of the JPL Horizons Ephemeris System. *IAU General Assembly*, *29*, 2256293.

Giorgini, J. D. (2020). *JPL horizons overview and future plans*. https://lsst-sssc.github.io/Sprint2020/talks/Day-2_Giorgini_Horizons_LSST20200617.pdf

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

M. Gastineau, A. F., J. Laskar. (2021). *CALCEPH* (Version 3.5.0) [Computer software]. Institut de mécanique céleste et de calcul des éphémérides. https://www.imcce.fr/inpop/calceph/

Mihok, M. (2014). *Horizon-jpl*. GitHub. https://github.com/mihok/horizon-jpl

NASA NAIF. (2021). *NAIF spice data*. https://naif.jpl.nasa.gov/naif/data.html

Rhodes, B. (2019). *Skyfield: High precision research-grade positions for planets and earth satellites generator*. Astrophysics Source Code Library. https://ui.adsabs.harvard.edu/abs/2019ascl.soft07024R/abstract

The pandas development team. (2020). *Pandas-dev/pandas: pandas* (latest) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.3509134

tpltnt (pseudonymous). (2013). *Py-NASA-horizons*. GitHub. https://github.com/tpltnt/py-NASA-horizons