The Journal of Open Source Software

commensurability: a Python package for classifying astronomical orbits based on their toroid volume

Subhadeep Sarkar[1¶] **and Michael S. Petersen**[1]

1 University of Edinburgh ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Stars like our Sun orbit the center of our Milky Way galaxy. Over many orbits in the disc of the galaxy, the Sun will (hypothetically) probe every point in a 3D toroid defined by a minimum/maximum radius and minimum/maximum height above the midplane. However, a special class of stars are on orbits that repeat their tracks with every revolution around the galactic center, and therefore only probe a small sub-volume of the toroid. These orbits are called “commensurate”.

To study the orbital content of galaxies, astronomers will often use models for galaxies that allow for the integration of orbits within the model potential. The integration results in a time series of positions (e.g. [x-y-z]) for the orbit that can then be analyzed to produce a classification. Complicating this picture, potentials evolving with time introduce a new frequency: the pattern frequency of the evolution. The evolution of the potential might be the rotation of a galactic bar, the formation or dissolution of spiral arms, the growth of the galaxy, or the interaction of a satellite galaxy. Orbits that are commensurate with the pattern frequency are particularly important in galactic dynamics as they experience the same force fluctuations during every revolution: this causes changes to the typically conserved quantities of orbits (e.g. energy and angular momentum).

Given their importance, astronomers have developed techniques to identify commensurate orbits. One technique is “orbit tessellation”, which seems to estimate the (sub-)volume of the toroid traversed by an orbit (Petersen et al., 2021). Orbit tessellation aims to pick out commensurate orbits with relatively short orbit integration times, as well as pick out commensurabilities in cases where the kinematic frequencies need not stay constant (such as in a potential with multiple pattern frequencies from multiple causes).

The Python package presented here, commensurability, provides a straightforward Python framework and connection to powerful tools for modeling potentials to estimate the volume of a toroid that a given orbit fills.

Statement of need

The technique of measuring orbit volumes has previously been used to classify orbits in 2D in both fixed and evolving potentials, revealing families of commensurate orbits that are critical for galactic evolution (Petersen et al., 2021). However, the technique had not (1) been extended to 3D, and (2) did not have a user-friendly Python interface. The commensurability package solves both problems while providing interoperability with three leading galactic dynamics packages.

More generally, classifying orbits in model galaxies, let alone in models that evolve with time, is a non-trivial task. The primary method to classify orbits in galaxies relies on frequency analysis, such as with the superfreq (Price-Whelan, 2015) or naif (Silva et al., 2023) codes.

While frequency analysis is a useful tool, ambiguity in the classification remains. Frequency analysis relies on the stability of the frequencies of an orbit, which may miss short-lived families and smoothly changing frequencies. Orbit tessellation avoids these pitfalls. Orbit tessellation improves on fundamental frequency classifications, but also exists as an independent orbit classification scheme that can operate on orbits run in self-consistent simulations.

Additionally, computing the orbit volume provides an opportunity to efficiently search the model potential space for unique orbits. By creating a measure that is defined at all points in phase space (typically defined in the position and velocity of a radial extreme), one can search a model potential for commensurate orbits using optimization techniques.

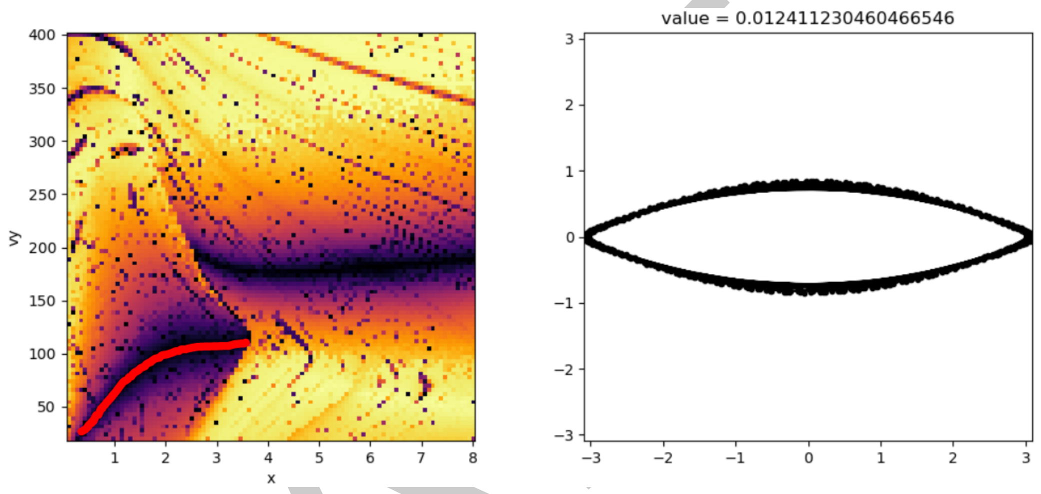

Figure 1: An example diagnostic from commensurability, where an orbit is selected from a map of toroid volumes (left panel). In the map, orbits with small volumes are dark-colored. The "tracks", including the curve highlighted in red, are families of orbits. The orbits are all in the frame of a rotating bar; the shape traced by this particular orbit in x-y position space (right panel) is a characteristic bar orbit, drawn from a point on the red curve.

Features and dependencies

Commensurability provides several notable features, with an emphasis on extensibility and compatibility with existing code.

Commensurability defines a framework for analyzing potentials in its "analysis" objects. "TessellationAnalysis" uses multiprocessing to efficiently compute the normalized toroidal volumes of orbits, or evaluate orbit "measures". Analysis objects are equipped with interactive matplotlib (Hunter, 2007) figures displaying a map of orbit measures as a function of phase space (for instance in x-vy). A user can explore the potential by visually selecting regions of interest, and the interactive plot will update with the x-y-z time series of the orbit. Since evaluating a large number of orbits can require significant computational power, analysis objects also include serialization and deserialization methods using the HDF5 format (The HDF Group, n.d.). Although TessellationAnalysis objects focus on orbit tessellation specifically, the analysis framework can be easily extended to any orbit evaluation method.

Commensurability provides a subpackage called tessellation that implements mathematical and computational improvements over early versions of orbit tessellation-based classification (Petersen et al., 2021). While only two- and three-dimensional orbits are evaluated in commensurability, the subpackage implements a general N-dimensional evaluation algorithm. Several normalizations are provided for the toroid volume, computed after trimming insignificant simplices based on their axis ratio, and new normalizations can be defined easily. The default normalization in 3D for the orbit volume is the convex hull of four rotated copies of the points around the z-axis. The rotated copies protect against missing orbits that only span a small range of azimuth about the z-axis. The convex hull is computed using QHull (Barber et al., 1996) as implemented in scipy (Virtanen et al., 2020). This subpackage can be used for its orbit evaluation function independent of commensurability.

Commensurability depends on the Python package pidgey for orbit integration. Pidgey is a standalone package that asserts a uniform interface to three major galactic dynamics packages—agama (Vasiliev, 2019), gala (Price-Whelan, 2017), and galpy (Bovy, 2015)—and its interface can be extended to more packages trivially. Pidgey uses astropy (Astropy Collaboration et al., 2022) SkyCoord objects to store orbit coordinates, a format familiar to most astronomers. Pidgey depends on iext, a defensive framework for extending the dependencies associated with a class without requiring the dependencies to be present. This enables pidgey to operate in the absence of a complete suite of orbit integration packages; the user need only have one of agama, gala, or galpy installed to use commensurability (but may have all three).

Provided example workflows and usage

To help users begin using the software in their research, we provide a readthedocs page with pip-based installation instructions, a quickstart, and examples drawn from real scientific applications. First, we feature a demonstration of orbital classification in a model of a rotating bar, where a bar orbit is detected by tracing a curve of vanishing toroid volume as shown in Figure 1. Second, we demonstrate orbital classification in the solar neighborhood for a standard Milky Way potential, revealing a rich dynamical structure of commensurate orbits.

The readthedocs page also hosts an extensive API reference, with all code adhering to the Black code style for ease of readability.

Acknowledgements

We thank Aneesh Naik and Eugene Vasiliev for discussions regarding early versions of the software.

References

Astropy Collaboration, Price-Whelan, A. M., Lim, P. L., Earl, N., Starkman, N., Bradley, L., Shupe, D. L., Patil, A. A., Corrales, L., Brasseur, C. E., Nöthe, M., Donath, A., Tollerud, E., Morris, B. M., Ginsburg, A., Vaher, E., Weaver, B. A., Tocknell, J., Jamieson, W., … Astropy Project Contributors. (2022). The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package. *935*(2), 167. https://doi.org/10.3847/1538-4357/ac7c74

Barber, C. B., Dobkin, D. P., & Huhdanpaa, H. (1996). The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, *22*(4), 469–483. https://doi.org/10.1145/235815.235821

Bovy, J. (2015). galpy: A python Library for Galactic Dynamics. *216*(2), 29. https://doi.org/10.1088/0067-0049/216/2/29

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Petersen, M. S., Weinberg, M. D., & Katz, N. (2021). Using commensurabilities and orbit structure to understand barred galaxy evolution. *500*(1), 838–858. https://doi.org/10.1093/mnras/staa3202

Price-Whelan, A. M. (2015). *SuperFreq: Numerical determination of fundamental frequencies of an orbit*. Astrophysics Source Code Library, record ascl:1511.001.

Price-Whelan, A. M. (2017). Gala: A Python package for galactic dynamics. *The Journal of Open Source Software*, *2*, 388. https://doi.org/10.21105/joss.00388

Silva, L. B. e, Debattista, V. P., Anderson, S. R., Valluri, M., Erwin, P., Daniel, K. J., & Deg, N. (2023). Orbital support and evolution of flat profiles of bars (shoulders). *The Astrophysical Journal*, *955*(1), 38. https://doi.org/10.3847/1538-4357/ace976

The HDF Group. (n.d.). *Hierarchical Data Format, version 5*. https://github.com/HDFGroup/hdf5

Vasiliev, E. (2019). AGAMA: action-based galaxy modelling architecture. *482*(2), 1525–1544. https://doi.org/10.1093/mnras/sty2672

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2