


Hardware-Accelerated Computation of Stellar Light Curves

LIONEL J. GARCIA,¹ SOICHIRO HATTORI,¹ AND DANIEL FOREMAN-MACKEY¹

¹*Center for Computational Astrophysics, Flatiron Institute, New York, NY, USA*

ABSTRACT

The measurement of stellar fluxes over time serves a wide variety of science cases. Transit and occultation light curves, for example, are used to characterize the orbital parameters of stellar systems, and infer the atmospheric properties of their transiting bodies. Phase curves, on the other hand, can be used to study the thermal emission of exoplanets, and the atmospheric properties of stars. In order to infer these properties, a fast, accurate and robust model of stellar light curves is needed, one that accounts for the non-uniform surface intensity of spherical bodies and their mutual occultations. The analytical framework known as **starry** satisfies these requirements, and was successfully applied over the years following its development. However, as datasets and model complexity grow, the tractable inference of these parameters becomes increasingly challenging, and often motivates biased approximations. In this paper, we present a new implementation of the **starry** framework in **jax**, a high-performance machine-learning library designed for large-scale computations. We describe the changes made to the original version of **starry** and evaluate the performance of its new implementation that we release in the open-source Python package **jaxoplanet**. Through this implementation, we provide the community with a differentiable model of stellar light curves, and enable the use of powerful machine-learning tools available in the **jax** ecosystem. 

INTRODUCTION

$$f = \mathbf{s}^T \mathbf{A} \mathbf{R}' \mathbf{R} \mathbf{y} \tag{1}$$

1. STARRY FORMALISM

2. OPTIMIZATION

2.1. Limb-darkening multiplicative maps

2.2. Spherical harmonics rotations

With *starry*, every light curve computation at a given time t involves a rotation of the spherical harmonics basis, from the rest-frame of the star (Figure 1, left) to its sky-projected orientation (Figure 1, right), with a final rotation relative to the position of the occulting body (see Figure 2. from Luger et al. 2019).

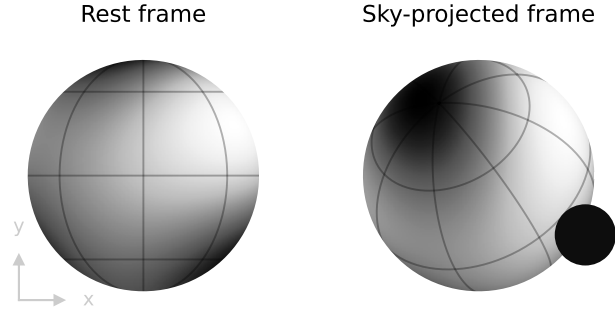


Figure 1. Rotation of the spherical harmonics basis to place the star in the sky-projected frame of the observer.

The rotation of spherical harmonics involves the computation of Wigner-D-matrices, commonly obtained through robust recursion relations in both degree l and order m , but leading to costly computations. Hence, knowing how to decompose the full spherical harmonics rotation using pre-computed rotations is essential to achieve optimal performance in the *starry* framework. Figure 2 shows the 6 elementary rotations currently used in the *starry* implementation.

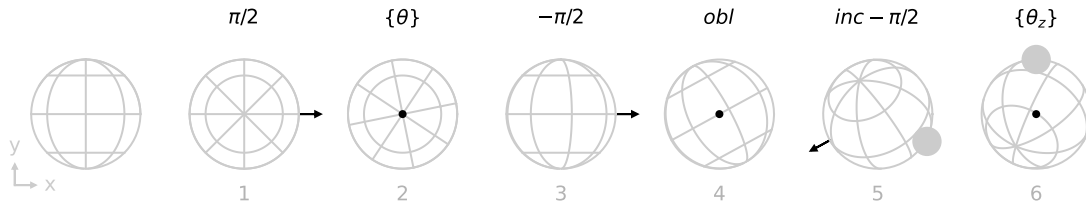


Figure 2. Consecutive rotations of the spherical harmonics basis in the *starry* implementation. Each schematic shows a sphere rotated from a previous orientation around an axis displayed as a black vector and an angle shown on top.

In this figure, rotations 1 to 3 correspond to a rotation of the star around its rotation axis, and rotations 3 to 5 place the star on a sky-projected frame. Finally, rotation

6 aligns the spherical harmonics map to the occulting body on the y axis in order to apply Green’s theorem in the appropriate basis (Figure 2 from [Luger et al. 2019](#)). The first thing to notice is that rotations 1 to 3 could be simply reduced to a rotation of angle θ about the y -axis, turning steps 1 to 3 into one. However, rotations around the pole, i.e. with a rotation axis along z , have simpler expressions that can be implemented separately. In addition, pre-computing these matrices at lower cost is particularly important for steps 2 and 6, involving a potentially large set of angles $\{\theta\}$ and $\{\theta_z\}$, defined over times $\{t\}$, for which to compute the full light curve. This explains the current decomposition of the complete rotation in six separate steps.

In `jaxoplanet`, we compute the Wigner-D-matrices by employing the Risbo recursion relations ([Risbo 1996](#)) implemented in `JAX` as part of the `s2fft` Python package ([Price & McEwen 2023](#)). In addition, we merge rotations 3, 4 and 5 (see [Figure 3](#)) into a single compound rotation of axis

$$\mathbf{v} = \frac{1}{\sqrt{1 - \cos^2\left(\frac{inc}{2}\right) \cos^2\left(\frac{obl}{2}\right)}} \begin{pmatrix} \sin\left(\frac{inc}{2}\right) \cos\left(\frac{obl}{2}\right) \\ \sin\left(\frac{inc}{2}\right) \sin\left(\frac{obl}{2}\right) \\ -\cos\left(\frac{inc}{2}\right) \sin\left(\frac{obl}{2}\right) \end{pmatrix}, \quad (2)$$

and angle

$$\omega = 2 \cos^{-1} \left(\cos\left(\frac{inc}{2}\right) \cos\left(\frac{obl}{2}\right) \right). \quad (3)$$

This way, the complete rotation reduces to the four separate steps shown in [Figure 3](#).

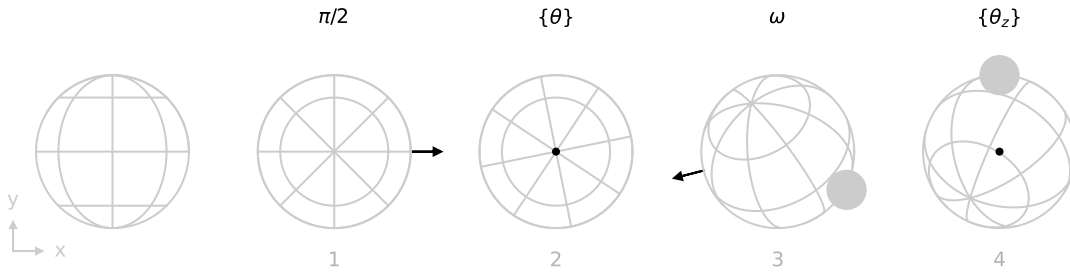


Figure 3. Consecutive rotations of the spherical harmonics basis in the `jaxoplanet` implementation. Each schematic shows a sphere rotated from a previous orientation around an axis displayed as a black vector and an angle shown on top.

2.3. Computation of the solution vectors

3. PERFORMANCE

Although *starry* occultation light curves have analytical expressions, their practical implementation relies on numerical approximations chosen to balance precision and computation time. This is the case, for example, of the solution vector \mathbf{s} , computed using numerical series expansions in *starry* (Luger et al. 2019, section D.2.3) and numerical integration in *jaxoplanet* (see section 2.3). Other expressions, such as the Wigner-D matrices used in the rotation of the spherical harmonics basis, are computed using recurrence relations prone to the accumulation of numerical errors and instability.

In this section, we evaluate the accuracy and speed of the *jaxoplanet* implementation and compare it with the C++ implementation of *starry*.

3.1. Precision

We evaluate the precision of the *starry* and *jaxoplanet* terms against quantities computed at arbitrary precision, using a ground truth version of the code implemented with the *mpmath* Python library¹. The precision of the overall flux f can only be understood by considering the precision of the terms involved in Equation 1: the solution vector \mathbf{s} (\mathbf{r} if no occultation), the basis matrices \mathbf{A} and \mathbf{B} , and the Wigner-D rotation tensor \mathbf{R} . Although we show in Figure 12 that the precision of the *starry* and *jaxoplanet* implementations differ for many of these terms, we focus this section on the solution vector \mathbf{s} and the overall flux f .

Assuming that we perform the numerical integration of the solution vector \mathbf{s} at a relatively high order ($n = 500$; see section 2.3), Figure 4 shows that our reimplementa-tion of *starry* reaches relative errors lower than 1 part per billion for \mathbf{s} and the overall flux f , considering both a small ($r = 0.01$) and a large ($r = 100$) occul-tor transiting the star accross numerically-sensitive values of impact parameters b . For reference, Figure 13 shows comparable errors on the same quantities computed with the C++ implementation of *starry*. In addition, Figure 5 shows the evolution of the error on f for increasing degrees of the spherical harmonics, both for *starry* and *jaxoplanet*. With these results, we validate the precision of *jaxoplanet* and show it is on par with the legacy C++ *starry* implementation² described in Luger et al. (2019).

¹ <https://mpmath.org/>

² version 1.2.0 (Luger et al. 2021)

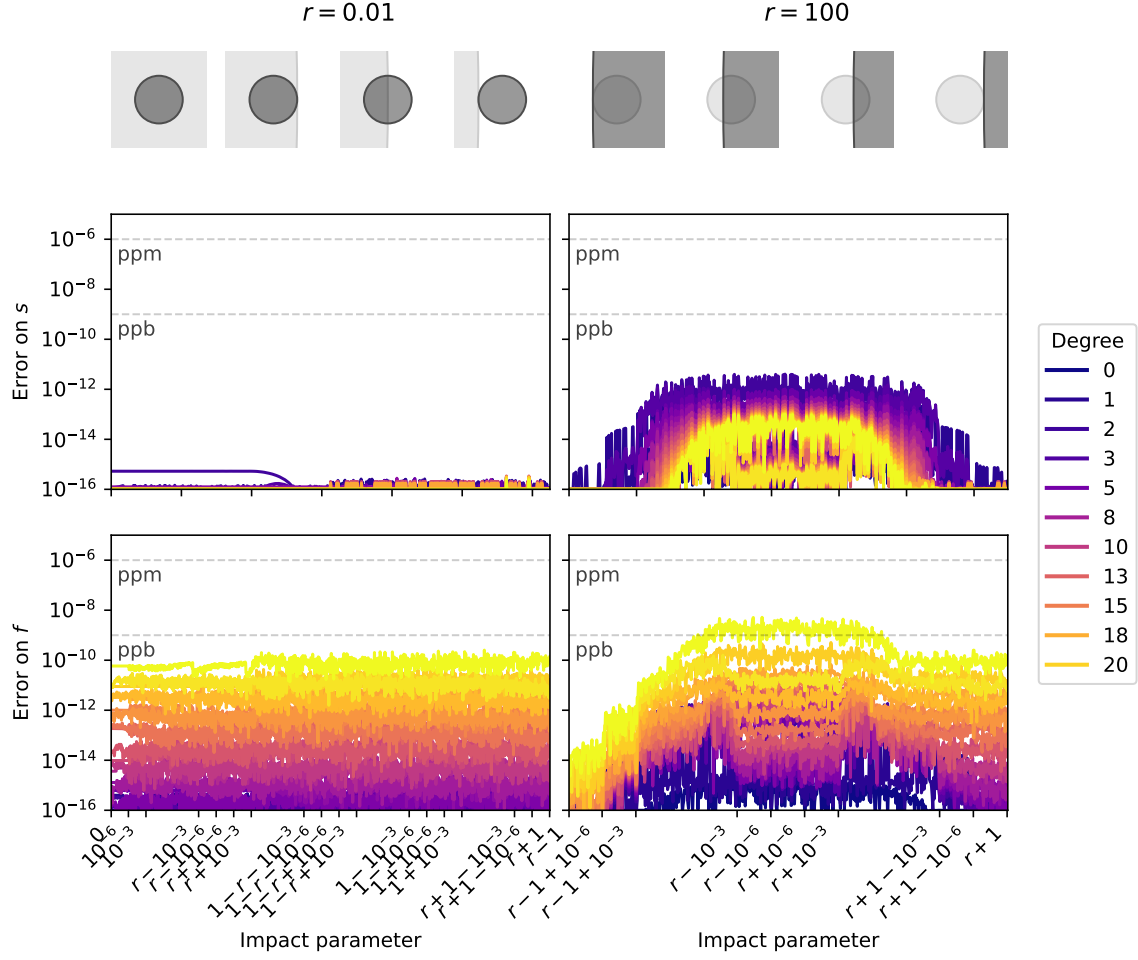


Figure 4. Error of the solution vector \mathbf{s} and the overall flux f for all terms of the spherical harmonics basis up to degree $l = 20$. Errors are computed against arbitrary-precision computations of \mathbf{s} and f for a small ($r = 0.01$) and a large ($r = 100$) occulter transiting the star across numerically-sensitive values of impact parameters b . For each (l, m) basis component all coefficients are set to zero except $y_{l,m} = 1$ and errors are scaled to the highest values of \mathbf{s} (respectively f) over b . Then, the plotted errors for each degree l are the maximum of the scaled errors over all $m \in [-l, l]$ components over b . The disks at the top of the figure show the transit configurations of the occulter (light dark) and the star (light gray) for different values of b . [↗](#)

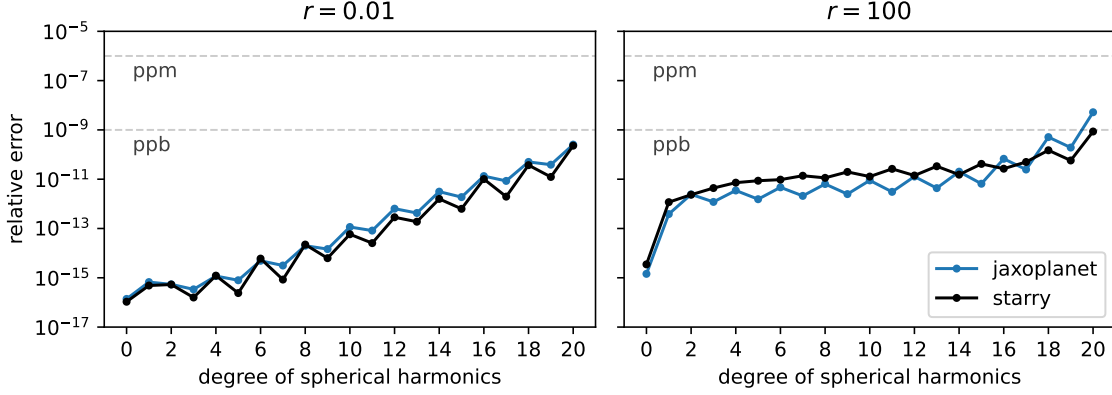


Figure 5. Maximum errors in the overall flux f for all degrees of the spherical harmonics components. Errors shown in this figure correspond to the maximum values of the scaled errors shown in Figure 4, maxima taken over the range of values b . [☞](#)

For reference, Figure 6 shows the relative error in the flux of a limb-darkened star occulted by an opaque body for increasing orders of a polynomial limb-darkening law. In this figure, we also report the error of the flux computed using the `exoplanet` Python package (limited to linear and quadratic limb-darkening).

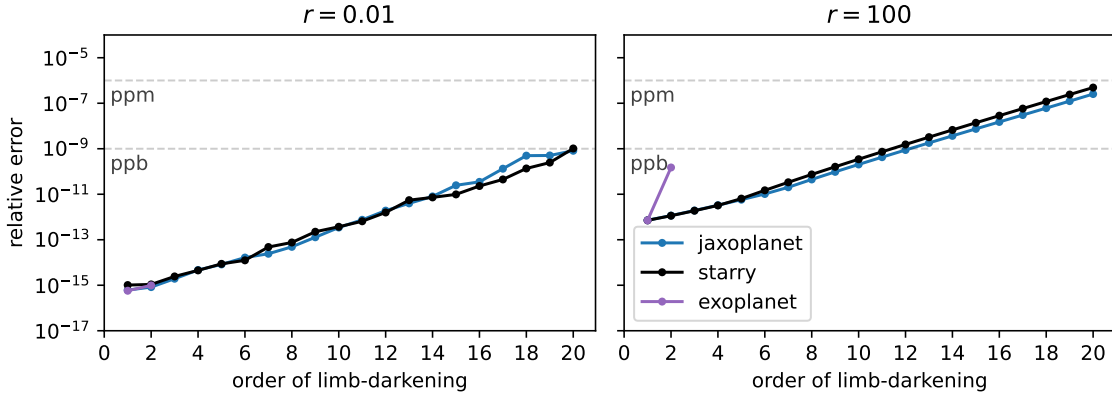


Figure 6. Maximum errors in the overall flux f of a star occulted by an opaque body for increasing orders of its polynomial limb-darkening law. Errors shown in this figure correspond to the maximum values of the scaled errors shown in Figure 4, taken over the range of values b . For each order n all limb-darkening coefficients are set to zero except $u_n = 1$. [☞](#)

A set of unitary polynomial limb-darkening coefficients of order 20 leads to spherical harmonics coefficients as large as 10^6 . This explains the larger errors observed in Figure 6 (error versus limb-darkening order) compared to the ones shown in Figure 5 (error versus degree of the spherical harmonics). We note that the errors scale with the values of u .

As described in [section 2.3](#), we compute *starry* light curves using the Gauss Legendre quadrature to approximate the \mathcal{P} integral involved in the expression of the solution vector \mathbf{s} . Hence, the precision of the computed flux f depends on the order of the Legendre polynomial which defines the number of points used to approximate \mathcal{P} . For this reason, users must carefully set the order parameter to reach the precision required for their application, as shown in [Figure 7](#).

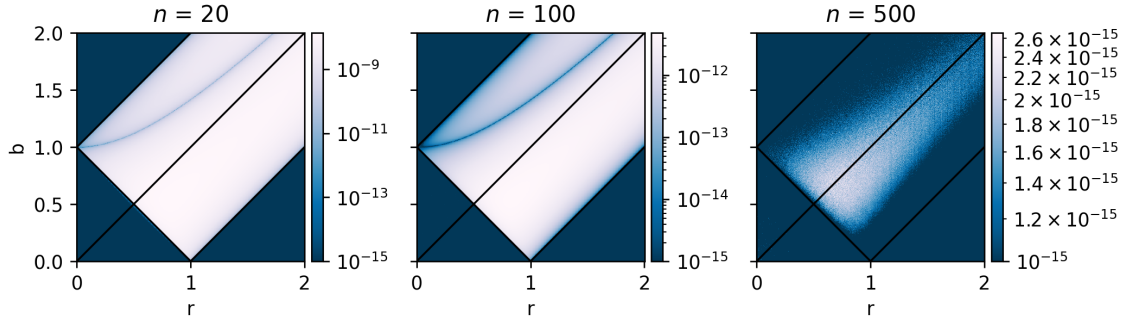


Figure 7. relative numerical error in the flux f of a linearly limb-darkened star (degree $l=1$) transited by an opaque companion over the (r, b) parameter space. The error is evaluated for different orders n of the Gauss-Legendre quadrature and computed against *starry* for speed and convenience. Note that [Figure 4](#) was produced using $n = 500$ (right plot) for which \mathbf{s} reaches machine precision. [↗](#)

In [Figure 8](#), we show how the error in the solution vector evolves with the order n of the Gauss-Legendre quadrature, and its impact on the error in the overall flux.

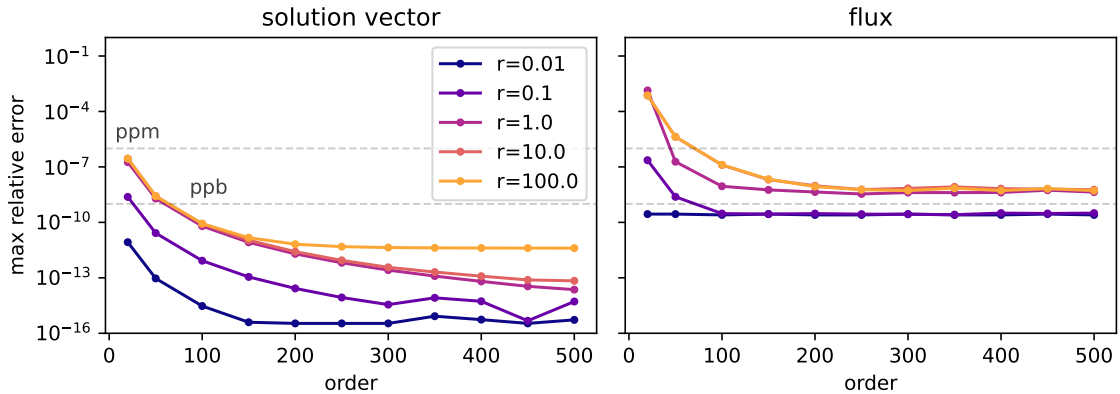


Figure 8. Maximum errors in the solution vector and overall flux f for different occulter radii and orders n of the Gauss-Legendre quadrature used to compute the \mathcal{P} integral (see [section 2.3](#)). Errors shown in this figure correspond to the maximum values of the scaled errors shown in [Figure 4](#), maxima taken over the range of values b . As in [Figure 4](#) the maximum degree of the spherical harmonics basis is set to $l = 20$.

The minimum error is always achieved for higher values of n , which comes with higher computational cost (see [section 3.2](#)). Hence users must carefully set the order parameter to balance precision and computational speed. As the minimum error reached also depends on the degree of the spherical harmonics basis used, [Figure 9](#) shows the errors on the flux for each degree of the spherical harmonics basis, for an occulter with relative radii $r = 0.01$, $r = 1$ and $r = 100$.

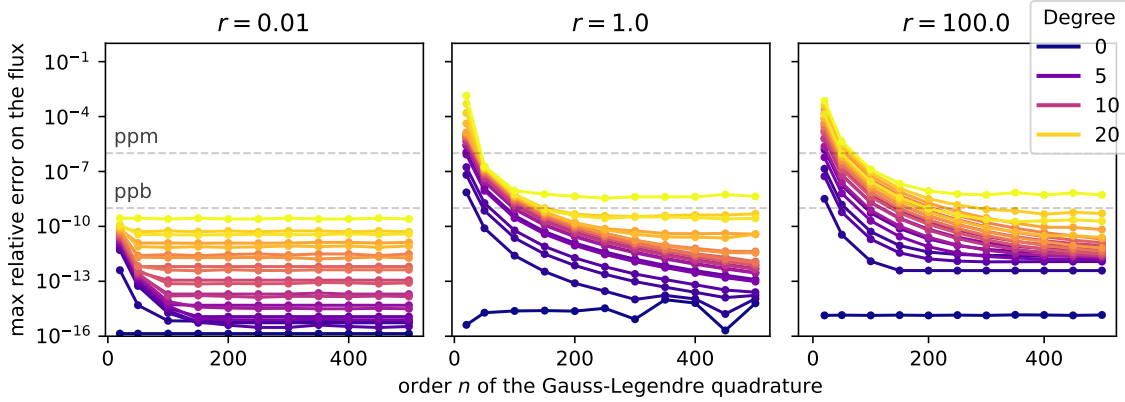


Figure 9. Maximum errors in the overall flux f for all degrees of the spherical harmonics components for different occulter radii and orders n of the Gauss-Legendre quadrature used to compute the \mathcal{P} integral (see [section 2.3](#)). Errors shown in this figure correspond to the maximum values of the scaled errors shown in [Figure 4](#), maxima taken over the range of values b .

3.2. Speed

One of the advantage of the JAX library is its ability to perform hardware-accelerated computations on CPUs and GPUs. In this section, we evaluate the speed of the `jaxoplanet` implementation and compare it with the C++ implementation of `starry`. For quadratically limb-darkened light curve, we also compare our implementation with the one provided by the `exoplanet` Python package.

In [Figure 10](#), we show the evolution of the computing time required to compute an occultation light-curve of a limb-darkened star and that of a non-uniform star described by spherical harmonics with a maximum degrees $l = 20$, depending on the order n of the Gauss-Legendre quadrature used to compute the \mathcal{P} integral numerically (see [section 2.3](#)).

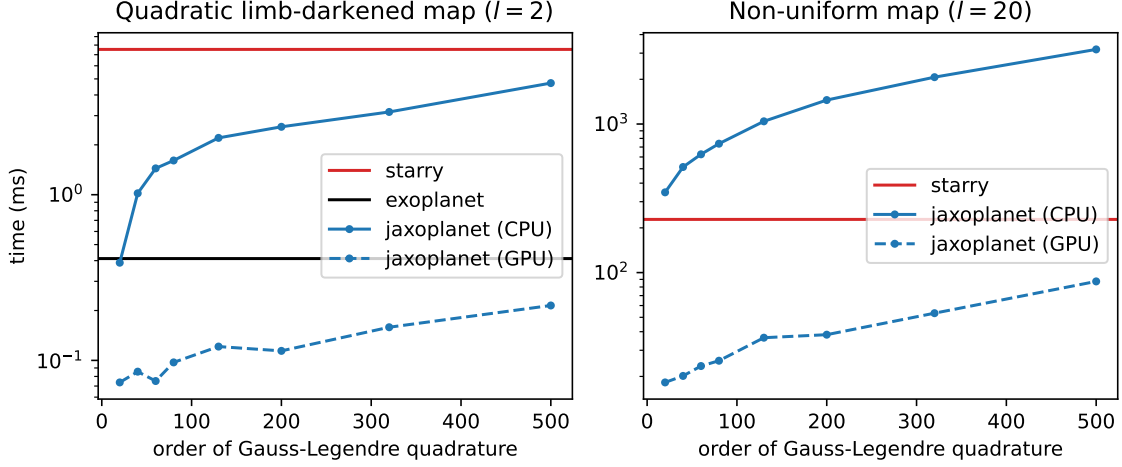


Figure 10. Computation time of the occultation light curve of a limb-darkened star (left) and a non-uniform star described by spherical harmonics up to order $l = 20$ (right) depending on the order n of the Gauss-Legendre quadrature used to compute the P integral numerically (see section 2.3). The processing time reported for `exoplanet` and `starry` is independent of n as these two codes provide closed-form solutions for the integral \mathcal{P} . Light curves are computed for 10 000 points in transit and an occulter radius $r = 0.1$.

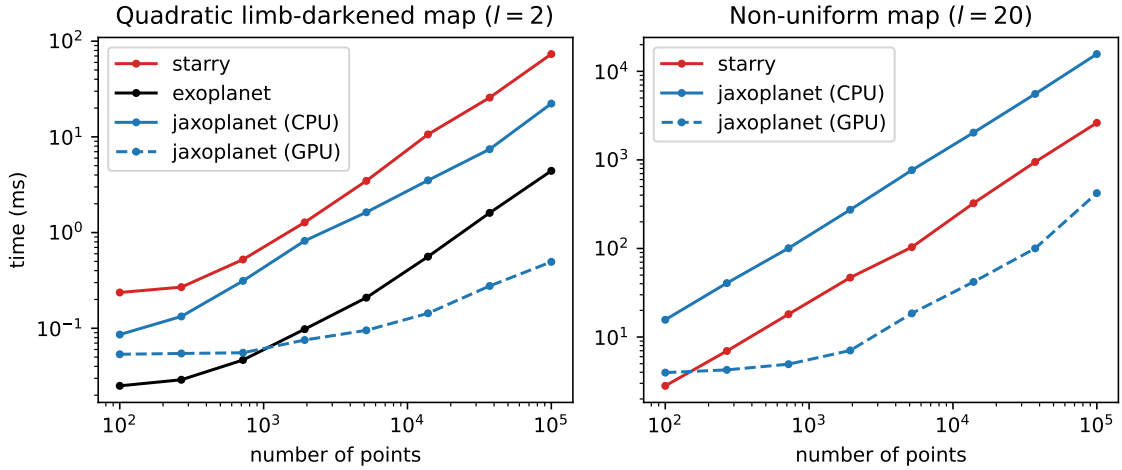


Figure 11. Computation time of the occultation light curve of a limb-darkened star (left) and a non-uniform star described by spherical harmonics up to order $l = 20$ (right) depending on the number of points in the light curve. Light curves are computed at order $n = 200$ and an occulter radius $r = 0.1$.

Figure 10, in combination with Figure 9, can be used to determine a value of n balancing precision and computation time.

Finally, Figure 11 shows the evaluation time of `jaxoplanet` as a function of the number of points in the light curve, compared to `starry` for an $l = 20$ surface map,

and **exoplanet** for a quadratically limb-darkened star. As in the previous figure, we show that **jaxoplanet** is competitive against state-of-the-art codes on CPUs and offer a clear advantage on GPUs.

4. CASE STUDIES: ...

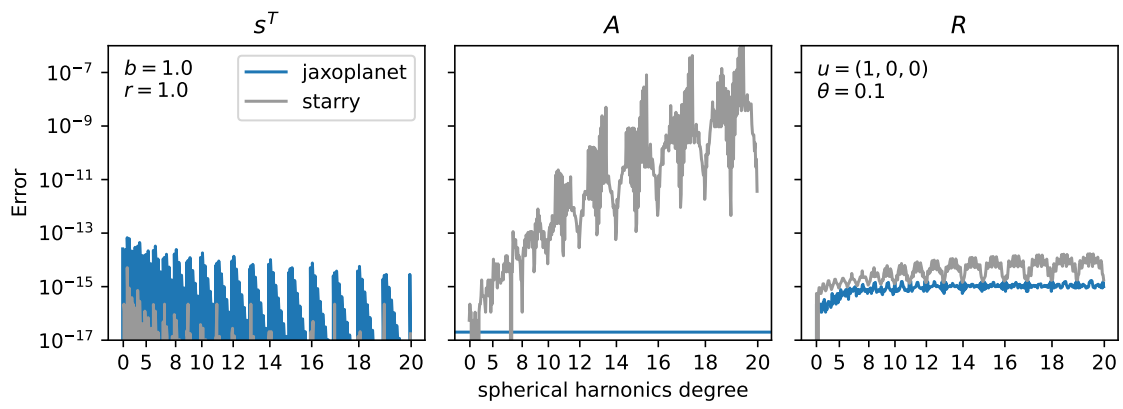

4.1.

REFERENCES

- | | |
|--|---|
| <p>Luger, R., Agol, E., Foreman-Mackey, D.,
et al. 2019, <i>The Astronomical Journal</i>,
157, 64, doi: 10.3847/1538-3881/aae8e5</p> <p>—. 2021, <i>rodluger/starry</i>: v1.2.0, v1.2.0,
Zenodo, doi: 10.5281/zenodo.5567781</p> | <p>Price, M. A., & McEwen, J. D. 2023,
<i>Journal of Computational Physics</i>,
submitted</p> <p>Risbo, T. 1996, <i>Journal of Geodesy</i>, 70,
383, doi: 10.1007/BF01090814</p> |
|--|---|

APPENDIX

A. ERROR BUDGET

Figure 12. 

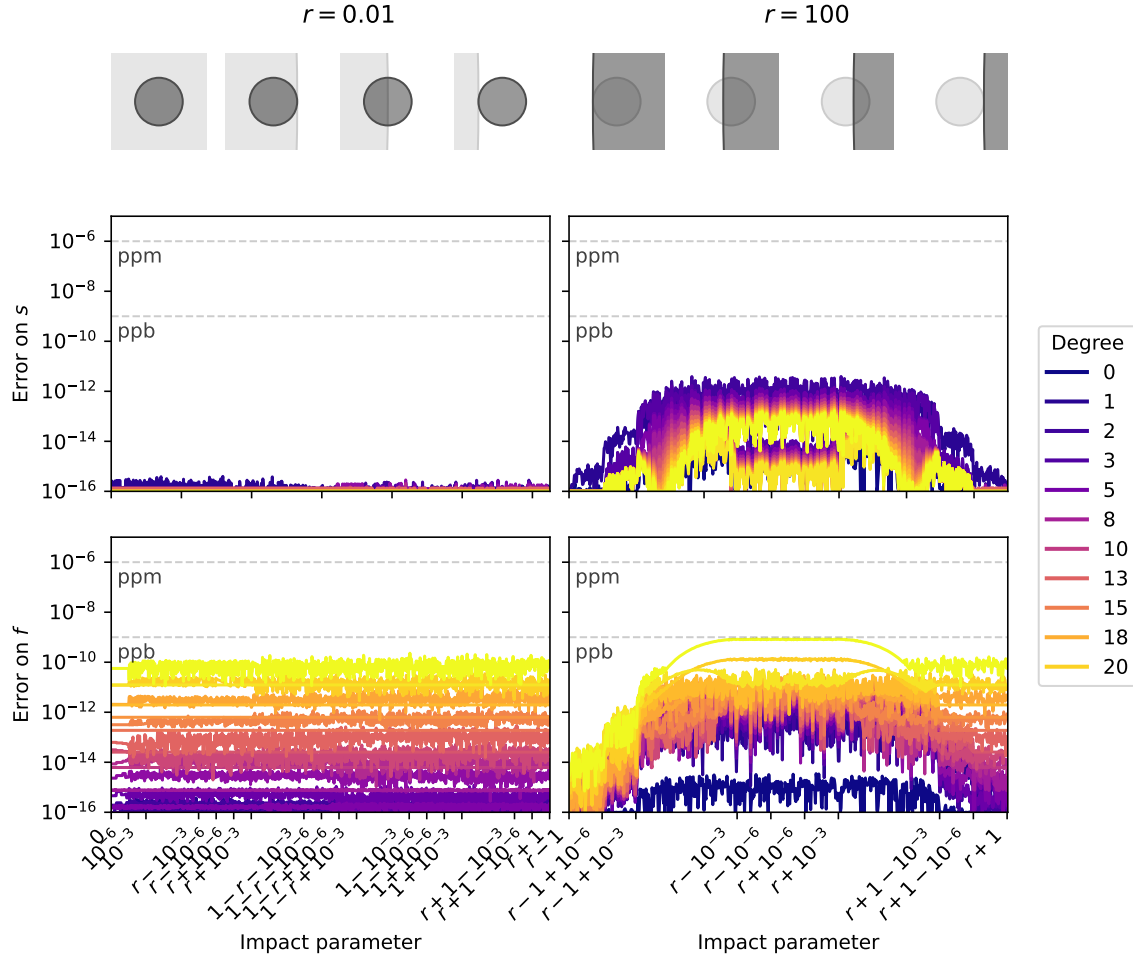


Figure 13. Same as Figure 4 but with $bvecs$ and f computed with the C++ implementation of starry. [\[4\]](#)

Table 1. Symbols used in this paper

Symbol	Definition	Reference
ω	Rotation angle of the combined rotation	Equation 3
n	order of the Gauss-Legendre approximation	
r	occultor radius in units of occulted body's radius	