


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).

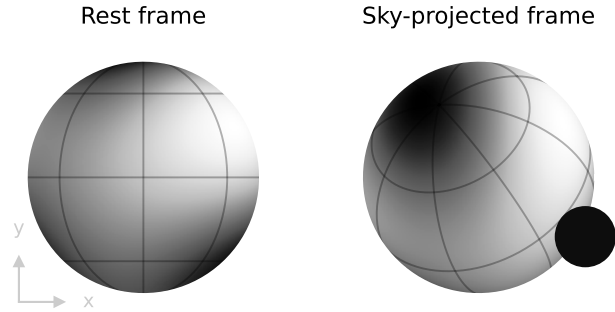


Figure 1. Rotation of the spherical harmonics basis to place the star in the sky-projected frame of the observer.

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.

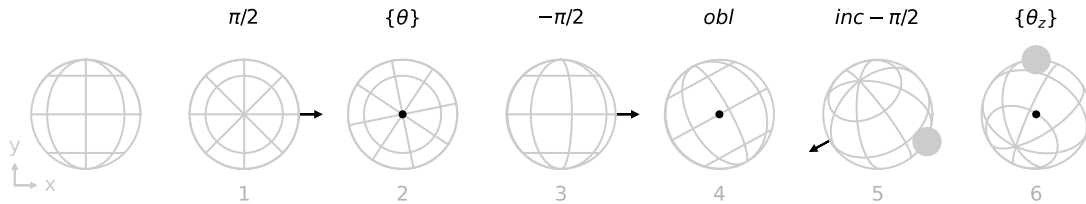


Figure 2. Consecutive rotations of the spherical harmonics basis in the *starry* implementation. Each schematic shows a sphere rotated from a previous orientation around an axis displayed as a black vector and an angle shown on top.

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](#).

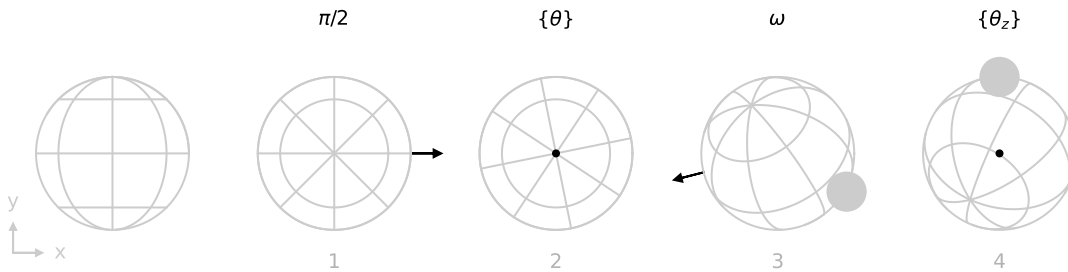


Figure 3. Consecutive rotations of the spherical harmonics basis in the `jaxoplanet` implementation. Each schematic shows a sphere rotated from a previous orientation around an axis displayed as a black vector and an angle shown on top.

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 7 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 reimplement 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 ($r = 100$) occulor transiting the star accross numerically-sensitive values of impact parameters b . For reference, Figure 8 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)

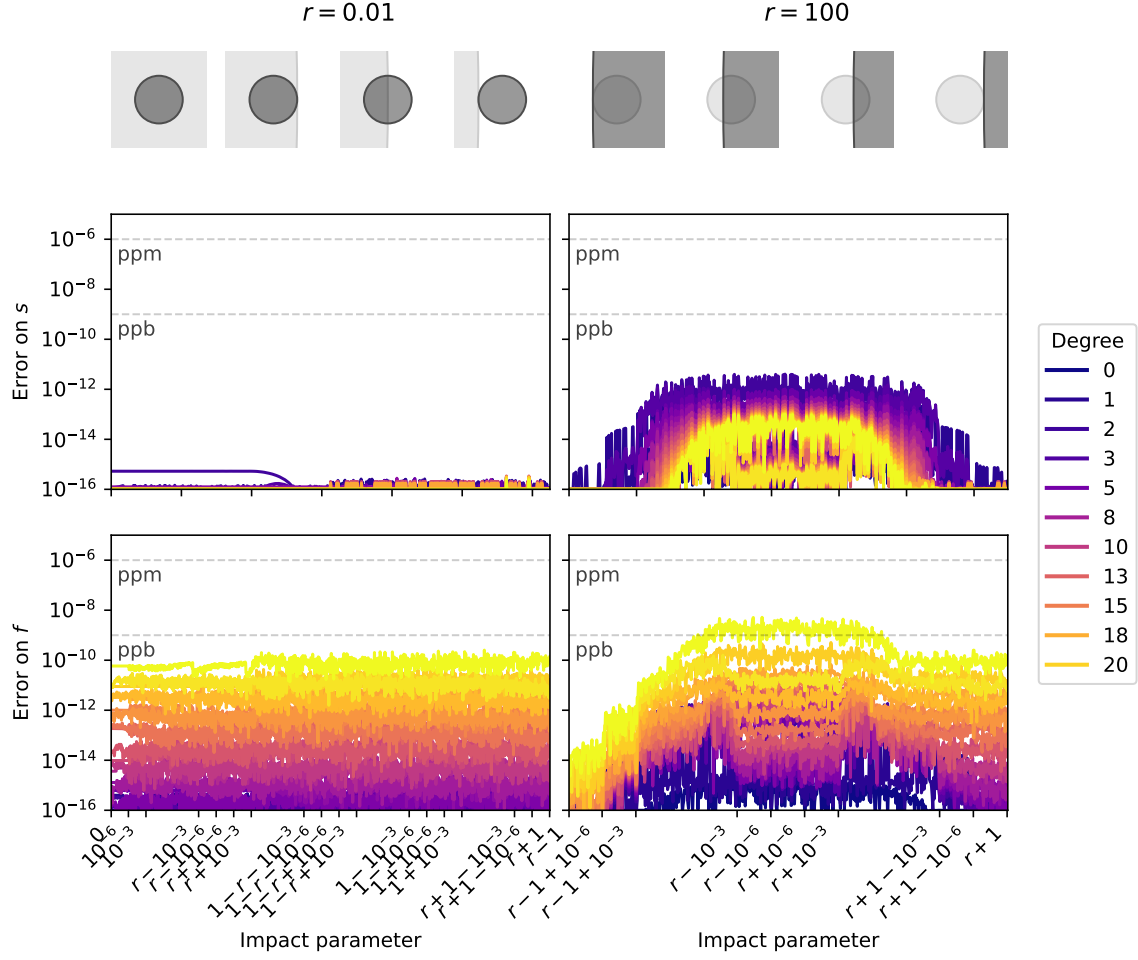


Figure 4. Error of the solution vector \mathbf{s} and the overall flux f for all terms of the spherical harmonics basis up to degree $l = 20$. Errors are computed against arbitrary-precision computations of \mathbf{s} and f for a small ($r = 0.01$) and a large ($r = 100$) occultor transiting the star across numerically-sensitive values of impact parameters b . For each (l, m) basis component, 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 occultor (light dark) and the star (light gray) for different values of b . [↗](#)

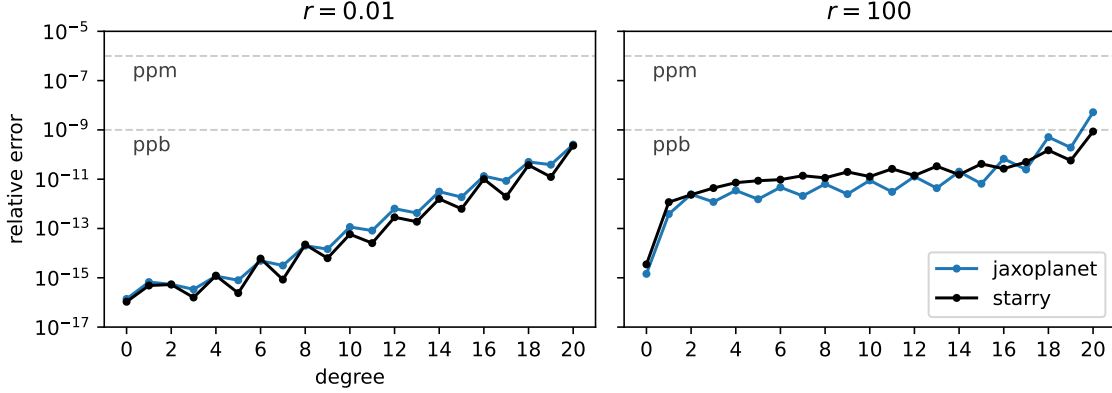


Figure 5. Maximum errors in the overall flux f for all degrees of the spherical harmonics components. Errors shown in this figure correspond to the maximum values of the scaled errors shown in Figure 4, maxima taken over the range of values b . [☞](#)

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 6.

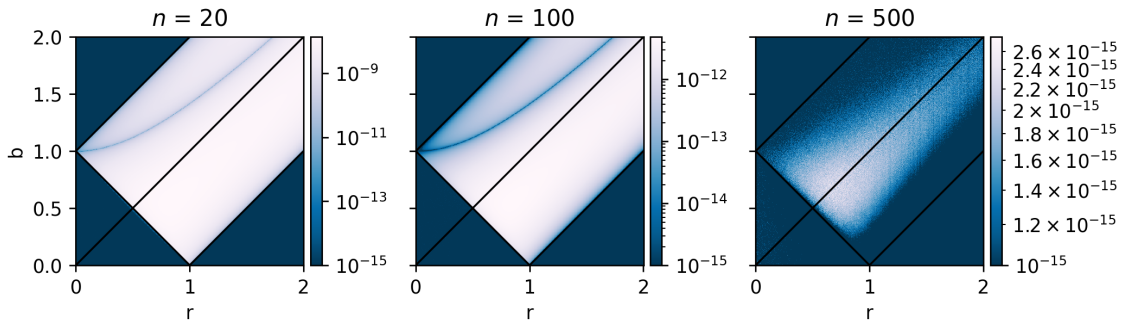


Figure 6. relative numerical error in the flux f of a linearly limb-darkened star transited by an opaque companion over the (r, b) parameter space. The error is evaluated for different orders n of the Gauss-Legendre quadrature and computed against *starry* for speed and convenience. Note that Figure 4 was produced using $n = 500$ (right plot) for which \mathbf{s} reaches machine precision.

At a fixed occulter radius, ?? shows the time required to compute the flux of the system for a range of N impact parameters b , plotted against the maximum relative error reached, the order n of the Gauss-Legendre approximation, and the hardware used for the computation.

These computations were run on an apple M1 CPU and an NVIDIA ... GPU. Thanks to the use of JAX, no hardware-specific...

By computing \mathcal{P} numerically, we show that our vectorized implementation reaches machine-precision on a GPU, up to an order of magnitude faster than the current C++ implementation of `starry` ran on a CPU.

Plot of the precision versus speed of the computed light curves.

Plot of the solution vectors computation similar to figure 11 and 12 of luger (maybe in the appendix)

3.2. *Speed*

4. CASE STUDY: ...

APPENDIX

A. ERROR BUDGET

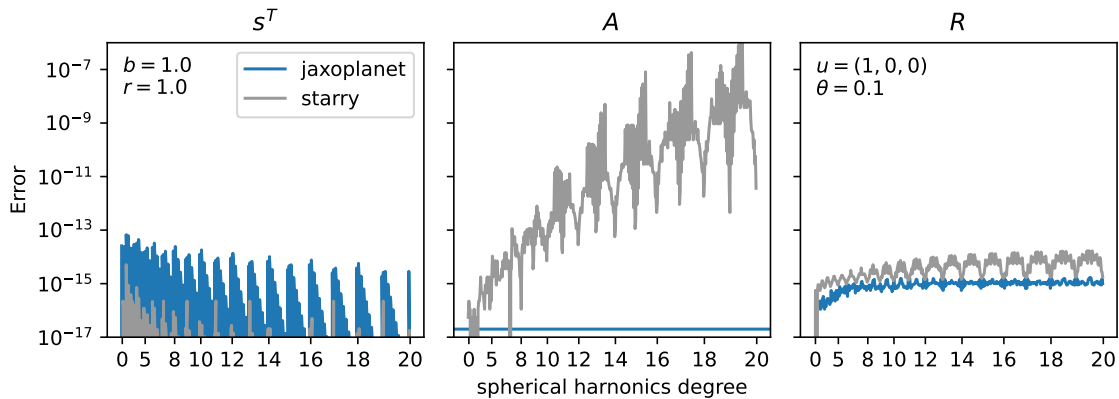


Figure 7. 

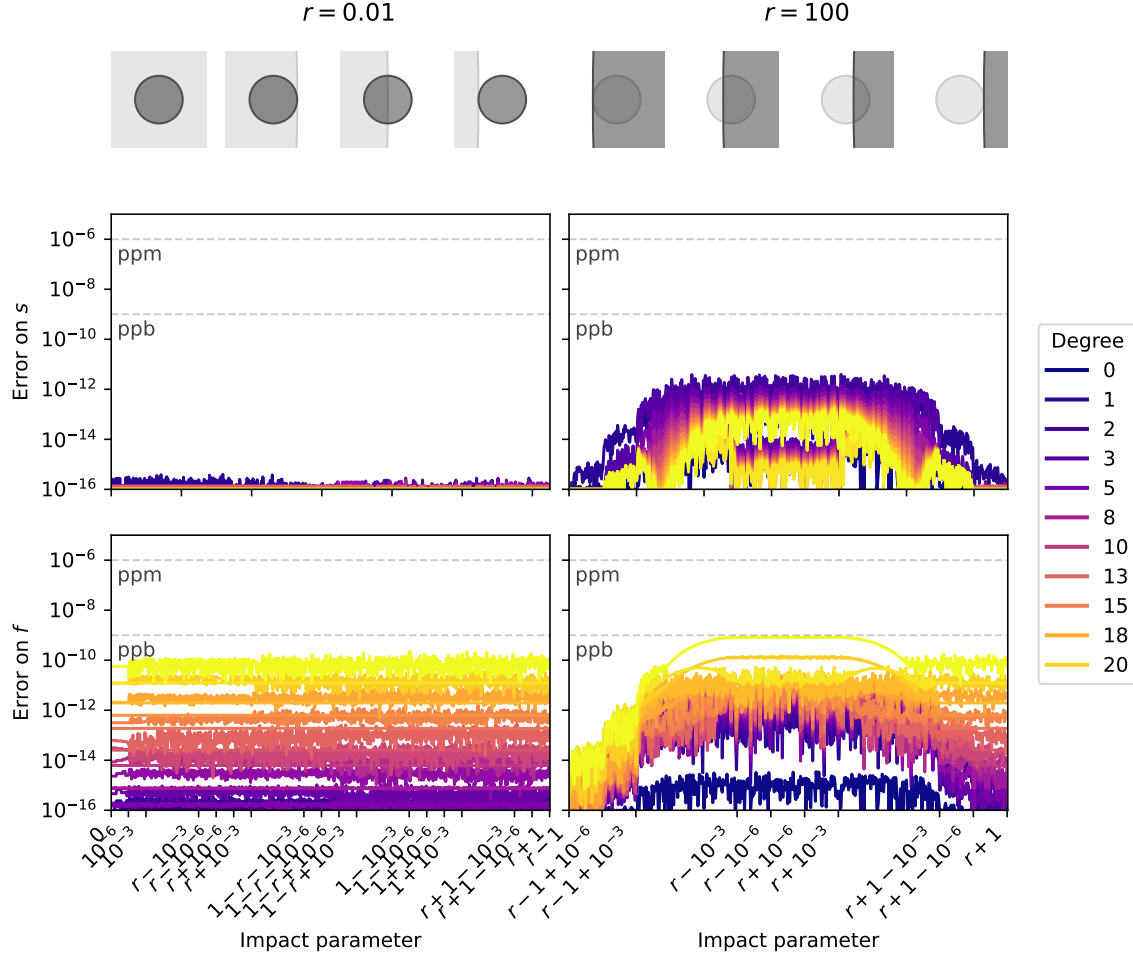


Figure 8. Same as Figure 4 but with $bvecs$ and f computed with the C++ implementation of `starry`. [↗](#)

REFERENCES

- Luger, R., Agol, E., Foreman-Mackey, D., et al. 2019, *The Astronomical Journal*, 157, 64, doi: [10.3847/1538-3881/aae8e5](https://doi.org/10.3847/1538-3881/aae8e5)
- . 2021, `rodluger/starry: v1.2.0, v1.2.0`, Zenodo, doi: [10.5281/zenodo.5567781](https://doi.org/10.5281/zenodo.5567781)
- Price, M. A., & McEwen, J. D. 2023, *Journal of Computational Physics*, submitted
- Risbo, T. 1996, *Journal of Geodesy*, 70, 383, doi: [10.1007/BF01090814](https://doi.org/10.1007/BF01090814)