jaxoplanet: Hardware-Accelerated Orbits and Light Curves

Lionel J. Garcia,[1] Soichiro Hattori,[1] and Daniel Foreman-Mackey[1]

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

ABSTRACT

Modeling light curves of celestial bodies serves a wide variety of science cases. From the orbital characterization of stellar systems to the study of exoplanet and stellar atmospheres. In this context, the analytical framework known as *starry* provided the community with a powerful tool to analytically compute light curves of bodies with non-uniform surface intensities. However, as datasets and models complexity grow, the tractable inference of their physical parameters becomes increasingly challenging, and often motivates biased approximations. In this paper, we present a new implementation of the starry C++ implementation 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 our 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. ⬡

1. INTRODUCTION

Planets of the solar system used to be known and studied through their position in the sky. It is only starting in the 16th century that Earth-based telescopes progressively enabled the direct imaging of their surface. But despite these new instruments, some minor planets were too small and too distant to be imaged, and remained unresolved until the end of the 20th century, when more capable space-based observatories were launched. This is the case of Pluto, for example, whose surface was first studied thanks to the light it reflects while rotating and orbiting the Sun. Using point source observations obtained from 1945 to 1986, Drish et al. (1995) tentatively reconstructed the map of the planet, and only later compared it to the first *Hubble Space Telescope* low-resolution images obtained by Stern et al. (1997). This mapping effort was gloriously completed with the extremely detailed pictures obtained by the *New Horizons* spacrecraft during its fly-by of the planet (Stern et al. 2015).

Although imaging the surface of an exoplanet might be several decades away, their

study follows that of Pluto, by first employing indirect methods and unresolved light curves (see Mayor & Queloz 1995 and Charbonneau et al. 2000) while waiting for the development of more powerful direct imaging instruments . Similarly, stellar photospheres can currently be mapped using interferometry and Doppler imaging, but these techniques are costly and can only be applied to a handful of objects, so that time-resolved spectrophotometry remains our main way to study them. In this context, the development of accurate and robust models of stellar light curves is crucial.

In the age of wide-field surveys (such as the *Transiting Exoplanet Survey Satellite*; Ricker et al. 2015) and high-precision spectroscopic instruments, these models are being compared to large datasets in probabilistic ways, requiring fast and robust evaluations of their posterior likelihood over larger and larger parameter spaces. These inferences can be highly facilitated by gradient-based optimization and sampling algorithms (see for example Betancourt 2017), so that being able to differentiate these models with respect to their parameters offers clear advantages.

In this context, Luger et al. (2019) and Agol et al. (2020) presented a framework to analytically compute the light curves of spherical bodies with non-uniform surface intensities, such as stars and exoplanets, and the flux observed from their mutual occultations. This framework, known as *starry*, has been successfully applied to a wide variety of science cases, like the study of transiting exoplanets (Kostov et al. 2019), precise modeling of radial velocities (Dawson et al. 2021), the study of exoplanets atmospheres (Bell et al. 2023), the study of stellar photospheres (Almenara et al. 2022), or even the surface mapping of stellar system objects (Bartolić et al. 2022). However, as datasets and models complexity grow, the tractable inference of the larger parameter spaces we have to deal with becomes increasingly challenging, and often motivates biased approximations. This is the case, for example, in transmission spectroscopy measurements conducted with the *James Webb Space Telescope* NIR-Spec instrument, that should ideally consist in the simultaneous modeling of more than 3000 transit light curves, one per pixel element (e.g. Alderson et al. 2023).

In this paper we present jaxoplanet, a new implementation of the *starry* framework in JAX, a high-performance machine-learning library designed for large-scale computations, including autodifferentiation capabilities, with code compiled and deployable on various hardware accelerators. In section 2, we summarize the formalism of *starry*, introduced by Luger et al. (2019) to analytically compute stellar light curves. In section 3, we describe the jaxoplanet implementation and the changes made to the starry implementation. Finally, in section 4, we evaluate the performance of our 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.

For convenience, we provide links to Python code and Jupyter notebooks ( ) to reproduce all of the figures throughout the paper. Code to generate this paper is hosted at https://github.com/lgrcia/paper-jaxoplanet and makes used of jaxoplanet version $3.0.0$[1].

2. LIGHT CURVE MODELS

Inspired by the idea of Pál (2012), Luger et al. (2019) and Agol et al. (2020) present a general framework to compute the light curves of spherical bodies with non-uniform surface intensities. In this section, we describe the mathematical formalism behind the *starry* model.

2.1. *Spherical harmonics*

The surface of a spherical body can be naturally described by real two-dimensional spherical harmonics: a set of special functions defined on the sphere that can serve as orthonormal basis to represent any spherical surface map. If $\tilde{\mathbf{y}}$ denotes the spherical harmonics basis of degree l_{max} (defined in Luger et al. 2019, section 2.2) and $\mathbf{y}$ a vector that contains the coefficients describing the surface in this basis, the intensity I of the map projected onto the sky-projected (x, y) plane of the observer is

$$I(x, y) = \tilde{\mathbf{y}}^{\mathsf{T}}(x, y)\, \mathbf{y}. \tag{1}$$

In order to orient this surface, for example with a given inclination, obliquity and phase around its rotation axis, we define a rotation matrix $\mathbf{R}$ that acts on the spherical harmonics coefficients $\mathbf{y}$, such that the intensity of the map projected onto the sky of the observer is

$$I(x, y) = \tilde{\mathbf{y}}^{\mathsf{T}}(x, y)\, \mathbf{R}\, \mathbf{y}. \tag{2}$$

2.2. *Rotation light curve*

Using this expression, the integrated intensity of the rotated map is

$$F = \iint I(x, y)\, \mathrm{d}S = \iint \tilde{\mathbf{y}}^{\mathsf{T}}\mathbf{R}\mathbf{y}\, \mathrm{d}S \tag{3}$$

In order to simplify this integral, Luger et al. 2019 introduce a change of basis from the spherical harmonics basis $\tilde{\mathbf{y}}$ to a simple polynomial basis $\tilde{\mathbf{p}}$ (defined in Luger et al.

---

[1] TODO make release

4

2019, section 2.3) characterized by the change of basis matrix $\mathbf{A_1}$. In this new basis, $\mathbf{y}$ can be expressed as

$$\mathbf{p} = \mathbf{A_1 y},\tag{4}$$

such that

$$I(x, y) = \mathbf{\tilde{p}}^\mathsf{T}(x, y)\,\mathbf{A_1 R y}.\tag{5}$$

Hence,

$$F = \iint \mathbf{\tilde{p}}^\mathsf{T}\mathbf{A_1 R y}\,\mathrm{d}S = \mathbf{r}^\mathsf{T}\mathbf{A_1 R y},\tag{6}$$

with

$$\mathbf{r}^\mathsf{T} \equiv \iint \mathbf{\tilde{p}}^\top\,\mathrm{d}S,\tag{7}$$

since $\mathbf{A_1}$, $\mathbf{R}$ and $\mathbf{y}$ are independent of the coordinates on the sphere.

As $\mathbf{\tilde{p}}$ is simply made of polynomials of the coordinates x, y and z, the elements of the *rotation vector* $\mathbf{r}^\mathsf{T}$ can be computed analytically and only once for a given l_{max} (see Luger et al. 2019, Eq. 20).

2.3. *Occultation light curve*

In order to model transits and occultations, the same formalism can be used to compute the integrated intensity of a surface map occulted by another spherical body. In this case, the intensity of the map must be integrated only over the unocculted surface S of the disk, such that

$$F = \iint_S I(x, y)\,\mathrm{d}S = \left(\iint_S \mathbf{\tilde{p}}^\mathsf{T}\,\mathrm{d}S\right)\mathbf{A_1 R y}.\tag{8}$$

To simplify this expression, Pál (2012) note that any two-dimensional integral of the unocculted disk can be reduced to a one-dimensional integral over the contour C of the occulting body using Green's theorem[2]. This can be done if the integrand of the integral over S can be described by a function f such that

$$f(x, y) = \frac{\delta G_y}{\delta x} - \frac{\delta G_x}{\delta y},\tag{9}$$

where G is a vector field defined and differentiable over S.

By applying a last change of basis matrix $\mathbf{A_2}$ to the polynomial basis $\mathbf{\tilde{p}}$, Luger et al. (2019) transforms $\mathbf{\tilde{p}}$ to the *Green's basis* $\mathbf{\tilde{g}}$ that satisfies this property, meaning

---

[2] Also known as Stokes' theorem in three-dimensional vector calculus.

that the surface integral of each of its components $\tilde{g}_n$ can be transformed to the line integral

$$s_n \equiv \iint_S \tilde{g}_n(x, y) \, \mathrm{d}S = \oint_C \mathbf{G}_n(x, y) \, \mathrm{d}\mathbf{r}, \tag{10}$$

with $\mathbf{G}_n$ given in Luger et al. 2019, Equation 34. These functions require the occulting body to be aligned with the y-axis of the reference frame, which can be achieved by applying a final rotation matrix $\mathbf{R}'$ to the surface map.

Using these expressions, the integrated intensity of the occulted map follows the idiomatic linear expression

$$F = \mathbf{s}^{\mathsf{T}} \mathbf{A} \, \mathbf{R}' \, \mathbf{R} \, \mathbf{y}, \tag{11}$$

with $\mathbf{A} = \mathbf{A_2 A_1}$ and the integrals s_n arranged in the *solution vector* $\mathbf{s}^{\mathsf{T}}$.

2.4. *Limb darkened surfaces*

A simpler situation is encountered when the surface of the spherical body can be described by a radially-symmetric intensity profile $I(r)$, such as a limb darkened star. In this case, the vector of spherical harmonics coefficients $\mathbf{y}$ is very sparse (because all $m \neq 0$ components are zero) and the surface map doesn't need to be rotated.

Contemporary to Luger et al. (2019), Agol et al. (2020) explore this case and provide simple expressions for the integrated intensity of a star whose surface can be described by a polynomial limb darkening law and occulted by an opaque body. In this case:

1. The surface is simply described by $\mathbf{u}$, a vector of polynomial limb darkening coefficients.

2. The Green's basis takes a simpler form (see Agol et al. 2020, Equation 14).

3. Matrices involved in the computation of the integrated flux have l_{max} columns, versus $(l_{max} + 1)^2$ in the general case, reducing drastically the computational cost of the model.

For these reasons, and because of its wide application, our implementation treats radially-symmetric maps separately, leading to highly optimize computations of limb darkened light curves.

3. IMPLEMENTATION DETAILS

The formalism introduced by Luger et al. (2019), summarized in section 2, provides an elegant analytical model of stellar light curves. Implemented in C++ and released in the starry Python package, it was built for the probabilistic programming library PyMC3 and led to several advances in the field of exoplanetary science. However, since its initial development, the PyMC project adopted several breaking changes that made starry incompatible with its latest versions. Updating starry to the latest version of PyMC would require a complete rewriting of the code, that would again be highly dependent on the latter. Hence, our first motivation comes from the broken dependencies of starry and aims to provide the community with an implementation compatible with a wider ecosystem of tools. In this respect JAX represents an interesting alternative. This high-performance machine-learning library can be used to perform hardware-accelerated computations on various types of devices (CPU, GPU, TPU) and gave rise to probabilistic tools developed by an ever-growing community of scientists and engineers. However, JAX capabilities really shine when models are written with a functional-programming mindset, and computations performed in a vectorized fashion that allows parallelization.

In this section, we describe the changes made to the original starry implementation to allow its rewriting in JAX. We also highlight details about the latest version of starry's original implementation that were omitted in Luger et al. 2019 but described in parts of its code and documentation.

3.1. Limb darkened maps

Limb darkened maps are obtained by multiplying a base map, devoid of limb darkening, with a pure limb darkened map. If $\mathbf{U}$ is the change of basis matrix that transforms the polynomial limb darkening coefficients $\mathbf{u}$ to the spherical harmonics coefficients $\mathbf{y}_u$, the pure limb darkened map is

$$\mathbf{y}_u = \mathbf{U}\mathbf{u}. \tag{12}$$

The multiplication of the base map represented by $\mathbf{y}$ with the limb darkening map represented by $\mathbf{y}_u$ can be seen as the multiplication of two polynomials, only with special coefficients (called Clebsch–Gordan coefficients) inherent to the properties of spherical harmonics. To simplify this operation, we first transform both maps into the polynomial basis before multiplying them, so that their product is the actual product of two polynomials which doesn't require the computation of special coefficients.

3.2. Limb darkening normalization

In starry version 0.2.2 (Luger et al. 2019), the normalization of a limb-darkened light curve was done by assuming that adding limb darkening to a non-uniform surface does not modify its total luminosity. Hence, the total flux received from a star during

its complete rotation should be the same with or without limb darkening. In older versions of starry ($\leq$0.2.2), this was done by actually computing the light curve of the star with and without limb darkening, and using the ratio of the two to normalize the limb darkened light curve. As discussed in the **starry** documentation[3] this approach is wrong and leads to an unphysical normalization.

Instead, one can assume that the total flux of the star should be similar for all observers, independently of the surface map of the star. In addition to the fact that all components of the spherical harmonics basis integrate to 0 (except the constant component $Y_{0,0}$) we deduce that the mean flux of the star must be coming from a static surface seen by any observer, no matter his viewpoint, which corresponds to a purely limb darkened disk (with no other source of non-uniformity).

The flux of this purely limb darkened disk is

$$f_u = \mathbf{r}^\mathsf{T} \mathbf{A_1} \mathbf{y}_u \tag{13}$$

with $\mathbf{y}_u$ defined in Equation 12. Hence, we normalize limb-darkened maps by π/f_u, so that the total flux for a uniform map with no limb darkening is unchanged.

3.3. Spherical harmonics rotations

As shown in section 2, computing the flux of the surface 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).

Rest frame   Sky-projected frame
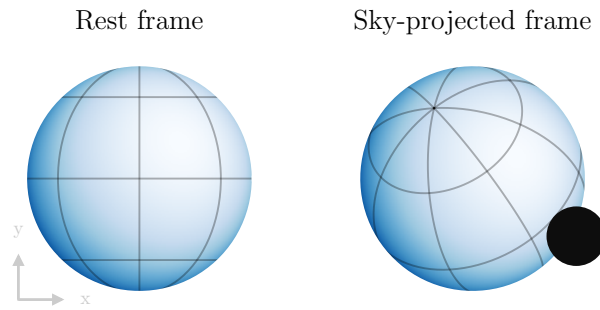
Figure 1. Rotation of a primary body's surface to place it in the sky-projected frame of the observer. ⟨/⟩

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 this framework. Figure 2 shows the 6 elementary rotations currently used in starry.
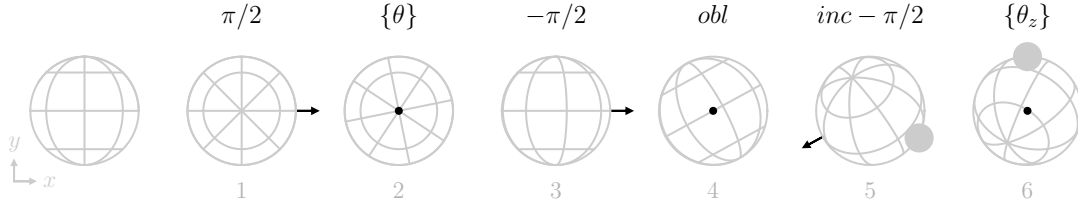
Figure 2. Decomposition of the full rotation of the surface into elementary rotations. Each schematic shows a sphere rotated from a previous orientation around an axis displayed as a black vector and an angle shown on top.

In this figure, the first 3 rotations have the effect to rotate 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 $\{t\}$, in order 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
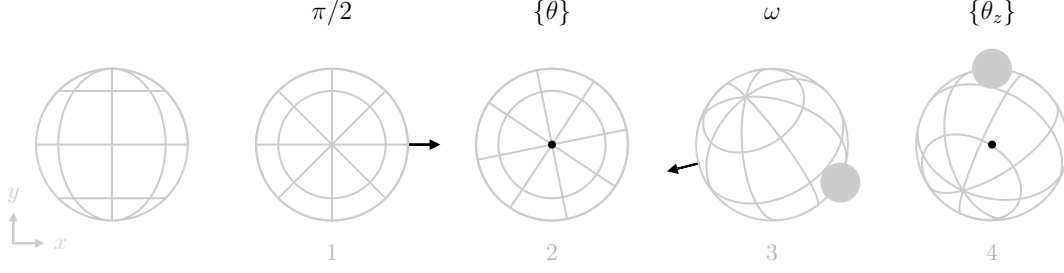
$$\mathbf{v} = \frac{1}{\sqrt{1 - \cos^2\left(\frac{inc}{2}\right)\cos^2\left(\frac{obl}{2}\right)}} \begin{pmatrix} \sin\left(\frac{inc}{2}\right)\cos\left(\frac{obl}{2}\right) \\ \sin\left(\frac{inc}{2}\right)\sin\left(\frac{obl}{2}\right) \\ -\cos\left(\frac{inc}{2}\right)\sin\left(\frac{obl}{2}\right) \end{pmatrix}, \tag{14}$$

and angle

$$\omega = 2\cos^{-1}\left(\cos\left(\frac{inc}{2}\right)\cos\left(\frac{obl}{2}\right)\right). \tag{15}$$

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. 📄

3.4. *Computation of the solution vectors*

We described in section 2.3 how the integrated intensity of a spherical body occulted by another can be computed using the solution vector $\mathbf{s}^\mathsf{T}$. Pál (2012) emphasizes that, knowing $\mathbf{G}_n$, each component s_n of the solution vector defined in Equation 10 can be decomposed into

$$s_n = \mathcal{Q}(\mathbf{G}_n) - \mathcal{P}(\mathbf{G}_n) \tag{16}$$

where $\mathcal{Q}(\mathbf{G}_n)$ and $\mathcal{P}(\mathbf{G}_n)$ are line integrals over the contours of the occulting body and the primary's surface, as shown in Figure 4.
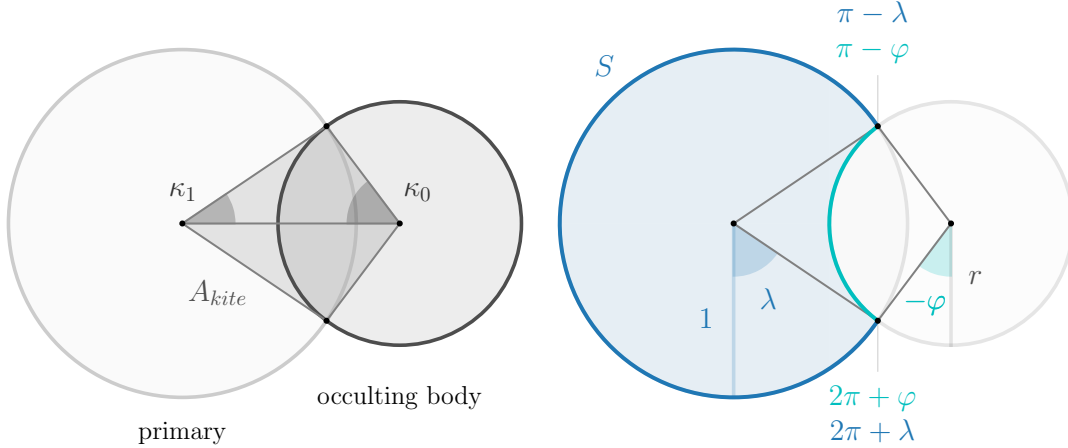
Figure 4. Geometric configuration and angles definitions for the two-body occultation. On the left figure, the larger light grey disk corresponds to the primary body being occulted by the darker secondary body. On the right figure, the area S over which the intensity is integrated is shown in light blue. By choosing an appropriate transform, the integration over S can be reduced to a line integral over the contour of the occulting body, which can be decomposed into the contour of the occulting body within the area of the primary body (in cyan) and the contour of the primary body everywhere else (in blue). 📄

In the general case, the $\mathcal{Q}$ integral (over the blue contour shown in Figure 4) takes a simple analytical form and can be computed using the recurrence relations from

Luger et al. (2019) (Equation D29). The case of a limb-darkened surface is even simpler as $\mathcal{Q}(\mathbf{G}_n) = 0$ for all $n > 0$. On the other hand, the $\mathcal{P}$ integral (over the cyan contour shown in Figure 4) is harder to evaluate and requires the use of recurrence relations and truncated series expansions, with a scheme that depends on the geometric configuration of the system (see Luger et al. 2019, Section D.2).

In order to translate this part of the starry implementation to JAX, we evaluate the $\mathcal{P}$ integrals numerically using Gauss-Legendre integrations, requiring the evaluation of the integrated functions only over a fixed number of points q.

3.4.1. *Polynomial limb darkening*

For a surface described by a polynomial limb darkening law of order n, the $\mathcal{P}$ integral is given in Agol et al. (2020) by

$$\mathcal{P}(\mathbf{G}_n) = \begin{cases} \pi - \kappa_1 - r^2 \kappa_0 + A_{kite} & n = 0 \\[2em] \text{see below} & n = 1 \\[2em] 2s_0 + 4\pi\eta - 2\pi & n = 2 \\[2em] 2r(4br)^{\frac{n}{2}} \displaystyle\int_{-\kappa_0/2}^{\kappa_0/2} (k^2 - \sin^2 \varphi')^{\frac{n}{2}} (r - b + 2b\sin^2 \varphi')\, \mathrm{d}\varphi' & n \geq 3, \end{cases}$$

where κ_0, κ_1 and A_{kite} are the angles and area shown in Figure 4 and computed following Agol et al. (2020) (Equations 31 and 32). For $n = 2$, η is defined in Agol et al. (2020) (Equation 53) and depends on the elliptic parameter

$$k = \frac{1 - r^2 - b^2 + 2br}{4br}, \tag{17}$$

also used in the case $n = 3$. For $n = 1$, we use Equation 31, 32 and 34 from Luger et al. (2019) to write

$$\mathcal{P}(\mathbf{G}_1) = \int_{-\kappa_0/2}^{\kappa_0/2} r(r - b\cos 2\varphi')\, \frac{1 - z^3}{3(1 - z^2)}\mathrm{d}\varphi' \quad \text{and} \quad \mathcal{Q}(\mathbf{G}_1) = \frac{2}{3}(\pi - \kappa_1), \tag{18}$$

where $1 - z^2 = b + r - 2\cos\varphi'$ [4].

---

[4] note that φ' is different from the angle φ shown in Figure 4 and was introduced through the change of variable $\varphi' = \frac{1}{2}(\varphi - \frac{3\pi}{2})$.

3.4.2. *Non-uniform surface*

In the general case, for a star with a non-radially symmetric surface intensity, n denotes the index of the spherical harmonics component such that

$$n = l^2 + l + m, \tag{19}$$

where l is the degree of the spherical harmonic and m its order. For each component, the expression of $\mathcal{P}(\mathbf{G}_n)$ depends on μ and ν defined as

$$\begin{aligned} \mu &\equiv l - m, \\ \nu &\equiv l + m. \end{aligned} \tag{20}$$

The $\mathcal{P}$ integral is then given by Equation D32 from Luger et al. (2019) as

$$\mathcal{P}(\mathbf{G}_n) = \begin{cases} 2(2r)^{l+2} \displaystyle\int_{-\kappa/2}^{\kappa/2} (s_{\varphi'}^2 - s_{\varphi'}^4)^{\frac{\mu+4}{4}} (\delta + s_{\varphi'}^2)^{\frac{\nu}{2}} \, \mathrm{d}\varphi' & \dfrac{\mu}{2} \text{ even} \\[2em] \mathcal{F} \displaystyle\int_{-\kappa/2}^{\kappa/2} (s_{\varphi'}^2 - s_{\varphi'}^4)^{\frac{l-2}{2}} (k^2 - s_{\varphi'}^2)^{\frac{3}{2}} (1 - 2s_{\varphi'}^2) \, \mathrm{d}\varphi' & \mu = 1, l \text{ even} \\[2em] \mathcal{F} \displaystyle\int_{-\kappa/2}^{\kappa/2} (s_{\varphi'}^2 - s_{\varphi'}^4)^{\frac{l-3}{2}} (\delta + s_{\varphi'}^2)(k^2 - s_{\varphi'}^2)^{\frac{3}{2}} (1 - 2s_{\varphi'}^2) \, \mathrm{d}\varphi' & \mu = 1, l \neq 1, l \text{ odd} \\[2em] 2\mathcal{F} \displaystyle\int_{-\kappa/2}^{\kappa/2} (s_{\varphi'}^2 - s_{\varphi'}^4)^{\frac{\mu-1}{4}} (\delta + s_{\varphi'}^2)^{\frac{\nu-1}{2}} (k^2 - s_{\varphi'}^2)^{\frac{3}{2}} \, \mathrm{d}\varphi' & \dfrac{\mu - 1}{2} \text{ even}, l \neq 1 \\[2em] \displaystyle\int_{-\kappa/2}^{\kappa/2} r(r - b\cos 2\varphi') \frac{1 - z^3}{3(1 - z^2)} \mathrm{d}\varphi' & \mu = 1, l = 1 \\[2em] 0 & \text{otherwise,} \end{cases}$$

where $s_{\varphi'} = \sin\varphi'$, $\kappa = \varphi + \frac{\pi}{2}$ (see Luger et al. 2019, Equation D31) and δ is defined for numerical stability as

$$\delta = \frac{b - r}{2r}. \tag{21}$$

Note how, unlike in the starry implementation, the term $\mu = 1$, $l = 1$ is derived exactly like in the polynomial limb darkening case, instead of analytically which would require the evaluation of elliptic integrals (see Luger et al. 2019, Section D.2.3).

Because of its numerical integration, the precision of the $\mathcal{P}$ integral is limited and depends on the order q of the Gauss-Legendre quadrature, which corresponds to the number of point for which the integrand is evaluated. To minimize this number,

we note that all integrands are symmetrical functions, except for $\mu = 1$ and $l = 1$. Hence, these integrals can be computed over the interval $\left[0, \frac{\kappa}{2}\right]$, instead of $\left[-\frac{\kappa}{2}, \frac{\kappa}{2}\right]$, and multiplied by 2. We also note that all terms for which $\frac{\mu}{2}$ is even reach machine precision for an integration order as low as $q = 20$, that we hold fix. With these optimizations, we benchmark the precision and speed of our final implementation for the solution vector and the total flux in section 4.

3.5. *Kepler's equation solver*

A core difference between our implementation and that of Luger et al. (2019) and Foreman-Mackey et al. (2021) lies in the method used to solve Kepler's equation

$$M = E - e \sin(E) \tag{22}$$

Indeed, iterative methods used in previous implementations (such as Raposo-Pulido & Peláez 2017 and Brandt et al. 2021) cannot be easily implemented in JAX and are not extensible to GPU or TPU acceleration. Instead, our implementation makes use of Markley (1995) which, despite its simplicity, leads to state-of-the-art precision and runtime performance.

In addition, although JAX features automatic differentiation, we find that explicitly implementing the derivatives of the Kepler equation solver is more efficient. To do so, we use the implicit function theorem to derive the *Jacobian-vector product* required by JAX to specify custom derivatives. This is done by differentiating Kepler's equation as

$$dM = dE \left(1 - e \cos E\right) - de \, \sin E, \tag{23}$$

which leads to

$$dE = \frac{1}{1 - e \cos E} \, dM + \frac{\sin E}{1 - e \cos E} \, de, \tag{24}$$

so that point estimates for the derivatives of E can be easily evaluated. We find that this method is 6 times faster than using JAX's automatic differentiation[5].

3.6. *Light curve functions and transformations*

Using all the optimizations described in the previous sections, the jaxoplanet Python package provides code to compute light curves of mutually occulting celestial bodies. Our implementation is written in JAX and contains fully vectorized light curve functions that can be compiled and run on several hardware accelerator.

- The `jaxoplanet.light_curves.limbdark` module contains functions to compute occultation light curves of limb darkened stars described by polynomial limb darkening laws (Agol et al. 2020).

[5] Test performed on an M2 Max Macbook.

- The `jaxoplanet.light_curves.emission` module contains functions to compute light curves of emissive limb darkened bodies. This module use the same formalism as the `limbdark` module, but allows for secondary ecclipses to be computed.

- The `jaxoplanet.starry.light_curves` module contains functions to compute occultation light curve of bodies with non-uniform surfaces (Luger et al. 2019).

These functions can optionally be transformed using special wrappers provided in the `jaxoplanet.light_curves.transforms` module. For example, the `integrate` transform enables the computation of light curves observed at finite exposure time, using the methods described in Kipping (2010)[6]. Additionlly, the `interpolate` transform can be used to precompute light curves on a grid near a transit, and then interpolate those computations to the required phases when computing the full model, leading to substantial speed ups when several transits are present in the data.

By providing the community with an alternative to starry entirely written in Python, we hope to foster the development of new tools and methods to model light curves from stellar and planetary systems. In the next section, we evaluate the performance of our implementation and compare it with the original starry implementation.

4. PERFORMANCE

Although 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 the solution vector $\mathbf{s}^\mathsf{T}$, computed using numerical series expansions in starry (Luger et al. 2019, section D.2.3) and numerical integration in jaxoplanet (see section 3.4). Other expressions, such as the Wigner-D matrices used in the rotation of the spherical harmonics basis, are computed using reccurence 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.

4.1. *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 library[7]. The precision of the overall flux f can only be understood by considering the precision of the terms involved in Equation 11: the

---

[6] See tutorial at TODO
[7] https://mpmath.org/

solution vector $\mathbf{s}^\mathsf{T}$ ($\mathbf{r}^\mathsf{T}$ if no occultation), the change of basis matrices $\mathbf{A_1}$ and $\mathbf{A_2}$, and the Wigner-D rotation matrix $\mathbf{R}$. Although we show in Figure 10 that the precision of the starry and jaxoplanet implentations slightly differ for many of these terms, we focus this section on the solution vector $\mathbf{s}^\mathsf{T}$ and the overall flux f.

Figure 5. Error of the solution vector $\mathbf{s}^\mathsf{T}$ and the overall flux f for all terms of the spherical harmonics basis up to degree $l_{max} = 20$. Errors are computed against arbitrary-precision computations of the solution vector and the flux for a small ($r = 0.01$) and a large ($r = 100$) occultor transiting the star accross numerically-sensitive values of impact parameters b. For each basis component i, all coefficients are set to zero except $y_i = 1$ 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 occultor (light dark) and the primary surface (light gray) for different values of b.

Assuming that we perform the numerical integration described in section 3.4 at a relatively high order $q = 500$, Figure 5 shows that our reimplentation of starry reaches relative errors lower than 1 part per billion for $\mathbf{s}^\mathsf{T}$ and the overall flux f, for almost all

degrees up to $l_{max} = 20$ and considering both a small ($r = 0.01$) and a large ($r = 100$) occultor transiting the star accross numerically-sensitive values of impact parameters b. For reference, Figure 11 shows comparable errors on the same quantities computed with the C++ implentation of starry.

In addition, Figure 6 shows the evolution of the error on f for increasing degrees of the spherical harmonics for starry and jaxoplanet.
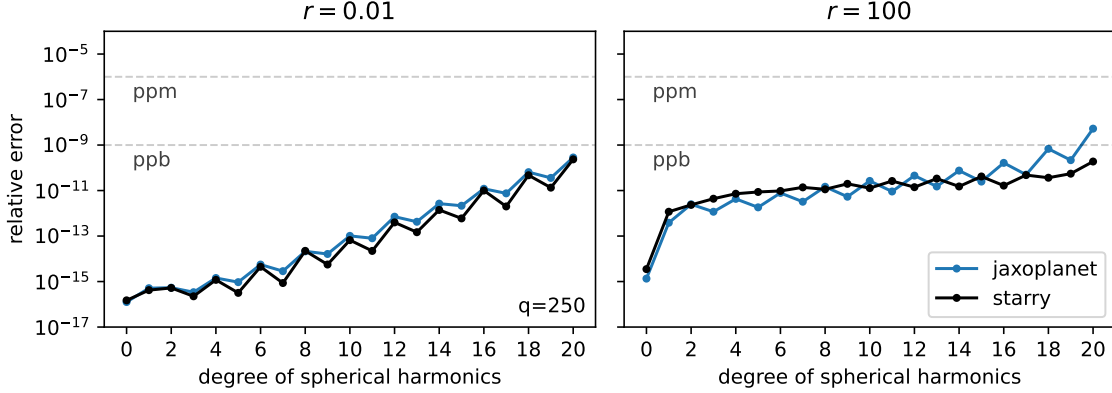

Figure 6. Maximum errors in the overall flux f for all degrees of the spherical harmonics components. Errors shown in this figure correspond to the maximum values of the scaled errors shown in Figure 5 over the range of values b, but computed with $q = 250$.
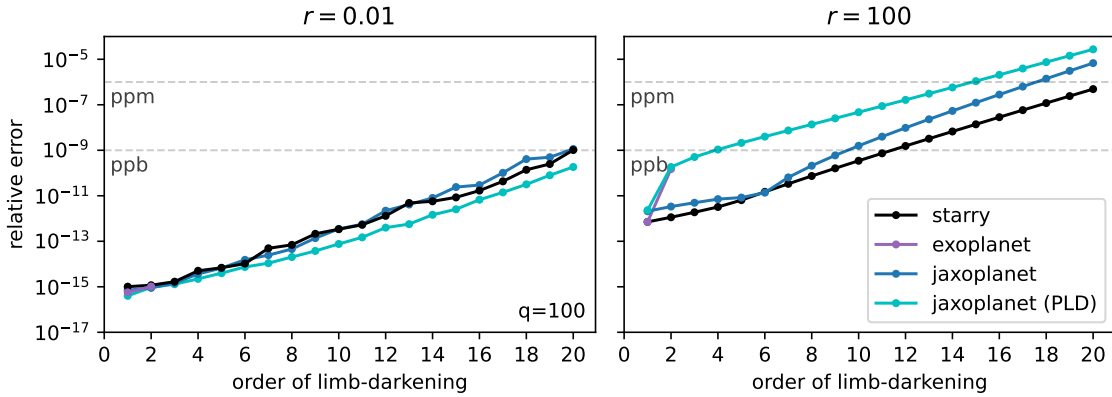
Figure 7. Maximum errors in the overall flux f of a star occulted by an opaque body for increasing orders of its polynomial limb darkening law. Errors shown in this figure correspond to the maximum values of the scaled errors shown in Figure 5 over the range of values b, but computed with $q = 100$. For each order n, all limb darkening coefficients are set to zero except the n-th coefficient $u_n = 1$. The purple points show the exoplanet errors on the flux f, limited to the linear and quadratic limb darkening laws. In order to benchmark our code (denoted jaxoplanet (PLD) and shown in cyan) for higher order laws, we show the error of the flux computed with the starry (black) and jaxoplanet (blue) codes, using the formalism of Luger et al. (2019) instead Agol et al. (2020).

In Figure 7, we compare the precision of our limb darkened light curve model with that of the exoplanet Python package (which uses the same formalism as Agol et al. (2020) but limited to linear and quadratic limb darkening). The larger errors observed in this figure are due to the transformation of the polynomial limb darkening coefficients to the spherical harmonics coefficients. For example, a set of unitary polynomial limb darkening coefficients of order 20 leads to spherical harmonics coefficients as large as 10^6. We note that these errors scale with the values of $\mathbf{u}$.

As described in section 3.4, we compute occultation light curves using the Gauss Legendre quadrature to approximate the $\mathcal{P}$ integral involved in the expression of the solution vector $\mathbf{s}^\mathsf{T}$. 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 $\mathcal{P}$ (see Figure 8, or Figure 12 in the Appendix to see the effect for different spherical harmonics degrees l). Hence, users must carefully set the order parameter q to balance precision and computational speed (see section 4.2).
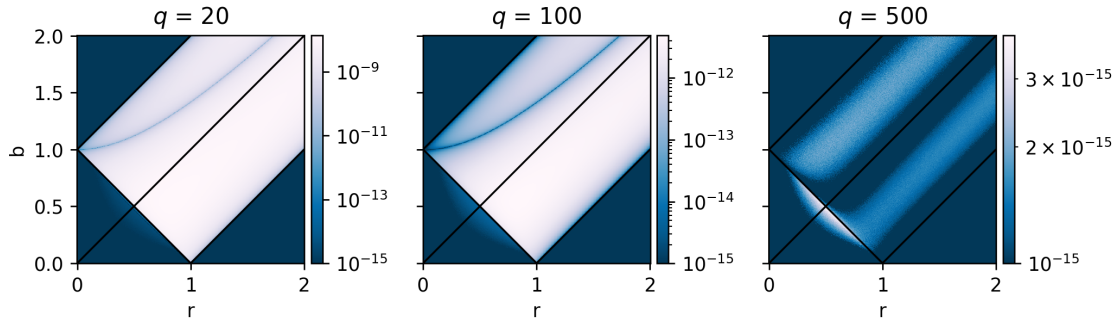
Figure 8. relative numerical error in the flux f of a linearly limb-darkened star (degree $l{=}1$) transited by an opaque companion over the (r,b) parameter space. The error is evaluated for different orders q of the Gauss-Legendre quadrature and computed against values of f obtained with $q = 800$. Note that Figure 5 was produced using $q = 500$ (right plot) for which $\mathbf{s}$ reaches machine precision.

With the results of this section, we validate the precision of jaxoplanet and show it is on per with the legacy C++ starry implentation[8] described in Luger et al. (2019).

4.2. *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 curves, we also compare our implementation with

---

[8] version 1.2.0 (Luger et al. 2021)

the one provided by the **exoplanet** Python package.

While performing numerical integration may be seen as a downside in our implementation, adapting the value of q to the precision required for specific applications allows users to optimize the computation time of the model to their advantage. To demonstrate this point, we first consider the transmission spectroscopy of TRAPPIST-1 b performed with the JWST/NIRISS instrument (GO-2589, Lim et al. 2023). These observations reached an estimated precision of 89 ppm for a planet with a relative radius of the order $r \sim 0.1$. In this context, numerical integration with an order as low as $q = 10$ guarantees that our model reaches a precision of 10^{-16}, an execution time on par with **exoplanet** on CPU and up to an order of magnitude faster on GPU for large datasets (see left plot of Figure 9). The second example we consider is the secondary ecclipse mapping of HD 189733 b ($r \sim 0.3$) using Spitzer's 8 micron channel (Majeau et al. 2012) obtained at a precision of about 1 ppt. For a spherical harmonics map of degree $l_{max} = 3$, $q = 10$ guarantees a model with a precision of 10^{-10}, an execution time one order of magnitude faster than **starry** on CPU and up to two orders of magnitude faster on GPU for large datasets (see right plot of Figure 9).
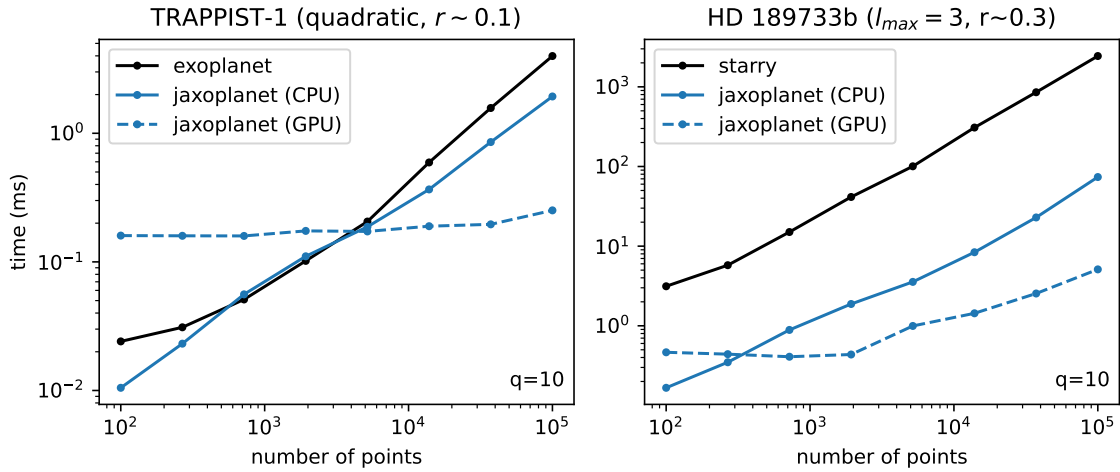
Figure 9. Evaluation time versus number of points for different light curve models used in realistic applications. On the left, we evaluate the time required by **jaxoplanet** and **exoplanet** to compute the quadratically limb darkened transit light curve of TRAPPIST-1 b observed with the JWST/NIRISS instrument (for which $q = 10$ yields machine precision). On the right we evaluate the time required by **jaxoplanet** and **starry** to compute the secondary eclipse light curve of HD 189733 b whose surface is modeled using spherical harmonics of degree $l_{max} = 3$ (with $q = 10$ leading to a precision of 10^{-10}). On CPU, our benchmark is done on the single core of an Apple M2 max chip, while on GPU we use the single core of an NVIDIA Tesla V100 processor.

In general, we show that models computed at low values of q, which are very fast, match the precision of measurements from modern instruments. Although this might not be required, Appendix B shows reference speed benchmarks for models evaluated at state-of-the-art precisions.

In order to help users set the value of q according to their required precision, the jaxoplanet Python package provide helper functions to estimate the precision of the light curve model given the relative radius r, the spherical harmonics degree l_{max} (if general non-uniform maps are required), and the order of the polynomial limb darkening law[9].

5. CONCLUSION

In this paper we presented jaxoplanet, a new implementation of the analytical light curve model *starry* developed using the machine learning library JAX. We showed that our implementation is on par with the legacy C++ starry implementation, and that it offers a significant speed advantage on GPUs for large datasets, enabled by JAX's just-in-time compilation and hardware acceleration. Our code can be used in association with the large ecosystem of probabilistic and sampling libraries developed in JAX, and will facilitate the development of tools and pipeline for the modeling of TESS and JWST observations, but also future missions such as PLATO (Rauer et al. 2014) and ARIEL (Tinetti et al. 2018).

As the core of our model is developed in Python, we expect jaxoplanet to serve as a basis for the implementation of more complex light curve models, such as light curves of Doppler maps (Luger et al. 2021), reflected maps (Luger et al. 2022), or non-spherical bodies maps (Dholakia et al. 2024). Eventually, jaxoplanet will take over starry in acting as a foundation to explore and develop analytical methods to numerically reconstruct the surface of exoplanets and stars, deepening our understanding of exoplanetary systems and their worlds.

The jaxoplanet code is open source under the MIT License and is hosted at https://github.com/exoplanet-dev/jaxoplanet, with documentation and tutorials hosted at https://jax.exoplanet.codes. A permanent version of the code used to generate the figures and results in this paper is archived at https://doi.org/10.5281/zenodo.1312286.

Software: JAX (Bradbury et al. 2018), starry (Luger et al. 2019), exoplanet (Foreman-Mackey et al. 2021), snakemake (Sna 2021), numpy (Harris et al. 2020), scipy (Virtanen

---

[9] See tutorial at TODO

et al. 2020), s2fft (Price & McEwen 2023), matplotlib (Hunter 2007), mpmath (mpmath development team 2023), equinox (Kidger & Garcia 2021), pint

REFERENCES

2021, F1000Research, 10, doi: 10.12688/f1000research.29032.1

Agol, E., Luger, R., & Foreman-Mackey, D. 2020, AJ, 159, 123, doi: 10.3847/1538-3881/ab4fee

Alderson, L., Wakeford, H. R., Alam, M. K., et al. 2023, Nature, 614, 664, doi: 10.1038/s41586-022-05591-3

Almenara, J. M., Bonfils, X., Forveille, T., et al. 2022, A&A, 667, L11, doi: 10.1051/0004-6361/202244791

Bartolić, F., Luger, R., Foreman-Mackey, D., Howell, R. R., & Rathbun, J. A. 2022, PSJ, 3, 67, doi: 10.3847/PSJ/ac2a3e

Bell, T. J., Welbanks, L., Schlawin, E., et al. 2023, Nature, 623, 709, doi: 10.1038/s41586-023-06687-0

Betancourt, M. 2017, arXiv e-prints, arXiv:1701.02434, doi: 10.48550/arXiv.1701.02434

Bradbury, J., Frostig, R., Hawkins, P., et al. 2018, JAX: composable transformations of Python+NumPy programs, 0.3.13. http://github.com/google/jax

Brandt, T. D., Dupuy, T. J., Li, Y., et al. 2021, AJ, 162, 186, doi: 10.3847/1538-3881/ac042e

Charbonneau, D., Brown, T. M., Latham, D. W., & Mayor, M. 2000, ApJL, 529, L45, doi: 10.1086/312457

Dawson, R. I., Huang, C. X., Brahm, R., et al. 2021, AJ, 161, 161, doi: 10.3847/1538-3881/abd8d0

Dholakia, S., Dholakia, S., & Pope, B. J. S. 2024, arXiv e-prints, arXiv:2410.03449, doi: 10.48550/arXiv.2410.03449

Drish, W. F., Harmon, R., Marcialis, R. L., & Wild, W. J. 1995, Icarus, 113, 360, doi: https://doi.org/10.1006/icar.1995.1028

Foreman-Mackey, D., Luger, R., Agol, E., et al. 2021, arXiv e-prints, arXiv:2105.01994. https://arxiv.org/abs/2105.01994

Harris, C. R., Millman, K. J., van der Walt, S. J., et al. 2020, Nature, 585, 357, doi: 10.1038/s41586-020-2649-2

Hunter, J. D. 2007, Computing in Science & Engineering, 9, 90, doi: 10.1109/MCSE.2007.55

Kidger, P., & Garcia, C. 2021, Differentiable Programming workshop at Neural Information Processing Systems 2021

Kipping, D. M. 2010, MNRAS, 408, 1758, doi: 10.1111/j.1365-2966.2010.17242.x

Kostov, V. B., Schlieder, J. E., Barclay, T., et al. 2019, AJ, 158, 32, doi: 10.3847/1538-3881/ab2459

Lim, O., Benneke, B., Doyon, R., et al. 2023, ApJL, 955, L22, doi: 10.3847/2041-8213/acf7c4

Luger, R., Agol, E., Bartolić, F., & Foreman-Mackey, D. 2022, AJ, 164, 4, doi: 10.3847/1538-3881/ac4017

Luger, R., Agol, E., Foreman-Mackey, D., et al. 2019, The Astronomical Journal, 157, 64, doi: 10.3847/1538-3881/aae8e5

Luger, R., Bedell, M., Foreman-Mackey, D., et al. 2021, arXiv e-prints, arXiv:2110.06271, doi: 10.48550/arXiv.2110.06271

Luger, R., Agol, E., Foreman-Mackey, D., et al. 2021, rodluger/starry: v1.2.0, v1.2.0, Zenodo, doi: 10.5281/zenodo.5567781

Majeau, C., Agol, E., & Cowan, N. B. 2012, ApJL, 747, L20, doi: 10.1088/2041-8205/747/2/L20

Markley, F. L. 1995, Celestial Mechanics and Dynamical Astronomy, 63, 101, doi: 10.1007/BF00691917

Mayor, M., & Queloz, D. 1995, Nature, 378, 355, doi: 10.1038/378355a0

mpmath development team, T. 2023, mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 1.3.0)

Pál, A. 2012, MNRAS, 420, 1630, doi: 10.1111/j.1365-2966.2011.20151.x

Price, M. A., & McEwen, J. D. 2023, Journal of Computational Physics, submitted

Raposo-Pulido, V., & Peláez, J. 2017, MNRAS, 467, 1702, doi: 10.1093/mnras/stx138

Rauer, H., Catala, C., Aerts, C., et al. 2014, Experimental Astronomy, 38, 249, doi: 10.1007/s10686-014-9383-4

Ricker, G. R., Winn, J. N., Vanderspek, R., et al. 2015, Journal of Astronomical Telescopes, Instruments, and Systems, 1, 014003, doi: 10.1117/1.JATIS.1.1.014003

Risbo, T. 1996, Journal of Geodesy, 70, 383, doi: 10.1007/BF01090814

Stern, S. A., Buie, M. W., & Trafton, L. M. 1997, AJ, 113, 827, doi: 10.1086/118304

Stern, S. A., Bagenal, F., Ennico, K., et al. 2015, Science, 350, aad1815, doi: 10.1126/science.aad1815

Tinetti, G., Drossart, P., Eccleston, P., et al. 2018, Experimental Astronomy, 46, 135, doi: 10.1007/s10686-018-9598-x

Virtanen, P., Gommers, R., Oliphant, T. E., et al. 2020, Nature Methods, 17, 261, doi: 10.1038/s41592-019-0686-2
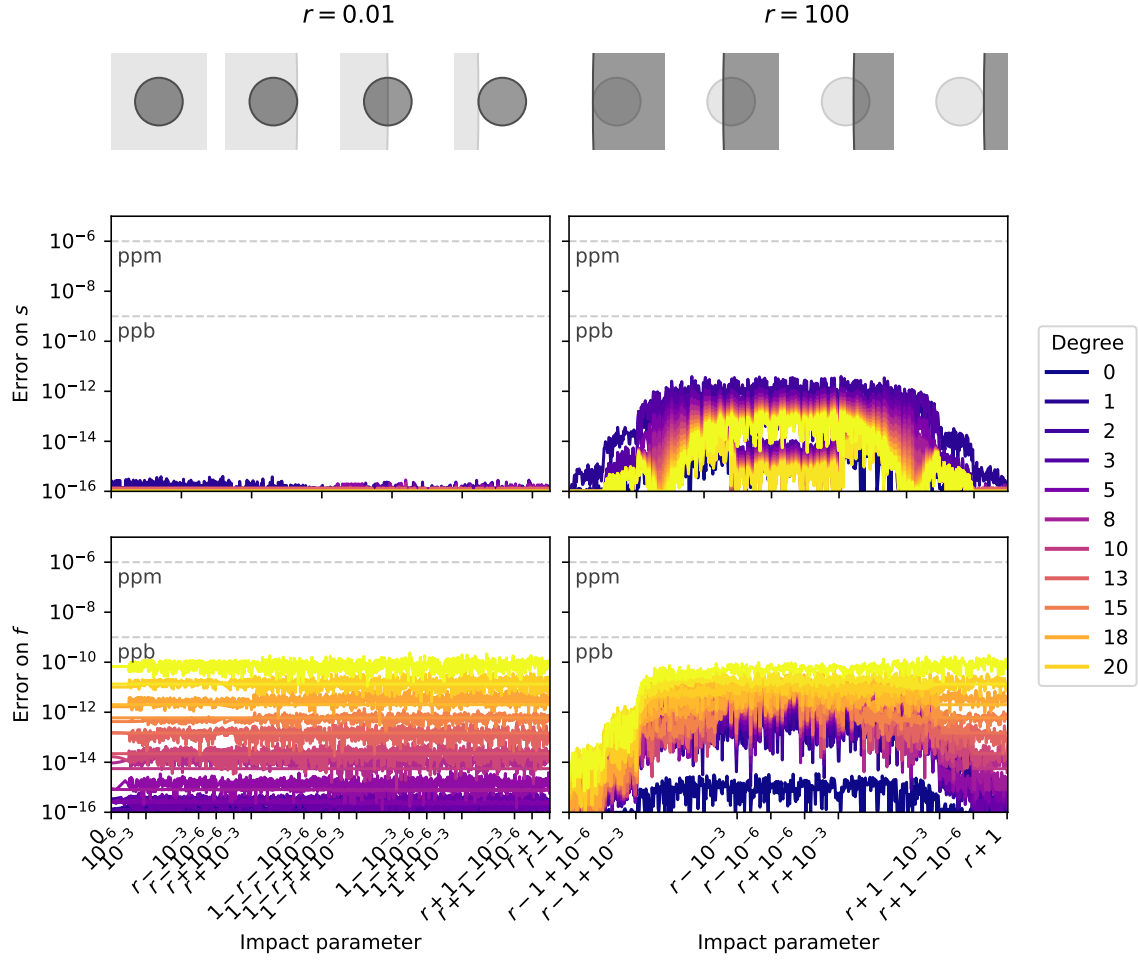
APPENDIX

A. ADDITIONAL PRECISION BENCHMARKS
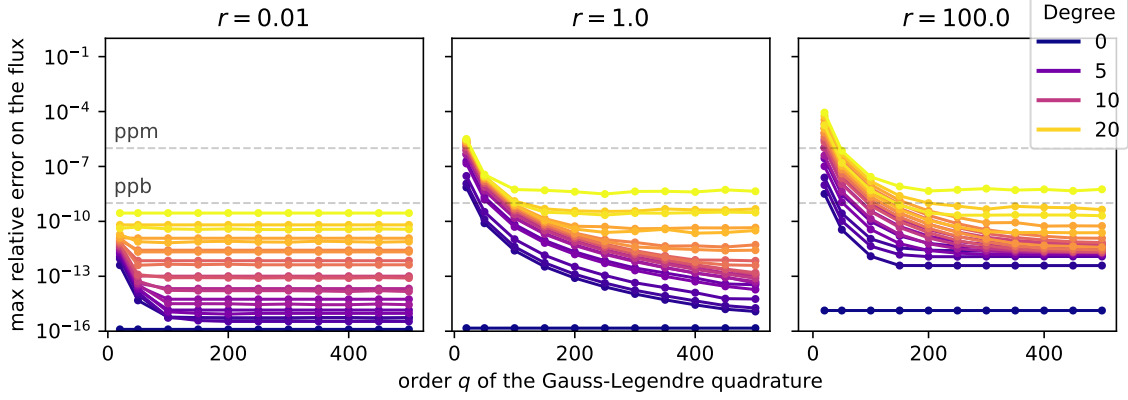
This section contains additional figures showing the precision of the jaxoplanet implementation and its comparison to the C++ implementation of starry.
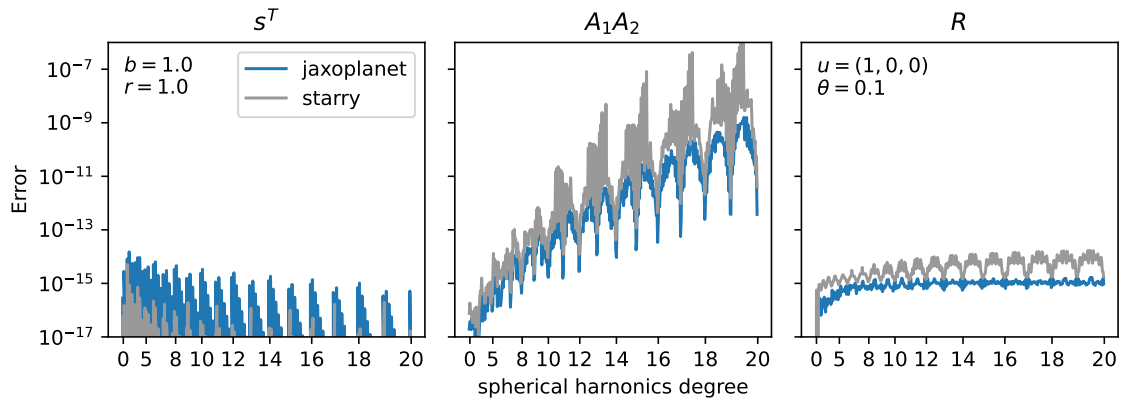
**Figure 10.** Typical errors in the matrices involved in the computation of the flux from Equation 11. 〈/〉

Figure 11. Same as Figure 5 but with $\mathbf{s}^\mathsf{T}$ and f computed with the C++ implementation of starry. ⟨/⟩

Figure 12. Maximum errors in the overall flux f for all degrees of the spherical harmonics components for different occultor radii and orders q of the Gauss-Legendre quadrature used to compute the $\mathcal{P}$ integral (see section 3.4). Errors shown in this figure correspond to the maximum values of the scaled errors shown in Figure 5, maxima taken over the range of values b.

B. ADDITIONAL SPEED BENCHMARKS

Unlike section 4.2, this section focuses on speed benchmarks at state-of-the-art precision. In Figure 13, we show the evolution of the time required to compute the occultation light curve of a limb darkened star and that of a non-uniform star described by spherical harmonics of degree $l_{max} = 20$, depending on the order q of the numerical integration. On CPU, our benchmark is done on the single core of an Apple M2 max chip, while on GPU we use the single core of an NVIDIA Tesla V100 processor.
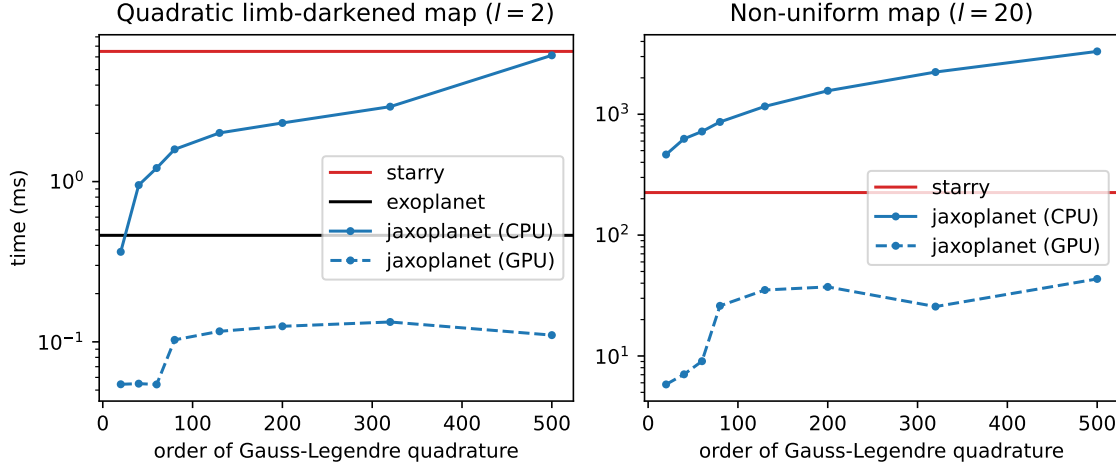
Figure 13. Computation time of the occultation light curve of a limb-darkened star (left) and a non-uniform star described by spherical harmonics up to order $l_{max} = 20$ (right) depending on the order q of the Gauss-Legendre quadrature used to compute the $\mathcal{P}$ integral numerically (see section 3.4). The processing time reported for exoplanet and starry is independent of q 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 occultor radius $r = 0.1$.

Finally, Figure 14 shows the evaluation time of jaxoplanet as a function of the number of points in the light curve compared to exoplanet, for a quadratically limb darkened star, and starry, for a surface map of degree $l_{max} = 20$.
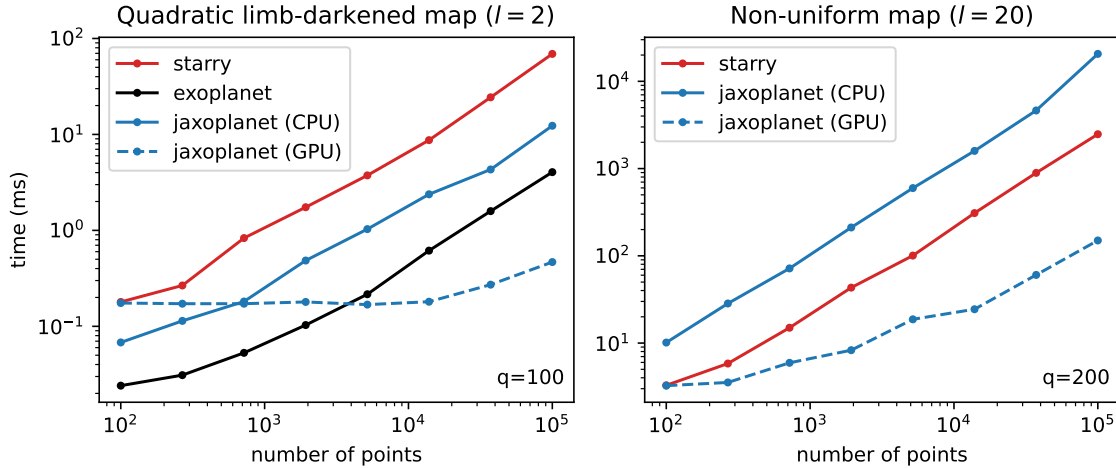
Figure 14. Computation time of the occultation light curve of a limb-darkened star (left) and a non-uniform star described by spherical harmonics up to order $l_{max} = 20$ (right) depending on the number of points in the light curve. Light curves are computed at $q = 200$ and an occultor radius $r = 0.1$.

These benchmarks are produced for values of q which guarantee that our models reach the same precision as state-of-the-art codes. As discussed in section 4.2, jaxoplanet can

be further optimized by reducing the order q of numerical integration while keeping the precision required for specific applications.

Table 1. Symbols used in this paper

Symbol	Definition	Reference
$\mathbf{A_1}$	change of basis matrix from Y_{lm} to polynomials	Equation 4
$\mathbf{A_2}$	change of basis matrix from polynomials to Green's basis	section 2.3
A_{kite}	Kite-shaped area b/w centers and contact points	Figure 4
b	impact parameter of the occultor	Equation 17
f_u	flux of a purely limb darkened star	Equation 13
$\tilde{\mathbf{g}}$	Green's basis vector	section 2.3
$\tilde{g}_n$	n-th component of the Green's basis vector	section 2.3
$\mathbf{G}_n$	Green's function	section 2.3
$I(x,y)$	intensity of the body's surface at coordinates (x,y)	Equation 1
l	degree of the Y_{lm}	section 2.1
k	Elliptic integral parameter	Equation 17
l_{max}	(maximum) degree of a Y_{lm} map	section 2.1
m	order of the Y_{lm}	section 2.1
n	Y_{lm} index in $\tilde{\mathbf{y}}$	Equation 19
$\mathcal{P}$	Primitive integral along perimiter of occultor	Equation 16
$\mathbf{p}$	polynomial coefficients describing the the surface	Equation 4
$\tilde{\mathbf{p}}$	polynomial basis vector	Equation 4
q	order of the Gauss-Legendre quadrature	Equation 4
$\mathcal{Q}$	Primitive integral along perimiter of occulted body	Equation 16
r	occultor radius in units of occulted body's radius	Equation 17
$\mathbf{R}$	Rotation matrix of the Y_{lm} basis	Equation 2
$\mathbf{R}'$	Rotation matrix to align the occultor with the y-axis	section 2.3
s_n	n-th component of the solution vector $\mathbf{s}^\mathsf{T}$	Equation 10
$\mathbf{s}^\mathsf{T}$	solution vector	section 2.3
$\mathbf{u}$	limb darkening coefficients	Equation 12
$\mathbf{U}$	change of basis matrix from Y_{lm} to limb darkening	Equation 12
$\mathbf{v}$	vector of compound rotation to sky-projected orientation	Equation 14
x	x cartesian coordinate	Equation 1
y	y cartesian coordinate	Equation 1
$\mathbf{y}$	Y_{lm} coefficients describing the surface	Equation 1
$\tilde{\mathbf{y}}$	Y_{lm} basis vector	Equation 1
Y_{lm}	Spherical harmonics basis function	section 2.1
$\mathbf{y}_u$	limb darkening coefficients of a purely limb darkened star	Equation 12
z	$z = \sqrt{1-x^2-y^2}$ cartesian coordinate	section 2
δ	numerical stability parameter	Equation 21
κ	$\varphi + \frac{\pi}{2}$	section 3.4.2
κ_0	Angular position of body's disks intersection	Figure 4
κ_1	Angular position of body's disks intersection	Figure 4
λ	angular position of the occultor contact points	Figure 4
μ	$l - m$	Equation 20

Table 1 – continued from previous page

Symbol	Definition	Reference
ν	$l + m$	Equation 20
φ	angular position of the occultor contact points	Figure 4
φ'	change of variable for φ	section 3.4.2
ω	Rotation angle of the combined rotation	Equation 15