`LiGrad`: User Manual

Eleftherios Voulimiotis

September 16, 2025

Abstract

This manual documents the function `grav_dark_transit_model` from the Python code `LiGrad`, created in the context of my bachelor thesis "Transit Model for Gravity-Darkened Oblate Stars" (Voulimiotis, 2025). In this document, we first explain the logical steps that the code follows to generate the transit light-curves of planets orbiting oblate-rotating stars with gravity darkening. Next, we describe the code processes and functions that execute these logical steps, along with the main user interface (expected units for each input parameter, recommended usage, example script).

Contents

1 Installation

1.1 Installation instructions and dependencies

1.2 Python dependencies

The package depends on the following Python libraries:

- `numpy` (Harris et al., 2020)
- `scipy` (Virtanen et al., 2020)
- `pylightcurve` (Tsiaras et al., 2016)
- `astropy` (Astropy Collaboration et al., 2022)

Additional libraries used for some example scripts in GitHub are `matplotlib` (Hunter, 2007), `exoclock` (Kokori et al., 2022), `emcee` (Foreman-Mackey et al., 2013) and `astroquery` (Ginsburg et al., 2019).

1.3 Install from pip (PyPI)

Published to PyPI:

```
pip install ligrad
```

1.4 Clone from GitHub

Clone directly from GitHub:

```
git clone https://github.com/evoulimiotis/ligrad.git
cd ligrad
python setup.py install
```

2 Code Description

The main objective of my thesis was the development of new code that is able to fully simulate the transit light-curves of planets orbiting oblate stars with gravity darkening. In this section we will talk about the `LiGrad` code (Transit Light-Curves for Gravity-Darkened Oblate Stars) which was constructed for this phenomenon, and how it works. The code was made entirely in Python and requires certain libraries in order to run properly (dependencies).

2.1 Main Inputs and Function

The main function has the following form:

```
    grav_dark_transit_model(t_vals, orbital_period, st_mass,
                      st_mean_radius, st_mean_temperature, beta, lamda,
                      i_s, omega, u1, u2, u3, u4, e, i_0, omega_p, t_p,
               rp_rs, obs_wavelength=800e-9, integration_grid_size=45)
```

Here, `t_vals` is an array of time data (in days), `period` is the orbital period (days), M, $R_{\rm mean}$, $T_{\rm mean}$ are the stellar mass, mean radius and mean effective temperature relative to the solar values (for example if the stellar mass-radius is $1.7M_{\odot}$-$1.7R_{\odot}$, then the given argument should be 1.7 in both cases). The gravity-darkening exponent is β. The code also requires the sky-projected obliquity λ and the stellar inclination i_s, the rotation factor ω (extracted from catalog stellar rotational velocities $u_{eq}\sin i_s$) and the four Claret limb-darkening coefficients $u_1, \ldots, u_4$. Finally, the orbital elements (eccentricity e, orbital inclination i_0, argument of periapsis ω_p, and time of periastron t_p) are inputs. The planet size is given as $R_p/R_\star$, which is the radius normalized to the star's mean projected radius.

In short, the function terms and their units are as follows:

```
t_vals,               # ndarray: times at which to compute flux (days)
orbital_period,       # float: orbital period (days)
st_mass,              # float: stellar mass (in solar masses)
st_mean_radius,       # float: stellar radius (in solar radii)
st_mean_temperature,  # float: in units of 10000 K
beta,                 # float: gravity-darkening exponent (dimensionless)
lamda,                # float: spin-orbit angle (degrees)
i_s,                  # float: stellar spin inclination (degrees)
omega,                # float: stellar rotation parameter (dimensionless)
u1,u2,u3,u4,          # floats: Claret 4-term limb-darkening coefficients
e,                    # float: orbital eccentricity (0 to 1)
i_0,                  # float: orbital inclination (degrees)
omega_p,              # float: argument of periastron (degrees)
t_p,                  # float: time of periastron / reference epoch (in days)
rp_rs,                # float: planet radius relative to effective stellar radius
obs_wavelength=800e-9,# float: observation wavelength (meters)
planet_focused=True,  # bool: faster integration when True
integration_grid_size=45  # int: integration grid resolution
```

- **t_vals, orbital_period, t_p**: **days**. Use the same time system consistently. If you want to work in BJD_TDB or similar, ensure all times are in that system.
- **st_mass**: **solar masses** (the code multiplies input by $M_\odot$).
- **st_mean_radius**: **solar radii** (the code multiplies input by $R_\odot$).
- **st_mean_temperature**: **input in units of 10,000 K**.
  Reason: normalization for data fitting. The code multiplies the `st_mean_temperature` argument by 10000 internally. Example: for a star with 6000 K, pass `st_mean_temperature = 0.6`.
- **lamda**, **i_s**, **i_0**, **omega_p**: **degrees**. (Within the code `lamda` and `i_s` are converted to radians before geometric rotations.)
- **omega**: dimensionless rotation factor. It is used to estimate the polar-equatorial radius through $R_p = R_{\mathrm{eq}} \cdot \frac{2}{2+\omega^2}$ inside the code. The values are in the range $0 \leq \omega < 1$.
- **u1...u4**: dimensionless 4-term Claret limb-darkening coefficients (Claret law).
- **rp_rs**: ratio of planetary radius to effective stellar radius (dimensionless). The code computes a physical planet radius in the projected stellar coordinate units as `Rp_phys = rp_rs * R_eff`.
- **obs_wavelength**: SI meters, with default 800 nm = 800e-9, as an average value of the TESS (Transiting Exoplanet Survey Satellite) bandpass.
- **integration_grid_size**: integer controlling integration points used in the planet-focused blocked-flux integration (larger ⇒ better accuracy but slower).

2.2 Spheroid and Coordinates

The star is modeled as an oblate spheroid. The code uses an equatorial radius R_{eq} (set to the mean radius in the function call) and computes a polar radius R_p through the dimensionless rotation parameter ω. We create the spheroidal surface as a grid of points given in spherical coordinates and then convert them to 3D Cartesian coordinates.

After generating the surface points, the code rotates them to the observer frame using rotation matrices. The input angles i_s and λ are essentially converted into two rotation angles, and these rotations are applied so that the star is seen in the correct sky-projected orientation.

2.3 Point Visibillity

To determine which surface points "face" the observer and which are on the invisible side, the code calculates the gradient $\vec{\eta}$ of each point and then checks the dot product of $\vec{\eta}$ with the unit vector of the x-axis $\hat{i}$. If $\vec{\eta} \cdot \hat{i} > 0$ then the point is considered visible and we keep it, removing ones with $\vec{\eta} \cdot \hat{i} < 0$.

2.4 Temperature, Gravity Darkening and Intensity

In the code, we compute the effective local gravity at each surface point. Then the local temperature is set by the power law:

$$T(\theta) = T_{\mathrm{mean}} \left(\frac{g(\theta)}{g_{\mathrm{mean}}} \right)^{\beta} ,$$

The stellar intensity is calculated from the Planck function. This intensity is later multiplied by the limb-darkening factor which will be described next.

For simplicity, the code uses a single wavelength at $800nm$ instead of integrating over a real bandpass (for example the TESS telescope has a bandpass of $600 - 1000nm$), so the resulting fluxes are monochromatic approximations to what an instrument would see.

2.5 Stellar Edge Radii

The general sky-projected shape of the spheroid is an ellipse. This indicates that the radius of the projected object is not constant. So, we must find a way to calculate its outer radius, in order to find the limb darkening. To accomplish this, the code finds the convex hull of the projected points and computes the radius as function of a polar angle around the center. Then, it sorts points by angle and interpolates the radial distance. This lets the code test if an arbitrary point on the (y, z) plane is inside the stellar disk by comparing its distance to the interpolated edge radius at that angle.

2.6 Limb Darkening

The code finds the projected center and radius of the star (using the previous method). Then, it defines the normalized radial coordinate ρ and $\mu = \sqrt{1-\rho^2}$ and applies the 4-parameter Claret limb-darkening law:

$$I_{\mathrm{LD}}(\mu) = 1 - u_1(1-\mu) - u_2(1-\mu)^2 - u_3(1-\mu)^3 - u_4(1-\mu)^4 \ ,$$

setting the intensity to zero outside the projected disk. Multiplying the Planck intensity by this limb-darkening factor, gives us the final intensity assigned to each visible surface point.

2.7 Total Unocculted Flux

The total flux from the visible stellar disk is computed once per parameter set and cached. To achieve this, the code takes the visible projected points (y, z), uses Delaunay triangulation on those 2D points, calculates the area of each triangle, and then approximates the surface integral by summing the area $\times$ the average intensity at the triangle's vertices. In other words, if A_i is the area of one triangle and $\bar{I}_i$ its mean intensity, then the flux is approximated as follows:

$$F_0 = \sum_i A_i \bar{I}_i$$

The accuracy of this calculation depends on the number of given surface points. If we have a surface with thousands of points, then the sum will give a very accurate result. But if we only have a few hundred surface points to begin with, the result will have noticeable noise because the approximation no longer applies.

The code also derives an effective projected radius R_{eff} by averaging the edge radii from the earlier method. That radius is used to turn the planet's normalized radius into a physical radius: $R_p = \texttt{rp\_rs} \times R_{\mathrm{eff}}$.

2.8 Planet Position

The planet's position is found from the orbital elements for each point in time. The code uses Kepler's third law to get the semi-major axis and then calls the module `pylightcurve` to generate the orbit. We assume the planet to be a light blocker (no emission). Then, we consider only the orbital positions in which the planet is located at positive x-coordinates. If $x < 0$, the planet begins traveling towards the backside of the star and the flux is set to 1 (no occultation).

2.9 Occulted Flux

The most computationally intensive part of the code is the calculation of the occulted flux during transit. The reason for this is because we have to integrate the stellar intensity over the planetary disk for every single temporal value in the given data array.

The code places a small circular grid centered on the planet (just large enough to cover the planet's disk) and evaluates the intensity at the grid points. Instead of interpolating the continuous intensity, it finds the nearest precomputed visible intensity value for each grid point using a k-d tree (nearest-neighbor lookup), then sums the intensities $\times$ the cell area for points that lie inside both the planet disk and the stellar disk. This is similar to a Riemann sum approach localized around the planet. It handles small planets well, because the grid resolution can be increased locally without regenerating the whole spheroid mesh.

If triangulation fails, the code has a failsafe and returns zero occulted flux for that time step.

2.10 Caching and Performance

Computing the spheroid mesh-grid, intensities, and triangulation is computationally expensive, so the code stores those results in a cache keyed by the main stellar parameters (mass, radius, temperature, beta, orientation, limb coefficients, etc.). If we run the model many times with the same stellar characteristics and different planet orbits, we save a lot of time by reusing the cached baseline.

2.11 Time Loop

Once the baseline flux is cached, the code loops over each time in `t_vals`. For each point in time, it calculates the planet position, checks whether it's in front of the star, then computes the occulted flux $\Delta F(t)$ and finally returns the normalized flux:

$$F_{\text{norm}}(t) = \frac{F_{\text{baseline}} - \Delta F(t)}{F_{\text{baseline}}}$$

The function returns an array of the normalized flux for each input time point.

2.12 Internal Subfunctions

Below is a short description of the subfunctions that executes the previous steps:

`spheroid(Re_eq, Rp_polar, num_points)`: Builds a spherical coordinate mesh (latitude-longitude) for a spheroid with equatorial radius `Re_eq` and polar radius `Rp_polar`. Returns coordinates (x, y, z) (surface points) and the polar angle grid `V` used later for gravity calculations.

`rotation_matrix_y(angle) / rotation_matrix_z(angle)`: Basic 3D rotation matrices.

`rotated_spheroid(x,y,z,lamda,i_s)`: Rotates the spheroid to align the stellar spin axis according to the projected spin-orbit angle `lamda` and stellar inclination `i_s`.

`edge_radii(y,z,yc,zc)`: Given a 2D projection of a shape, finds the convex hull and returns radial distances from center to the hull as an interpolated function of angle. Used to compute the effective projected stellar radius and normalized radial coordinates (rho).

`gravity_darkening(omega, Re_eq, Rp_polar, theta)`: Computes the local effective gravity and applies the gravity-darkening law $T_{\text{local}} = T_{\text{mean}}(g/g_{\text{mean}})^{\beta}$. It then computes an intensity via the Planck law at the observed wavelength (`obs_wavelength`).

`limb_darkening(y_proj, z_proj, u1,u2,u3,u4)`: Computes the Claret 4-parameter limb darkening factor on the projected surface ($\mu = \sqrt{1-\rho^2}$, where ρ is normalized radius).

`planet_position(t,period,e,inc,w,t_p)`: Wrapper that computes planetary coordinates at time `t` using `pylightcurve.planet_orbit`. The code computes a semi-major axis a from the input stellar mass and period (Kepler's third law) and passes parameters to pylightcurve.

`baseline_flux()`: Builds the visible hemisphere once (triangulates the visible (y,z) points), computes per-triangle areas and intensities, and returns a caching tuple: (`total_flux`, `y_vis`, `z_vis`, `I_total`, `R_eff`, ...). This is cached across calls with identical stellar parameters to avoid recomputation.

`grad_vector(x,y,z)`: Computes the surface normal (via cross products on the parametric grid) used to decide visibility (positive Nx marks the visible hemisphere).

`planet_integration(...)`: High-accuracy integration over the planet disk: creates a quasi-polar integration grid inside the planet's disk in projected coordinates, checks which points fall on the visible stellar disk, interpolates local intensity using a cKDTree, and sums up blocked flux. Controlled by `integration_grid_size`.

`flux_drop(...)`: Faster, lower-accuracy method: finds visible stellar points falling under the planet disk and computes blocked flux by triangulating those points (Delaunay). Useful as a faster approximate option.

`transit_lightcurve(t_vals)`: Loops for each input time:

1. gets planet position (x_p, y_p, z_p).
2. if the planet is on the observer side computes the blocked flux either via `planet_integration` (if `planet_focused=True`) or `flux_drop` (if `planet_focused=False`).
3. returns normalized flux = (baseline - blocked)/baseline, or 1 if the planet is behind the star.

2.13 Performance and Accuracy

- **Caching:** baseline flux and visible surface triangulation are cached across calls to avoid costly recomputation when repeating model evaluations with identical stellar parameters.
- **integration_grid_size:** increase to improve accuracy of the planet-focused integration. Default value of 45 (moderate accuracy). Doubling roughly increases compute cost by $\sim 4\times$. At default the code runtime is in the range of $0.2 - 1.5\,\text{sec}$, depending on computer capabilities.
- **planet_focused=True** is recommended due to significantly faster computations (the second method has a runtime of $\sim 40\,\text{sec}$ to $1.5\,\text{min}$).

3 Examples

3.1 Example Python script

```python
import numpy as np
import matplotlib.pyplot as plt
from ligrad import grav_dark_transit_model

t_vals = np.linspace(-0.12, 0.12, 400)
orbital_period = 3.0
st_mass = 1.0
st_mean_radius = 1.0
st_mean_temperature = 0.6
beta = 0.25
lamda = 30.0
i_s = 60.0
omega = 0.8
u1, u2, u3, u4 = 0.3, 0.1, 0.05, 0.0
e = 0.0
i_0 = 89.0
omega_p = 0.0
t_p = 0.0
rp_rs = 0.1
obs_wavelength = 800e-9

transit_flux = grav_dark_transit_model(t_vals, orbital_period, st_mass,
                    st_mean_radius, st_mean_temperature,
                    beta, lamda, i_s, omega, u1, u2, u3,
                    u4, e, i_0, omega_p, t_p, rp_rs,
                    obs_wavelength=obs_wavelength, planet_focused=True,
                    integration_grid_size=45)

plt.figure(figsize=(10,6))
plt.plot(t_vals, transit_flux)
plt.xlabel('Time from epoch (mid-transit in days)')
plt.ylabel('Normalized flux')
plt.title('Example Transit')
plt.grid(True, alpha=0.25)
plt.tight_layout()
plt.show()
```

3.1.1 Plotted Output (example figure)

Below is the plotted transit light-curve from the script above:

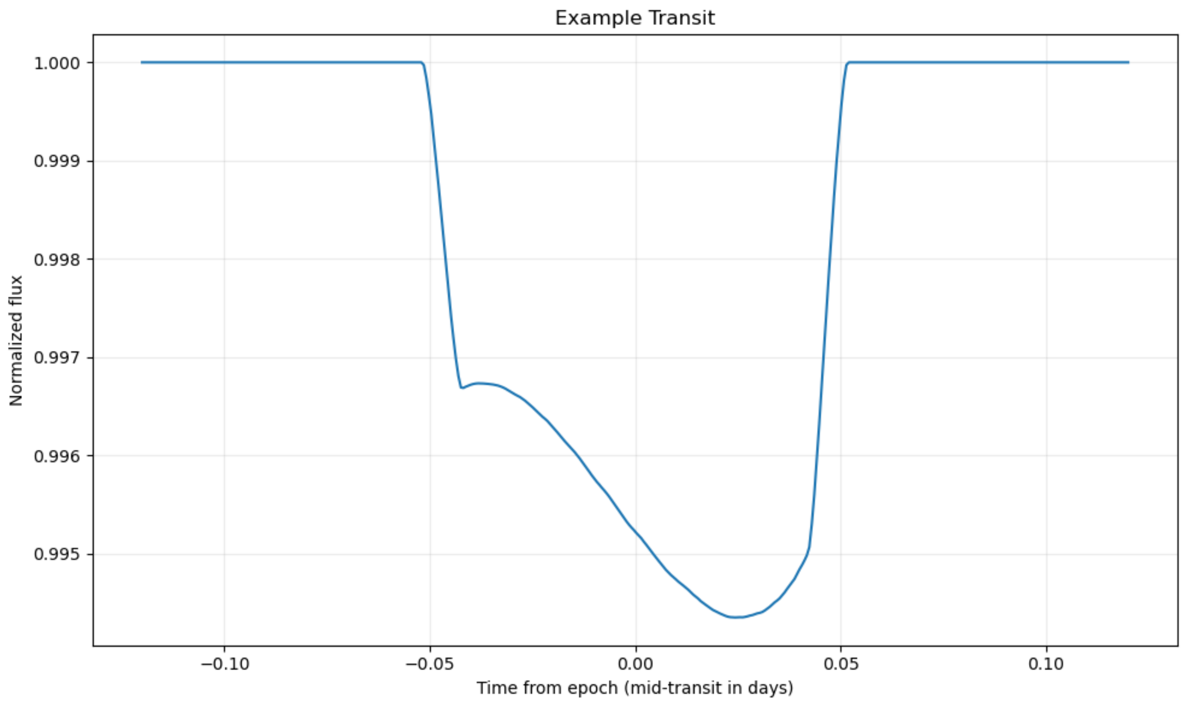
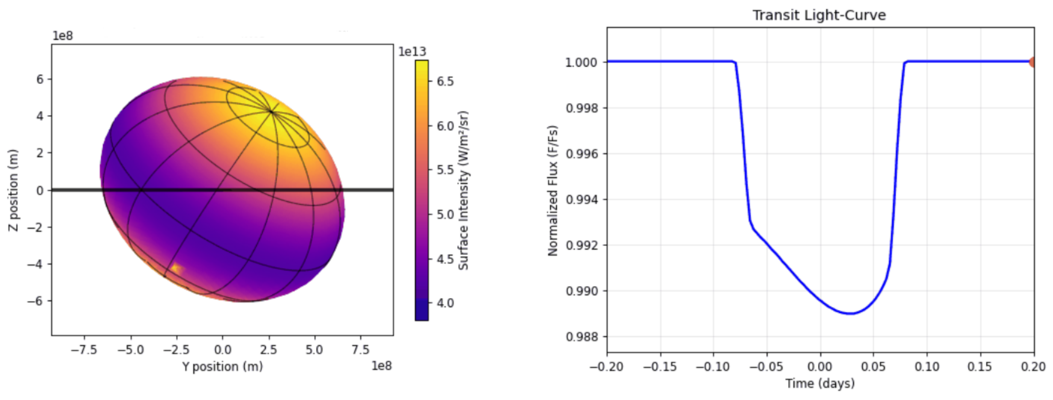


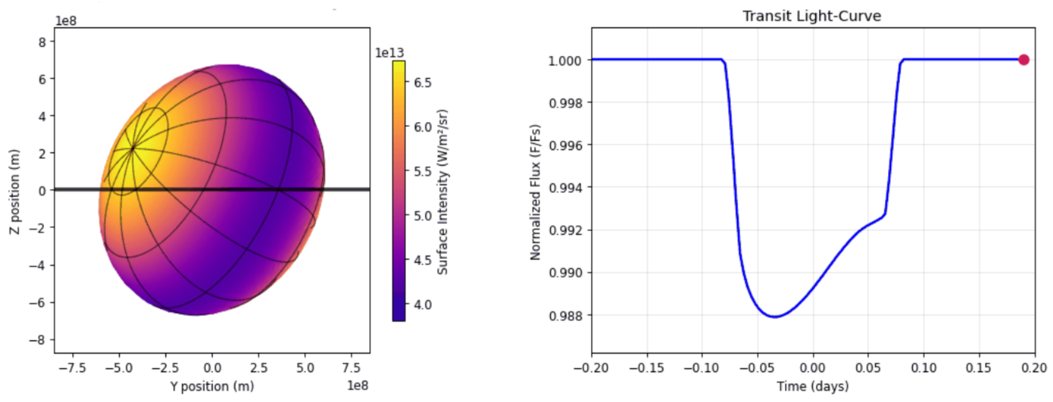
Figure 1: Simulated transit light-curve with example script

3.1.2 Example Simulations

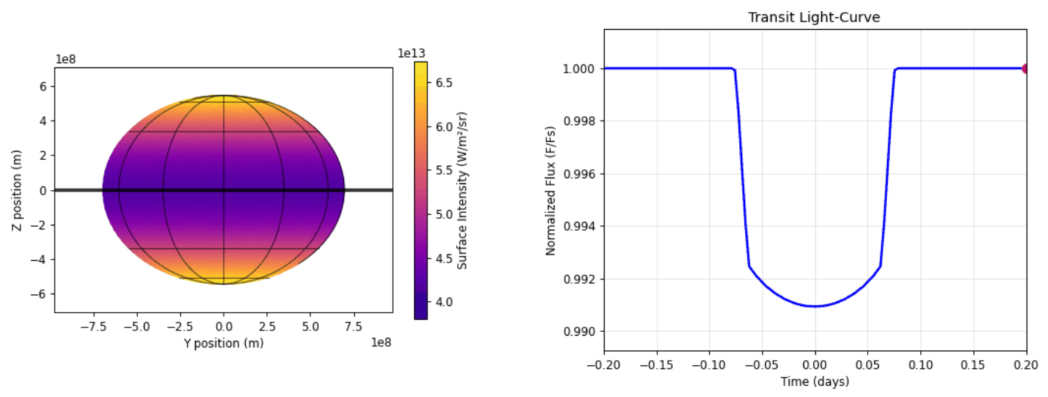
Using the code, we also generated test light-curve models for different set of parameters. First of all, for a rotation factor of $\omega = 0.75$, and a gravity-darkening exponent of $\beta = 0.15$, the parameters varied to alter the light-curves were the geometric terms $\lambda,\ i_s,\ i_0$. The rest of the parameters are kept constant with values $P = 10days,\ M = M_\odot,\ R_{mean} = R_\odot,\ T_{mean} = 9000K,\ u_1,\ u_2,\ u_3,\ u_4 = 0.5, -0.1,\ 0.05,\ 0.01,\ e = 0,\ \omega_p = 0,\ t_p = 0,\ R_p/R_\star = 0.1$. The light-curves can be seen below:



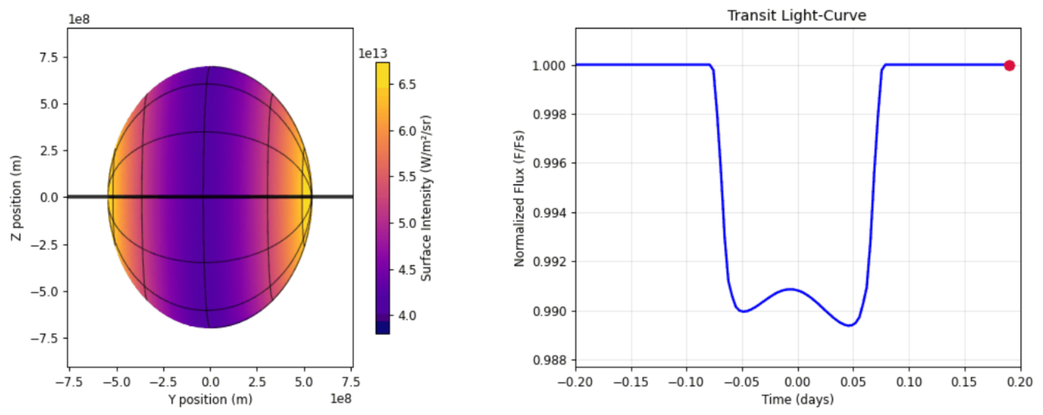
(a) Simulated transit light-curve with $\lambda = 45,\ i_s = 60,\ i_0 = 90$).



(b) Simulated transit light-curve with $\lambda = 285,\ i_s = 60,\ i_0 = 90$).



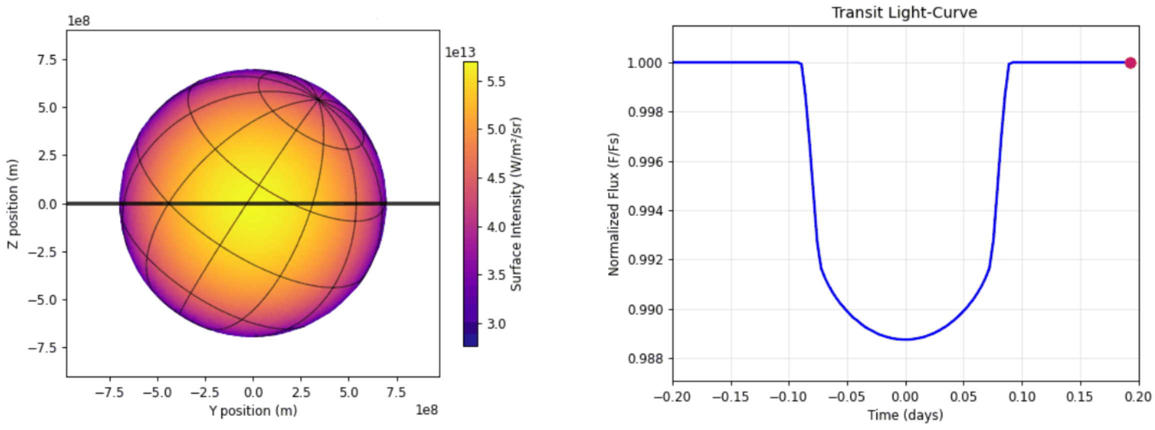
(c) Simulated transit light-curve with $\lambda = 0,\ i_s = 90,\ i_0 = 90$).



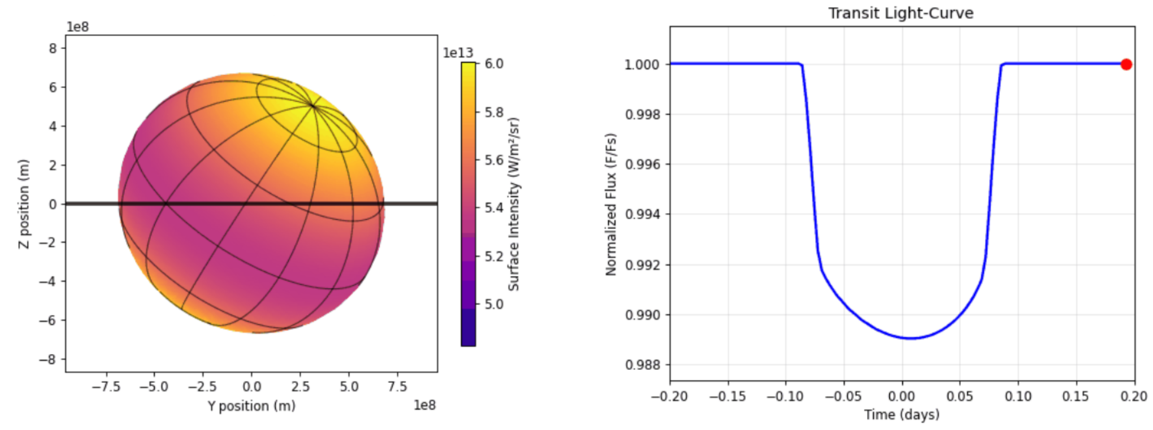
(d) Simulated transit light-curve with $\lambda = 90,\ i_s = 90,\ i_0 = 90$).

Figure 2: Simulated transit light-curves for different sets of angles $\lambda,\ i_s,\ i_0$).

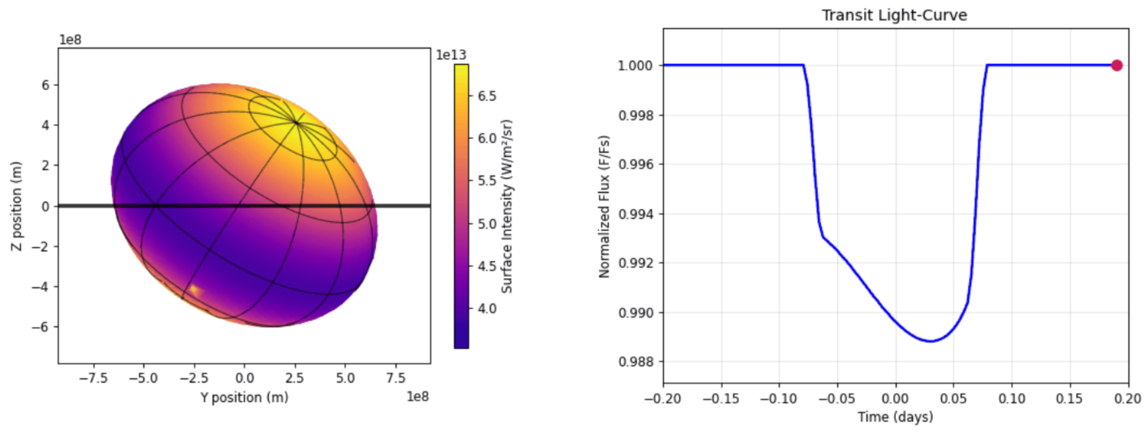
The second set of graphs is mainly based on the stellar parameters β and ω. We consider a static geometry of $\lambda = 45,\ i_s = 60,\ i_0 = 90$, as well as the same constant parameters of the previous simulations ($P = 10\,\text{days},\ M = M_\odot,\ R_{mean} = R_\odot,\ T_{mean} = 9000\,\text{K},\ u_1,\ u_2,\ u_3,\ u_4 = 0.5, -0.1,\ 0.05,\ 0.01,\ e = 0,\ \omega_p = 0,\ t_p = 0,\ R_p/R_\star = 0.1$). The new light-curves are the following:



(a) Simulated transit light-curve with $\beta = 0.15,\ \omega = 0$.

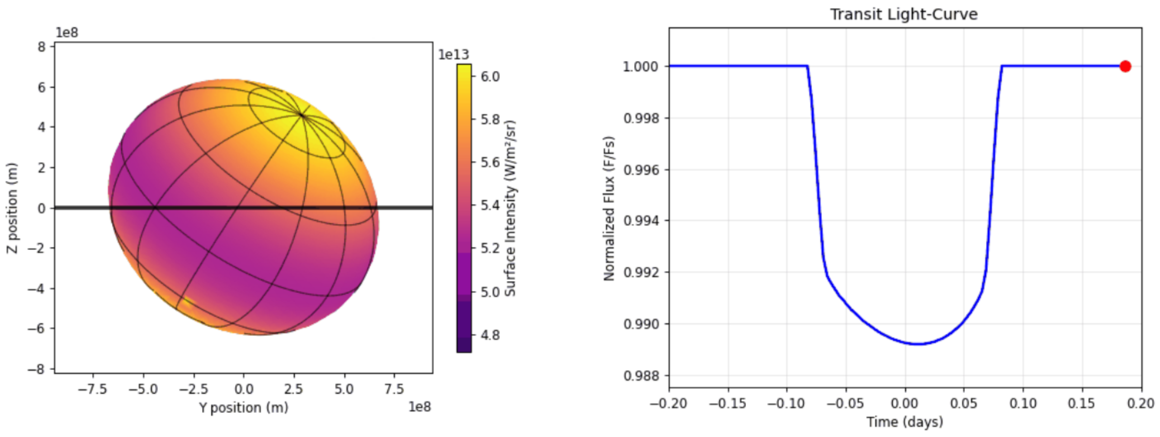


(b) Simulated transit light-curve with $\beta = 0.15,\ \omega = 0.4$.

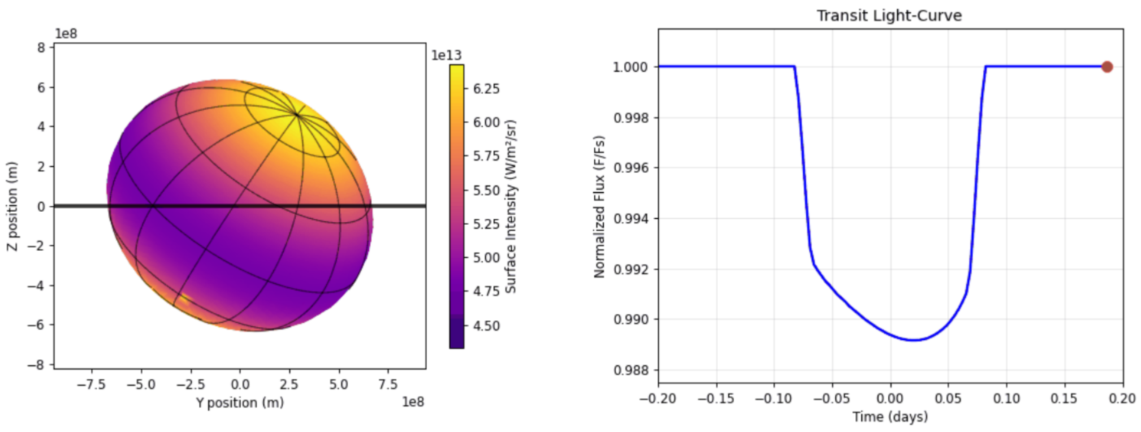


(c) Simulated transit light-curve with $\beta = 0.15,\ \omega = 0.8$.

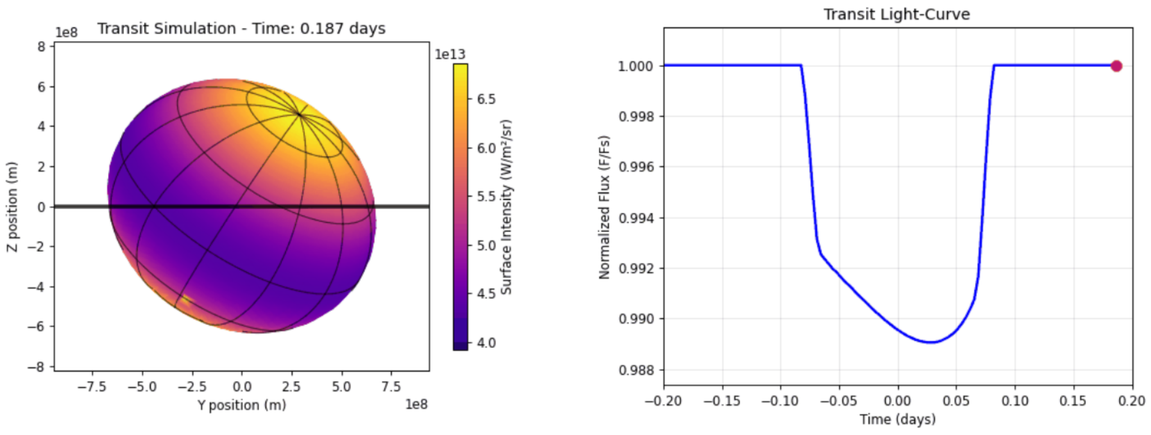
Figure 3: Simulated transit light-curves for different values of ω.



(a) Simulated transit light-curve with $\beta = 0.08,\ \omega = 0.6$.



(b) Simulated transit light-curve with $\beta = 0.16,\ \omega = 0.6$.



(c) Simulated transit light-curve with $\beta = 0.25,\ \omega = 0.6$.

Figure 4: Simulated transit light-curves for different values of β.

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), Article 167, 167. https://doi.org/10.3847/1538-4357/ac7c74

Foreman-Mackey, D., Hogg, D. W., Lang, D., & Goodman, J. (2013). emcee: The MCMC Hammer., *125*(925), 306. https://doi.org/10.1086/670067

Ginsburg, A., Sipőcz, B. M., Brasseur, C. E., Cowperthwaite, P. S., Craig, M. W., Deil, C., Guillochon, J., Guzman, G., Liedtke, S., Lian Lim, P., Lockhart, K. E., Mommert, M., Morris, B. M., Norman, H., Parikh, M., Persson, M. V., Robitaille, T. P., Segovia, J.-C., Singer, L. P., … a subset of the astropy collaboration. (2019). astroquery: An Astronomical Web-querying Package in Python., *157*, Article 98, 98. https://doi.org/10.3847/1538-3881/aafc33

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

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

Kokori, A., Tsiaras, A., Edwards, B., Rocchetto, M., Tinetti, G., Wünsche, A., Paschalis, N., Agnihotri, V. K., Bachschmidt, M., Bretton, M., Caines, H., Caló, M., Casali, R., Crow, M., Dawes, S., Deldem, M., Deligeorgopoulos, D., Dymock, R., Evans, P., … Tomatis, A. (2022). ExoClock project: an open platform for monitoring the ephemerides of Ariel targets with contributions from the public. *Experimental Astronomy*, *53*(2), 547–588. https://doi.org/10.1007/s10686-020-09696-3

Tsiaras, A., Waldmann, I. P., Rocchetto, M., Varley, R., Morello, G., Damiano, M., & Tinetti, G. (2016, December). *pylightcurve: Exoplanet lightcurve model.*

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

Voulimiotis, E. (2025). *Transit model for gravity-darkened oblate stars* [Bachelor's Thesis]. Aristotle University of Thessaloniki, Faculty of Sciences, School of Physics [GRI-2025-51194]. https://ikee.lib.auth.gr/record/368128/