# platon Documentation

*Release 2.0*

**Michael Zhang, Yayaati Chachan**

**Feb 02, 2020**

# Contents

# CHAPTER 1

# Introduction

PLATON (PLanetary Atmospheric Transmission for Observer Noobs) is a fast and easy to use forward modelling and retrieval tool for exoplanet atmospheres. It is based on ExoTransmit by Eliza Kempton. The two main modules are:

1. *TransitDepthCalculator*: computes a transit spectrum for an exoplanet
2. *Retriever*: retrieves atmospheric properties of an exoplanet, given the observed transit spectrum. The properties that can be retrieved are metallicity, C/O ratio, cloudtop pressure, scattering strength, and scattering slope

The transit spectrum is calculated from 300 nm to 30 um, taking into account gas absorption, collisionally induced gas absorption, clouds, and scattering. *TransitDepthCalculator* is written entirely in Python and is designed for performance. By default, it calculates transit depths on a fine wavelength grid ($\lambda/\Delta\lambda$ = 1000 with 4616 wavelength points), which takes ~170 milliseconds on a midrange consumer computer. The user can instead specify bins which are directly relevant to matching observational data, in which case the code avoids computing depths for irrelevant wavelengths and is many times faster.

*Retriever* uses TransitDepthCalculator as a forward model, and can retrieve atmospheric properties using either MCMC or nested sampling. Typically, nestled sampling finishes in < 10 min. MCMC relies on the user to specify the number of iterations, but typically reaches convergence in less than an hour.

CHAPTER 2

# Install

Before installing PLATON, it is highly recommended to have a fast linear algebra library (BLAS) and verify that numpy is linked to it. This is because the heart of the radiative transfer code is a matrix multiplication operation conducted through numpy.dot, which in turn calls a BLAS library if it can find one. If it can't find one, your code will be many times slower.

On Linux, a good choice is OpenBLAS. You can install it on Ubuntu with:

```
sudo apt install libopenblas-dev
```

On OS X, a good choice is Accelerate/vecLib, which should already be installed by default.

To check if your numpy is linked to BLAS, do:

```
numpy.__config__.show()
```

If blas_opt_info mentions OpenBLAS or vecLib, that's a good sign. If it says "NOT AVAILABLE", that's a bad sign.

Once you have a BLAS installed and linked to numpy, download PLATON, install the requirements, and install PLATON itself. The easiest way is to use pip:

```
pip install platon
```

That's it! Because PyPI has a size limit on packages, this will not install the data files. The data files will be automatically downloaded when PLATON is first run.

Another option is to install from source:

```
git clone https://github.com/ideasrule/platon.git
cd platon/
python setup.py install
```

In this case, you can run unit tests to make sure everything works:

```
nosetests -v
```

The unit tests should also give you a good idea of how fast the code will be. On a decent Ubuntu machine with OpenBLAS, it takes 2 minutes.

# CHAPTER 3

# Quick start

The fastest way to get started is to look at the examples/ directory, which has examples on how to compute transit depths from planetary parameters, and on how to retrieve planetary parameters from transit depths. This page is a short summary of the more detailed examples.

To compute transit depths, look at transit_depth_example.py, then go to *TransitDepthCalculator* for more info. In short:

```
from platon.transit_depth_calculator import TransitDepthCalculator
from platon.constants import M_jup, R_jup, R_sun

# All inputs and outputs for PLATON are in SI

Rs = 1.16 * R_sun
Mp = 0.73 * M_jup
Rp = 1.40 * R_jup
T = 1200

# The initializer loads all data files.  Create a TransitDepthCalculator
# object and hold on to it
calculator = TransitDepthCalculator()

# compute_depths is fast once data files are loaded
wavelengths, depths, info_dict = calculator.compute_depths(Rs, Mp, Rp, T, logZ=0, CO_
↪ratio=0.53, full_output=True)
```

You can adjust a variety of parameters, including the metallicity (Z) and C/O ratio. By default, logZ = 0 and C/O = 0.53. Any other value for logZ and C/O in the range -1 < logZ < 3 and 0.2 < C/O < 2 can also be used. full_output=True indicates you'd like extra information about the atmosphere, which is returned in info_dict. info_dict includes parameters like the temperatures, pressures, radii, abundances, and molecular weights of each atmospheric layer, and the line of sight optical depth (tau_los) through each layer.

You can also specify custom abundances, such as by providing the filename or one of the abundance files included in the package (from ExoTransmit). The custom abundance files specified by the user must be compatible with the ExoTransmit format:

```
calculator.compute_depths(Rs, Mp, Rp, T, logZ=None, CO_ratio=None,
                          custom_abundances=filename)
```

To retrieve atmospheric parameters, look at retrieve_example.py, then go to *Retriever* for more info. In short:

```
from platon.fit_info import FitInfo
from platon.retriever import Retriever

# Set your best guess.  T_star is used only to improve binning of transit
# depths.
fit_info = retriever.get_default_fit_info(Rs, Mp, Rp, T, logZ=0, T_star=6100)

# Decide what you want to fit for, and add those parameters to fit_info

# Fit for the stellar radius and planetary mass using Gaussian priors.  This
# is a way to account for the uncertainties in the published values
fit_info.add_gaussian_fit_param('Rs', 0.02*R_sun)
fit_info.add_gaussian_fit_param('Mp', 0.04*M_jup)

# Fit for other parameters using uniform priors
fit_info.add_uniform_fit_param('R', 0.9*R_guess, 1.1*R_guess)
fit_info.add_uniform_fit_param('T', 0.5*T_guess, 1.5*T_guess)
fit_info.add_uniform_fit_param("log_scatt_factor", 0, 1)
fit_info.add_uniform_fit_param("logZ", -1, 3)
fit_info.add_uniform_fit_param("log_cloudtop_P", -0.99, 5)
fit_info.add_uniform_fit_param("error_multiple", 0.5, 5)

# Run nested sampling
result = retriever.run_multinest(bins, depths, errors, fit_info, plot_best=True)
```

Here, *bins* is a N x 2 array representing the start and end wavelengths of the bins, in metres; *depths* is a list of N transit depths; and *errors* is a list of N errors on those transit depths. *plot_best=True* indicates that the best fit solution should be plotted, along with the measured transit depths and their errors.

The example above retrieves the planetary radius (at a base pressures of 100,000 Pa), the temperature of the isothermal atmosphere, and the metallicity. Other parameters you can retrieve for are the stellar radius, the planetary mass, C/O ratio, the cloudtop pressure, the scattering factor, the scattering slope, and the error multiple–which multiplies all errors by a constant. We recommend either fixing the stellar radius and planetary mass to the measured values, or only allowing them to vary 2 standard deviations aw

Once you get the *result* object, you can make a corner plot:

```
fig = corner.corner(result.samples, weights=result.weights,
                    range=[0.99] * result.samples.shape[1],
                    labels=fit_info.fit_param_names)
```

Additionally, result.logl stores the log likelihoods of the points in result.samples.

If you prefer using MCMC instead of Nested Sampling in your retrieval, you can use the run_emcee method instead of the run_multinest method. Do note that Nested Sampling tends to be much faster and it does not require specification of a termination point:

```
result = retriever.run_emcee(bins, depths, errors, fit_info)
```

For MCMC, the number of walkers and iterations/steps can also be specified. The *result* object returned by run_emcee is slighly different from that returned by run_multinest. To make a corner plot with the result of run_emcee:

```
fig = corner.corner(result.flatchain, range=[0.99] * result.flatchain.shape[1],
                    labels=fit_info.fit_param_names)
```

CHAPTER 4

# Mie scattering

By default, PLATON uses a parametric model to account for scattering, with an amplitude and a slope. However, PLATON also has the ability to compute Mie scattering in place of the parametric model.

To use Mie scattering, follow *Quick start* to see how to do forward models and retrievals using the default parametric model. To use Mie scattering instead:

```
calculator.compute_depths(Rs, Mp, Rp, T,
    ri = 1.33-0.1j, frac_scale_height = 0.5, number_density = 1e9,
    part_size = 1e-6, cloudtop_pressure=1e5)
```

This computes Mie scattering for particles with complex refractive index 1.33-0.1j. The particles follow a lognormal size distribution with a mean radius of 1 micron and standard deviation of 0.5. They have a density of $10^9/m^3$ at the cloud-top pressure of $10^5$ Pa, declining with altitude with a scale height of 0.5 times the gas scale height.

To retrieve Mie scattering parameters, make sure to set log_scatt_factor to -np.inf, log_number_density to a finite value, and ri to the complex refractive index of the scattering particles (which cannot be a free parameter):

```
fit_info = retriever.get_default_fit_info(Rs, Mp, Rp, T,
    log_scatt_factor = -np.inf, log_number_density = 9, ri = 1.33-0.1j)

fit_info.add_uniform_fit_param('log_number_density', 5, 15)
fit_info.add_uniform_fit_param('log_part_size', -7, -4)
```

# CHAPTER 5

# Eclipse depths (beta)

Although PLATON began life as a transmission spectrum calculator, we have also written an eclipse depth calculator and retriever. These have not yet undergone extensive testing, so use at your own risk.

To use the eclipse depth calculator, first create a temperature-pressure profile:

```
from platon.TP_profile import Profile
p = Profile()
p.set_parametric(1200, 500, 0.5, 0.6, 1e6, 1900)
```

This creates a parametric T-P profile according to Madhusudhan & Seager 2018. The parameters are: $T_0$, $P_1$, $\alpha_1$, $\alpha_2$, $P_3$, $T_3$. $P_0$ is set to $10^{-4}$ Pa, while $P_2$ and $T_2$ are derived from the six specified parameters.

Then, call the eclipse depth calculator:

```
from eclipse_depth_calculator import EclipseDepthCalculator

wavelengths, depths = calc.compute_depths(p, Rs, Mp, Rp, Tstar)
```

Most of the same parameters accepted by the transit depth calculator are also accepted by the eclipse depth calculator.

It is also possible to retrieve on combined transit and eclipse depths:

```
from platon.combined_retriever import CombinedRetriever

retriever = CombinedRetriever()
fit_info = retriever.get_default_fit_info(Rs, Mp, Rp, T_limb,
               T0=1200, P1=500, alpha1=0.5, alpha2=0.6, P3=1e6, T3=1900)

fit_info.add_uniform_fit_param(...)
fit_info.add_uniform_fit_param(...)

result = retriever.run_multinest(transit_bins, transit_depths, transit_errors,
                                 eclipse_bins, eclipse_depths, eclipse_errors,
                                 fit_info)
```

Here, T_limb is the temperature at the planetary limb (used for transit depths), while the T-P profile parameters are for the dayside (used for eclipse depths).

CHAPTER 6

# Visualizer

If you want to see what your exoplanet might look like in transit, the Visualizer module is for you! Visualizer uses the absorption profile calculated by the transit depth calculator, which you can get using:

```
calculator = TransitDepthCalculator()
wavelengths, depths, info = calculator.compute_depths(Rs, Mp, Rp, T, full_output=True)
```

Then, to draw an image:

```
color_bins = 1e-6 * np.array([
    [4, 5],
    [3.2, 4],
    [1.1, 1.7]])
visualizer = Visualizer()
image, m_per_pix = visualizer.draw(info, color_bins, method='disk')
```

This maps all wavelengths between 4–5 microns to red, while 3.2–4 microns maps to green and 4–5 microns maps to blue. The draw function returns an image and an image scale, in meters per pixel. The image can be displayed with pyplot:

```
plt.imshow(image)
```

The 'method' argument can either be 'disk' or 'layers'. The difference is illustrated in the example images below. For purely aesthetic purposes, we have also made the star light a pale yellow color by passing [1,1,0.8] to the star_color argument of draw.

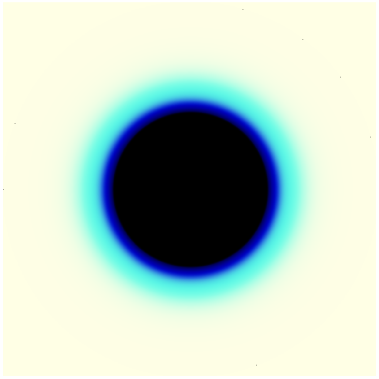

Fig. 2: 55 Cnc e as a disk transiting a yellow star

# CHAPTER 7

# Questions & Answers

This document describes niche use cases that the Quick Start does not cover. For typical usage patterns, consult the files in examples/ and the Quick Start, in that order.

- **What physics does PLATON take into account?**

  We account for gas absorption, collisional absorption, an opaque cloud deck, and scattering with user-specified slope and amplitude (or Rayleigh, if not specified). 34 chemical species are included in our calculations, namely the ones listed in data/species_info. The abundances of these species were calculated using GGchem for a grid of metallicity, C/O ratio, temperature, and pressure, assuming equilibrium chemistry. Metallicity ranges from 0.1-1000x solar, C/O ratio from 0.2 to 2, temperature from 300 to 3000 K, and pressure from 10^-4 to 10^8 Pa. If you wander outside these limits, PLATON will throw a ValueError.

- **Why does PLATON not exactly agree with ExoTransmit?**

  We have made many improvements to the ExoTransmit algorithm to enhance the accuracy of our transit depth calculations. Among the most consequential are allowing the gravitational acceleration to vary with height, and truncating the atmosphere at 10^-4 Pa instead of 0.1 Pa. Both of these changes tend to increase the transit depth. Precise agreement with ExoTransmit should not be expected.

- **How do I use PLATON with ExoTransmit input files? Or: how do I specify custom abundances and T/P profiles?**

  By example:

```
from platon.abundance_getter import AbundanceGetter
from platon.transit_depth_calculator import TransitDepthCalculator

_, pressures, temperatures = np.loadtxt("t_p_1200K.dat", skiprows=1, unpack=True)

# These files are found in examples/custom_abundances.  They are equivalent
# to the ExoTransmit EOS files, except that COS is renamed to OCS
abundances = AbundanceGetter.from_file("abund_1Xsolar_cond.dat")

calculator = TransitDepthCalculator()
wavelengths, transit_depths = calculator.compute_depths(star_radius, planet_mass,␣
↪planet_radius, temperature=None, logZ=None, CO_ratio=None, custom_
↪abundances=abundances, custom_T_profile=temperatures, custom_P_
↪profile=pressures)
```

(continues on next page)

(continued from previous page)

- **Should I use PLATON with ExoTransmit input files?**

  No. The recommended usage is much simpler, and is outlined in both the examples/ directory and the Quick Start.

- **Which parameters are supported in retrieval?** See the documentation for *get_default_fit_info()*. All arguments to this method are possible fit parameters. However, we recommend not fitting for T_star, as it has a very small effect on the result to begin with. Mp and Rs are usually measured to greater precision than you can achieve in a fit, but we recommend fitting them with Gaussian priors to take into account the measurement errors.

- **Should I use run_multinest, or run_emcee?**

  That depends on whether you like nested sampling or MCMC! You should try both and compare the results. Nestled sampling is faster and has an automatically determined stopping point, so we recommend starting with that.

- **My corner plots look ugly. What do I do?**

  If you're using nested sampling, increase the number of live points. This will increase the number of samples your corner plot is generated from:

```
# By default, npoints is 100
result = retriever.run_multinest(bins, depths, errors, fit_info, npoints=1000)
```

  If you're using MCMC, increase nsteps. However, we should note that the default of 10,000 should be enough to generate a good corner plot for almost all cases. In fact, it should be overkill.

- **How do I get statistics from the nestled sampling result?**

  The easiest way is to resample and get final samples with equal weights. You can then take the mean, median, 16th and 84th percentiles, and any other statistics you normally do:

```
import nestle
equal_samples = nestle.resample_equal(result.samples, result.weights)

# Your first column is not necessarily radii
radii = equal_samples[:,0]
print(np.percentile(radii, 16), np.median(radii), np.percentile(radii, 84))
```

- **How do I do check what effect a species has on the transit spectrum?** You can tweak the atmospheric abundances and see what happens. First, get baseline abundances:

```
from platon.abundance_getter import AbundanceGetter
getter = AbundanceGetter()
# Solar logZ and C/O ratio. Modify as required.
abundances = getter.get(0, 0.53)
```

  You can then modify this at will:

```
# Zero out CO.  (Note that if CO is a major component, you should probably
# renormalize the abundances of other species so that they add up to 1.)

abundances["CO"] *= 0

# Set CH4 abundance to a constant throughout the atmosphere
```

(continues on next page)

(continued from previous page)

```
abundances["CH4"] *= 0
abundances["CH4"] += 1e-5
```

Then call compute_depths with logZ and CO_ratio set to None:

```
calculator.compute_depths(star_radius, planet_mass, planet_radius, temperature,␣
↪logZ=None, CO_ratio=None, custom_abundances=abundances)
```

- **How do I specify custom abundances in the forward model?** See the answer to the above question. If you want to specify different abundances for each temperature/pressure point instead of a constant abundance, see the documentation for custom_abundances in *compute_depths()*
- **How do I retrieve individual species abundances?** You can't. While this would be trivial to implement–and you can do so if you really need to–it could easily lead to combinations of species that are unstable on very short timescales. We have therefore decided not to support retrieving on individual abundances.

# CHAPTER 8

# platon

## 8.1 platon package

### 8.1.1 Submodules

### 8.1.2 platon.TP_profile module

**class** platon.TP_profile.**Profile**(*num_profile_heights=500*)
Bases: object

**set_from_arrays**(*P_profile*, *T_profile*)

**set_from_opacity**(*Tirr*, *info_dict*, *visible_cutoff=8e-07*, *Tint=100*)

**set_isothermal**(*T*)

**set_parametric**(*T0*, *P1*, *alpha1*, *alpha2*, *P3*, *T3*)

### 8.1.3 platon.abundance_getter module

**class** platon.abundance_getter.**AbundanceGetter**(*include_condensation=True*)
Bases: object

**static from_file**(*filename*)
Reads abundances file in the ExoTransmit format (called "EOS" files in ExoTransmit), returning a dictionary mapping species name to an abundance array of dimension

**get**(*logZ*, *CO_ratio=0.53*)
Get an abundance grid at the specified logZ and C/O ratio. This abundance grid can be passed to TransitDepthCalculator, with or without modifications. The end user should not need to call this except in rare cases.

**Returns** **abundances** – A dictionary mapping species name to a 2D abundance array, specifying the number fraction of the species at a certain temperature and pressure.

**Return type** dict of np.ndarray

**is_in_bounds**(*logZ*, *CO_ratio*, *T*)
Check to see if a certain metallicity, C/O ratio, and temperature combination is within the supported bounds

### 8.1.4 platon.combined_retriever module

### 8.1.5 platon.constants module

platon.constants.**M_earth = 5.97236e+24**
Earth mass

platon.constants.**M_jup = 1.89819e+27**
Jupiter mass

platon.constants.**M_sun = 1.98848e+30**
Solar mass

platon.constants.**R_earth = 6378100.0**
Earth radius

platon.constants.**R_jup = 71492000.0**
Jupiter radius

platon.constants.**R_sun = 695700000.0**
Solar radius

### 8.1.6 platon.eclipse_depth_calculator module

### 8.1.7 platon.errors module

**exception** platon.errors.**AtmosphereError**
Bases: Exception

### 8.1.8 platon.fit_info module

**class** platon.fit_info.**FitInfo**(*guesses_dict*)
Bases: object

**add_gaussian_fit_param**(*name*, *std*, *low_guess=None*, *high_guess=None*)
Fit for the parameter *name* using a Gaussian prior with standard deviation *std*. If using emcee, the walkers' initial values for this parameter are randomly selected to be between *low_guess* and *high_guess*. If *low_guess* is None, it is set to mean-2*std; if *high_guess* is None, it is set to mean+2*std.

**add_uniform_fit_param**(*name*, *low_lim*, *high_lim*, *low_guess=None*, *high_guess=None*)
Fit for the parameter *name* using a uniform prior between *low_lim* and *high_lim*. If using emcee, the walkers' initial values for this parameter are randomly selected to be between *low_guess* and *high_guess*. If not specified, *low_guess* is set to *low_lim*, and similarly with *high_guess*.

### 8.1.9 platon.retriever module

**class** platon.retriever.**Retriever**
Bases: object

**static get_default_fit_info**(*Rs*, *Mp*, *Rp*, *T*, *logZ=0*, *CO_ratio=0.53*, *log_cloudtop_P=inf*, *log_scatt_factor=0*, *scatt_slope=4*, *error_multiple=1*, *T_star=None*, *T_spot=None*, *spot_cov_frac=None*, *frac_scale_height=1*, *log_number_density=-inf*, *log_part_size=-6*, *ri=None*)

Get a `FitInfo` object filled with best guess values. A few parameters are required, but others can be set to default values if you do not want to specify them. All parameters are in SI. For information on the parameters, see the documentation for `compute_depths()`

**Returns** **fit_info** – This object is used to indicate which parameters to fit for, which to fix, and what values all parameters should take.

**Return type** `FitInfo` object

**run_emcee**(*wavelength_bins*, *depths*, *errors*, *fit_info*, *nwalkers=50*, *nsteps=1000*, *include_condensation=True*, *plot_best=False*)

Runs affine-invariant MCMC to retrieve atmospheric parameters.

**Parameters**

- **wavelength_bins** (*array_like, shape (N,2)*) – Wavelength bins, where wavelength_bins[i][0] is the start wavelength and wavelength_bins[i][1] is the end wavelength for bin i.
- **depths** (*array_like, length N*) – Measured transit depths for the specified wavelength bins
- **errors** (*array_like, length N*) – Errors on the aforementioned transit depths
- **fit_info** (`FitInfo` object) – Tells the method what parameters to freely vary, and in what range those parameters can vary. Also sets default values for the fixed parameters.
- **nwalkers** (*int, optional*) – Number of walkers to use
- **nsteps** (*int, optional*) – Number of steps that the walkers should walk for
- **include_condensation** (*bool, optional*) – When determining atmospheric abundances, whether to include condensation.
- **plot_best** (*bool, optional*) – If True, plots the best fit model with the data

**Returns** **result** – This returns emcee's EnsembleSampler object. The most useful attributes in this item are result.chain, which is a (W x S X P) array where W is the number of walkers, S is the number of steps, and P is the number of parameters; and result.lnprobability, a (W x S) array of log probabilities. For your convenience, this object also contains result.flatchain, which is a (WS x P) array where WS = W x S is the number of samples; and result.flatlnprobability, an array of length WS

**Return type** EnsembleSampler object

**run_multinest**(*wavelength_bins*, *depths*, *errors*, *fit_info*, *include_condensation=True*, *plot_best=False*, ***nestle_kwargs*)

Runs nested sampling to retrieve atmospheric parameters.

**Parameters**

- **wavelength_bins** (*array_like, shape (N,2)*) – Wavelength bins, where wavelength_bins[i][0] is the start wavelength and wavelength_bins[i][1] is the end wavelength for bin i.
- **depths** (*array_like, length N*) – Measured transit depths for the specified wavelength bins
- **errors** (*array_like, length N*) – Errors on the aforementioned transit depths

- **fit_info** (*FitInfo* object) – Tells us what parameters to freely vary, and in what range those parameters can vary. Also sets default values for the fixed parameters.
- **include_condensation** (*bool, optional*) – When determining atmospheric abundances, whether to include condensation.
- **plot_best** (*bool, optional*) – If True, plots the best fit model with the data
- ****nestle_kwargs** (*keyword arguments to pass to nestle's sample method*)

**Returns** **result** – This returns the object returned by nestle.sample The object is dictionary-like and has many useful items. For example, result.samples (or alternatively, result["samples"]) are the parameter values of each sample, result.weights contains the weights, and result.logl contains the log likelihoods. result.logz is the natural logarithm of the evidence.

**Return type** Result object

### 8.1.10 platon.transit_depth_calculator module

**class** platon.transit_depth_calculator.**TransitDepthCalculator**(*include_condensation=True*, *num_profile_heights=500*, *ref_pressure=100000.0*)

Bases: object

**__init__**(*include_condensation=True*, *num_profile_heights=500*, *ref_pressure=100000.0*)

All physical parameters are in SI.

**Parameters**

- **include_condensation** (*bool*) – Whether to use equilibrium abundances that take condensation into account.
- **num_profile_heights** (*int*) – The number of zones the atmosphere is divided into
- **ref_pressure** (*float*) – The planetary radius is defined as the radius at this pressure

**change_wavelength_bins**(*bins*)

Specify wavelength bins, instead of using the full wavelength grid in self.lambda_grid. This makes the code much faster, as *compute_depths* will only compute depths at wavelengths that fall within a bin.

**Parameters** **bins** (*array_like, shape (N,2)*) – Wavelength bins, where bins[i][0] is the start wavelength and bins[i][1] is the end wavelength for bin i. If bins is None, resets the calculator to its unbinned state.

**Raises** NotImplementedError – Raised when *change_wavelength_bins* is called more than once, which is not supported.

**compute_depths**(*star_radius*, *planet_mass*, *planet_radius*, *temperature*, *logZ=0*, *CO_ratio=0.53*, *add_scattering=True*, *scattering_factor=1*, *scattering_slope=4*, *scattering_ref_wavelength=1e-06*, *add_collisional_absorption=True*, *cloud_top_pressure=inf*, *custom_abundances=None*, *custom_T_profile=None*, *custom_P_profile=None*, *T_star=None*, *T_spot=None*, *spot_cov_frac=None*, *ri=None*, *frac_scale_height=1*, *number_density=0*, *part_size=1e-06*, *full_output=False*, *min_abundance=1e-99*)

Computes transit depths at a range of wavelengths, assuming an isothermal atmosphere. To choose bins, call change_wavelength_bins().

**Parameters**

- **star_radius** (*float*) – Radius of the star
- **planet_mass** (*float*) – Mass of the planet, in kg

- **planet_radius** (*float*) – Radius of the planet at 100,000 Pa. Must be in metres.
- **temperature** (*float*) – Temperature of the isothermal atmosphere, in Kelvin
- **logZ** (*float*) – Base-10 logarithm of the metallicity, in solar units
- **CO_ratio** (*float, optional*) – C/O atomic ratio in the atmosphere. The solar value is 0.53.
- **add_scattering** (*bool, optional*) – whether Rayleigh scattering is taken into account
- **scattering_factor** (*float, optional*) – if *add_scattering* is True, make scattering this many times as strong. If *scattering_slope* is 4, corresponding to Rayleigh scattering, the absorption coefficients are simply multiplied by *scattering_factor*. If slope is not 4, *scattering_factor* is defined such that the absorption coefficient is that many times as strong as Rayleigh scattering at *scattering_ref_wavelength*.
- **scattering_slope** (*float, optional*) – Wavelength dependence of scattering, with 4 being Rayleigh.
- **scattering_ref_wavelength** (*float, optional*) – Scattering is *scattering_factor* as strong as Rayleigh at this wavelength, expressed in metres.
- **add_collisional_absorption** (*float, optional*) – Whether collisionally induced absorption is taken into account
- **cloudtop_pressure** (*float, optional*) – Pressure level (in Pa) below which light cannot penetrate. Use np.inf for a cloudless atmosphere.
- **custom_abundances** (*str or dict of np.ndarray, optional*) – If specified, overrides *logZ* and *CO_ratio*. Can specify a filename, in which case the abundances are read from a file in the format of the EOS/ files. These are identical to ExoTransmit's EOS files. It is also possible, though highly discouraged, to specify a dictionary mapping species names to numpy arrays, so that custom_abundances['Na'][3,4] would mean the fractional number abundance of Na at a pressure of self.P_grid[3] and temperature of self.T_grid[4].
- **custom_T_profile** (*array-like, optional*) – If specified and custom_P_profile is also specified, divides the atmosphere into user-specified P/T points, instead of assuming an isothermal atmosphere with T = *temperature*.
- **custom_P_profile** (*array-like, optional*) – Must be specified along with *custom_T_profile* to use a custom P/T profile. Pressures must be in Pa.
- **T_star** (*float, optional*) – Effective temperature of the star. If you specify this and use wavelength binning, the wavelength binning becomes more accurate.
- **T_spot** (*float, optional*) – Effective temperature of the star spots. This can be used to make wavelength dependent correction to the observed transit depths.
- **spot_cov_frac** (*float, optional*) – The spot covering fraction of the star by area. This can be used to make wavelength dependent correction to the transit depths.
- **ri** (*complex, optional*) – Complex refractive index n - ik (where k > 0) of the particles responsible for Mie scattering. If provided, Mie scattering will be computed. In that case, scattering_factor and scattering_slope must be set to 1 and 4 (the default values) respectively.
- **frac_scale_height** (*float, optional*) – The number density of Mie scattering particles is proportional to P^(1/frac_scale_height). This is similar to, but a bit different from, saying that the scale height of the particles is frac_scale_height times that of the gas.
- **number_density** (*float, optional*) – The number density (in m^-3) of Mie scattering particles

- **part_size** (*float, optional*) – The mean radius of Mie scattering particles. The distribution is assumed to be log-normal, with a standard deviation of 0.5.
- **full_output** (*bool, optional*) – If True, returns info_dict as a third return value.

**Raises** `ValueError` – Raised when invalid parameters are passed to the method

**Returns**

- **wavelengths** (*array of float*) – Central wavelengths, in metres
- **transit_depths** (*array of float*) – Transit depths at *wavelengths*
- **info_dict** (*dict*) – Returned if full_output is True, containing intermediate quantities calculated by the method. These are: absorption_coeff_atm, tau_los, stellar_spectrum, radii, P_profile, T_profile, mu_profile, atm_abundances, unbinned_depths, unbinned_wavelengths

### 8.1.11 platon.visualizer module

### 8.1.12 Module contents

CHAPTER 9

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## p

# Index

## Symbols



## A



## C



## F



## G



## I



## M



## P



## R



## S



## T