platon Documentation

Release 1.0

Michael Zhang, Yayaati Chachan

Jun 19, 2018

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/
pip install -r requirements.txt
python setup.py install
```

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

```
python setup.py test
```

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
calculator.compute_depths(Rs, Mp, Rp, T, logZ=0, CO_ratio=0.53)
```

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. You can use a dictionary of numpy arrays to specify abundances as well (See the API). 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

platon

4.1 platon package

4.1.1 Submodules

4.1.2 platon.abundance_getter module

`class platon.abundance_getter.`**`AbundanceGetter`**`(`*include_condensation=True*`)`

> **`static from_file()`**
> 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

4.1.3 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

4.1.4 platon.fit_info module

class `platon.fit_info.`**`FitInfo`**(*guesses_dict*)

`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*.

4.1.5 platon.retriever module

class `platon.retriever.`**`Retriever`**

static `get_default_fit_info`(*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*)
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.

Parameters

- **Rs** (*float*) – Stellar radius
- **Mp** (*float*) – Planetary mass
- **Rp** (*float*) – Planetary radius
- **T** (*float*) – Temperature of the isothermal planetary atmosphere
- **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.
- **log_cloudtop_P** (*float, optional*) – Base-10 log of the pressure level (in Pa) below which light cannot penetrate. Use np.inf for a cloudless atmosphere.
- **log_scatt_factor** (*float, optional*) – Base-10 logarithm of scattering factoring, which make scattering that many times as strong. If *scatt_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 the reference wavelength of 1 um.

- **scatt_slope** (*float, optional*) – Wavelength dependence of scattering, with 4 being Rayleigh.
- **error_multiple** (*float, optional*) – All error bars are multiplied by this factor.
- **T_star** (*float, optional*) – Effective temperature of the star. This is used to make wavelength binning of transit depths more accurate.

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=10000*, *include_condensation=True*, *plot_best=False*, *max_P_profile=100000.0*)

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
- **max_P_profile** (*float, optional*) – Maximum pressure at which to calculate radiative transfer. If you change this, the planetary radius will be interpreted as the radius at this max_P_profile

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*, *maxiter=None*, *include_condensation=True*, *plot_best=False*, *max_P_profile=100000.0*)

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.
- **maxiter** (*bool, optional*) – If not None, run at most this many iterations of nestled sampling
- **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
- **max_P_profile** (*float, optional*) – Maximum pressure at which to calculate radiative transfer. If you change this, the planetary radius will be interpreted as the radius at this max_P_profile.

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

4.1.6 platon.transit_depth_calculator module

class `platon.transit_depth_calculator.`**`TransitDepthCalculator`**(*include_condensation=True, min_P_profile=0.1, max_P_profile=100000.0, num_profile_heights=400*)

`__init__`(*include_condensation=True, min_P_profile=0.1, max_P_profile=100000.0, num_profile_heights=400*)

All physical parameters are in SI.

Parameters

- **include_condensation** (*bool*) – Whether to use equilibrium abundances that take condensation into account.
- **min_P_profile** (*float*) – For the radiative transfer calculation, the atmosphere is divided into zones. This is the pressure at the topmost zone.
- **max_P_profile** (*float*) – The pressure at the bottommost zone of the atmosphere
- **num_profile_heights** (*int*) – The number of zones the atmosphere is divided into

`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.

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, cloudtop_pressure=inf, custom_abundances=None, custom_T_profile=None, custom_P_profile=None, T_star=None*)

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 self.max_P_profile (by default, 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.

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*

4.1.7 Module contents

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index