The Journal of Open Source Software

SpectralModel: a high-resolution framework for petitRADTRANS 3

Doriann Blain [1¶], **Paul Mollière** [1], **and Evert Nasedkin** [1]

1 Max Planck Institut für Astronomie, DE ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Atmospheric characterisation from spectroscopic data is a key to understand planetary formation. Two kinds of observations can be performed for this kind of analysis. Space-based observations (e.g., using the James Webb Space Telescope, JWST), are not impeded by the Earth's atmosphere, but are currently limited to low resolving powers (< 3000), which can lead to ambiguities in some species detections. Ground-based observations (e.g., using the Very Large Telescope, VLT), on the other hand, can benefits from large resolving powers ($\approx 10^5$), allowing for unambiguous species detection, but are impacted by telluric spectral lines. `petitRADTRANS` (pRT) is a radiative transfer package used for computing emission or transmission spectra of planetary atmospheres (Mollière et al., 2019). The package has an important user base, the original article being cited in 264 works at the time of writing. pRT is already relatively easy to use on space-based, low-resolution observations. However, while the package technically has the capacity to analyse high-resolution spectra, thanks to its line-by-line opacities ($\mathcal{R} = 10^6$), ground-based observations analysis is a complex and challenging task. The new `SpectralModel` object provides a powerful and flexible framework that streamlines the setup necessary to model and retrieve high-resolution spectra.

Statement of need

Calculating a spectrum using pRT's core object `Radtrans` is a two-step process in which the user first instantiates the object, giving parameters that control the loading of opacities. The second step is for the user to call one of the `Radtrans` function, giving "spectral" parameters such as the temperatures or the mass fractions, that will be used in combination with the loaded opacities to generate the spectrum.

However, these two steps are by themselves often insufficient to build a spectrum in a real-life scenario. The spectral parameters may individually rely on arbitrarily complex models requiring their own parameters (e.g., the (Guillot, 2010) temperature profile), and may depend on each other. For example, getting mass fractions from equilibrium chemistry requires to know the temperature profile, and the mean molar mass requires to know the mass fractions. Common operations such as convolving the spectrum, scaling it to stellar flux, or more specifically for high-resolution spectra, Doppler-shiftting the spectrum and including the transit effect, must be done on the `Radtrans`-generated spectrum in post-process. Finally, using a retrieval requires to code a "retrieval model" including all the steps described above. This induces, especially for first-time users, a significant setup cost. The alternative is to use one of pRT's built-in models, but this lacks flexibility.

The `SpectralModel` object extends the base capabilities of the petitRADTRANS package by providing a standardized but flexible framework for spectral calculations. It has been especially designed to effectively erase the setup cost of modelling the spectral Doppler-shift, the transit effect, and of implementing the preparing step necessary for ground-based observations analysis.

The Journal of Open Source Software

`SpectralModel` is also interfaced with pRT's `retrieval` module (Nasedkin et al., 2024), and as such is an easy-to-use tool to perform atmospheric retrievals.

The combination of ease-of-use and flexibility offered by `SpectralModel` makes it a powerful tool for high-resolution (but also low-resolution) atmospheric characterisation. With the upcoming first light of a new generation of ground based telescopes, such as the Extremely Large Telescope, `SpectralModel` makes petitRADTRANS ready for the new scientific discoveries that will unveil in the next era of high-resolution observations.

The `SpectralModel` object

Features

Spectral parameter calculation framework

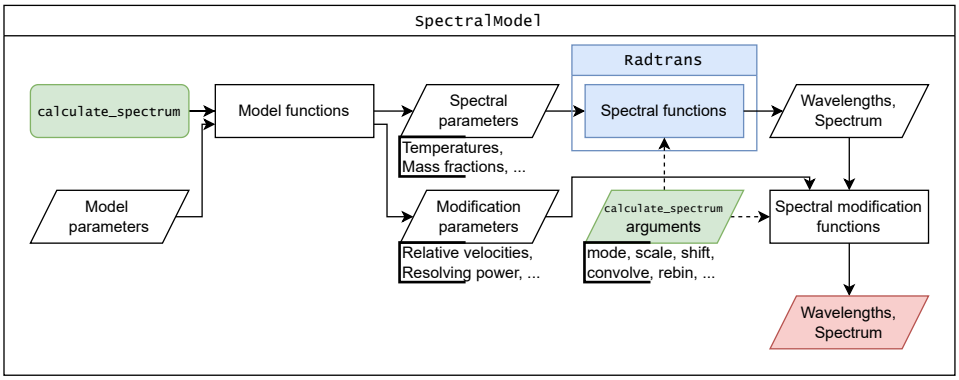

Figure 1: Flowchart of `SpectralModel.calculate_spectrum` function.

`SpectralModel` provides a framework to automatise the calculation of the spectral parameters. Each spectral parameter is linked (using a `dict`) to a function, called here "model function", which calculates its value. This feature can be extended to the parameters required for these functions, and so on. Before calculating spectra, the functions execution order is automatically determined through a topological sorting algorithm (Kahn, 1962). `SpectralModel` comes with built-in functions (Blain et al., 2024) for all the spectral parameters, so that the object can be used "out-of-the box". Parameters that ultimately do not depend on any function are called "model parameters", and must be given during instantiation.

In addition, `SpectralModel` provides built-in functions (Blain et al., 2024) to scale, convolve, Doppler-shift, rebin, include planet transit effect, and prepare a spectrum after it has been calculated. Similarly to model functions, these "spectral modification functions" must be given, if used, their own model parameters during instantiation.

The spectral calculation is done within the `calculate_spectrum` function (see Figure 1). The spectral mode (emission or transmission), as well as which of the spectral modification to activate (i.e. only scaling, or both convolving and rebinning, etc.), are controlled through the function's arguments ("spectral modification parameters").

Automatic optimal wavelength range calculation

High-resolution spectra require high-resolution opacities, which, when loaded, can take a lot of Random-Access Memory (RAM). For example, loading pRT's `1H2-160__POKAZATEL` line-by-line line list ($\mathcal{R} = 10^6$) between 1 and 2 μm takes 804 MB[1] of RAM. Moreover, it is not unusual

[1] According to `numpy.ndarray.nbytes`.

for a model to incorporate multiple species opacities. Fast retrievals in pRT are also performed in parallel, using multiple processes on distributed memory. Loaded opacities RAM usage is thus the amount of bytes taken by one species on the required wavelength range, times the number of species, time the number of processes. Using too many processes can thus overload hardware RAM, limiting retrieval speed.

A way to slightly reduce this memory usage is to load exactly the wavelength range required for an analysis, instead on relying on manual inputs. This task is complicated in high-resolution retrievals due to parameters influencing the Doppler-shift (that is, the radial velocity semi-amplitude K_p, the rest frame velocity shift $Vrest$, and the mid transit time offset T_0) being retrieved. `SpectralModel` comes with a class method `with_velocity_range`, which takes into account the (uniform) prior range of these parameters to automatically calculate the optimal wavelength range to load.

Interface with the `retrieval` module

The `Retrieval` object has been extended to support spectra with up to 3 dimensions, intended to be order, exposure, and wavelength. It now also has a class method that instantiates a `Retrieval` object from a `Data` object. The `Data` object has also been extended in two ways: it now allows taking directly data as arrays, instead of requiring an ASCII file, and allows taking directly `Radtrans` (or by extension `SpectralModel`) objects, instead of generating a new one during a `Retrieval` instance.

In addition, `SpectralModel`'s model parameters and spectral modification functions can be advantageously used to simplify the retrieval setup compared to `Radtrans`'s. This removes the need for several steps:

- building the `RetrievalConfig` object, as this has been be automated,
- declaring the fixed parameters, as all model parameters that are not retrieved parameters are *de facto* fixed parameters,
- writing the retrieval model function, as it is given by the `SpectralModel` itself.

Ground-based high-resolution spectra contain telluric and stellar lines that must be removed. This is usually done with a "preparing" pipeline (also called "detrending" or "pre-processing" pipeline). To this end, a new `retrieval.preparing` sub-module has been implemented, containing the "Polyfit" pipeline (Blain et al., 2024) and the "SysRem" pipeline (Tamuz et al., 2005). To perform a retrieval when the data are prepared with "Polyfit", the forward model be prepared in the same way (Blain et al., 2024). This forward model preparation step can be activated when calculating the spectrum with `SpectralModel`.

High-resolution data simulation

Lorem.

Workflows

Spectra calculation

Calculating spectra with `SpectralModel` is done in two steps:

1. Instantiation: similarly to `Radtrans`, this step is done to load the opacities, and thus requires the same parameter as a `Radtrans` instantiation. In addition, the user can provide model parameters, that will give the spectral parameters and the modification parameters. Finally, a custom `dict` can be given if the user desires to use different functions than the built-in ones.
2. Calculation: spectral calculation is done with a unique function. The spectrum type (emission or transmission), as well as modification flags (for scaling, Doppler-shifting, etc.) are given as arguments.

Retrievals

Retrieving spectra with `SpectralModel` is done in seven steps:

1. Loading the data,
2. For high-resolution ground-based data: preparing the data,
3. Setting the retrieved parameters, this is done by filling a `dict`,
4. Setting the forward model, by instantiating a `SpectralModel` object,
5. Instantiating a `Data` object with the `SpectralModel` dedicated function,
6. Instantiating a `Retrieval` object from the previously built `Data` object,
7. Running the retrieval.

In addition, a new corner plot function, based on the `corner` package (Foreman-Mackey, 2016), has been implemented to ease the representation of the retrieval results with this framework.

The petitRADTRANS 3 update

Along with `SpectralModel`, major changes has been made to pRT. The changes focus on optimisations (both for speed and RAM usage) for high-resolution spectra computing, but this also impacts the correlated-k part of the code (see Table 1). To speed-up "input data" (opacities, pre-calculated equilibrium chemistry table, star spectra table) loading times, pRT's loading system has been overhauled and the loaded files have been converted from a mix of Fortran unformatted files and HDF5 files (the later being for correlated-k opacities only), to fully HDF5. Opacities now also follow an extended Exo-Mol naming and structure convention. The package's code has also been rationalised, clarified, and refactored. Finally, several quality-of-life features (e.g., requested opacities are now automatically downloaded from the project's Keeper library).

Test	pRT 2.7.7 time (s)	pRT 3.1.0 time (s)	pRT 2.7.7 RAM (MB)	pRT 3.1.0 RAM (MB)
Opacity loading, `'c-k'`	3.2	0.9	–	–
Opacity loading, `'lbl'`	6.3	0.4	–	–
Emission, `'c-k'`	6.4	5.2	2428	1472
Emission, `'lbl'`	7.8	4.4	3929	2643
Transmission, `'c-k'`	1.2	0.6	992	757
Transmission, `'lbl'`	6.6	3.1	3929	2230

- Times are measured using the `cProfile` Python standard library, from the average of 7 runs.
- "RAM" is the peak RAM usage as reported by the `tracemalloc` Python standard library.
- `'c-k'`: using correlated-k opacities (CH_4 and H_2O), from 0.3 to 28 μm.
- `'lbl'`: using line-by-line opacities (CO and H_2O), from 0.9 to 1.2 μm.
- All spectra calculations are done using 100 pressure levels. Emission scattering is activated in `'c-k'` mode.
- Results obtained on Debian 12.5 (WSL2), CPU: AMD Ryzen 9 3950X @ 3.50 GHz.

Acknowledgements

Lorem.

References

Blain, D., Sánchez-López, A., & Mollière, P. (2024). A formally motivated retrieval framework applied to the high-resolution transmission spectrum of HD 189733 b. *The Astronomical Journal*, *167*(4), 179. https://doi.org/10.3847/1538-3881/ad2c8b

Foreman-Mackey, D. (2016). Corner.py: Scatterplot matrices in python. *Journal of Open Source Software*, *1*(2), 24. https://doi.org/10.21105/joss.00024

Guillot, T. (2010). On the radiative equilibrium of irradiated planetary atmospheres. *Astronomy & Astrophysics*, *520*, A27. https://doi.org/10.1051/0004-6361/200913396

Kahn, A. B. (1962). Topological sorting of large networks. *Commun. ACM*, *5*(11), 558–562. https://doi.org/10.1145/368996.369025

Mollière, P., Wardenier, J. P., Boekel, R. van, Henning, Th., Molaverdikhani, K., & Snellen, I. A. G. (2019). petitRADTRANS: A Python radiative transfer package for exoplanet characterization and retrieval. *Astronomy & Astrophysics*, *627*, A67. https://doi.org/10.1051/0004-6361/201935470

Nasedkin, E., Mollière, P., & Blain, D. (2024). Atmospheric retrievals with petitRADTRANS. *Journal of Open Source Software*, *9*(96), 5875. https://doi.org/10.21105/joss.05875

Tamuz, O., Mazeh, T., & Zucker, S. (2005). Correcting systematic effects in a large set of photometric light curves. *Monthly Notices of the Royal Astronomical Society*, *356*(4), 1466–1470. https://doi.org/10.1111/j.1365-2966.2004.08585.x