RAPOC Documentation

Release 1.0.8

Lorenzo V. Mugnai

Jan 13, 2023

CONTENTS:

Developed by Lorenzo V. Mugnai and Darius Modirrousta-Galian, the RAPOC code is the product of a collaboration between Sapienza Università di Roma, Università degli Studi di Palermo and INAF - Osservatorio Astronomico di Palermo.

RAPOC uses molecular absorption measurements (i.e. wavelength-dependent opacities) to calculate Rosseland and Planck mean opacities that are commonly used in atmospheric modelling.

RAPOC is designed to be simple, straightforward, and easily incorporated into other codes. It is completely written in Python and documented with docstrings.

CHAPTER

ONE

INSTALLATION & UPDATES

1.1 Installing from Pypi

RAPOC can be installed from the Pypi repository with the following script:

```
pip install rapoc
```

1.2 Installing from git

RAPOC may also be cloned from the main git repository:

```
git clone https://github.com/ExObsSim/Rapoc-public.git
```

The next step is to move into the *RAPOC* folder:

```
cd /your_path/Rapoc
```

Then:

```
pip install .
```

To check if one has the correct setup:

```
python -c "import rapoc"
```

1.3 Uninstall Rapoc

RAPOC is installed in your system as a standard python package: to uninstall it completely:

```
pip uninstall rapoc
```

1.4 Update Rapoc

1.4.1 Update from Pypi

To update *RAPOC* to the latest stable version:

```
pip install rapoc -upgrade
```

1.4.2 Update from source

One can download or pull a newer version of *RAPOC* over the old one, replacing all modified data.

Then one has to place themselves inside the installation directory within the console:

```
cd /your_path/Rapoc
```

RAPOC can now be updated:

```
pip install . --upgrade
```

or simply:

```
pip install .
```

CHAPTER

TWO

USER GUIDE

This guide is an instruction manual on the *RAPOC* (Rosseland And Planck Opacity Converter) tool and how it can be included in one's own code.

For a quickstart, please refer to the python notebook inside the *examples* directory. If *RAPOC* has been installed from Pypi, an example notebook can be found on the GitHub repository.

2.1 Input

Everything inside *RAPOC* is managed by the `Model` class. The only input required by *RAPOC* is the opacity data. Opacity data can be directly downloaded from dedicated repositories such as ExoMol or, instead, by a custom made Python dictionary. In this guide both approaches will be explored:

2.1.1 Using ExoMol (easier)

This is the most straightforward approach. Note that *RAPOC* only accepts data with the TauRex format. Once the opacities have been downloaded from ExoMol, *RAPOC* will load them using the `ExoMolFileLoader`. For practice, we use the TauRex formatted opacities for water. However, if other molecules are required then they can be downloaded here.

2.1.2 Using Dace

This is another straightforward approach. The opacities can be downloaded from Dace as binary files. *RAPOC* will load them using the `DACEFileLoader` by pointing to the directory where the binary files (.bin) are stored.

2.1.3 Python dictionary (harder)

To load data from a Python dictionary, the code uses the `DictLoader` class. The user may also refer to its documentation. To summarise, the python dictionary must have the layout described in the *Input data layout*.

Table 1: Input data layout

Information	data format	supported keys names
molecular name	string	mol
pressure grid in $[Pa]$	list or np.array	p, P, pressure, pressure_grid
temperature grid $[K]$	list or np.array	t, T, temperature, temperature_grid
wavenumber grid $[1/cm]$	list or np.array	wn, wavenumber, wavenumbers, wavenumbers_grid, wavenumber_grid
opacities $[m^2/kg]$	list or np.array	opacities

Every value in the dictionary can be assigned units by using the *units* `astropy.units.Quantity`.

2.2 The model

In this section we will assume that the data has been downloaded from ExoMol and that the *input_file* variable is a string pointing to that file.

Once the data is ready, it may loaded into the *Model* class. After this, two classes can be produced, one for Rosseland and the other for Planck mean opacities: *Rosseland*, *Planck*. These models can be initialised as such:

```
from rapoc import Rosseland, Planck

ross = Rosseland(input_data=input_file)
plan = Planck(input_data=input_file)
```

Normally, the molecule name is automatically extracted from the input data. However, if the molecule name is not present in the input data or if any errors occur, it can be manually set using the *molecule* keyword argument:

```
ross = Rosseland(input_data=input_file, molecule='H2O')
plan = Planck(input_data=input_file, molecule='H2O')
```

2.2.1 Calculating the mean opacities

For purely illustrative purposes, a temperature (T) of 1000 K with a pressure (P) of 1000 Pa in the wavelength range of 1-10 micron will be used. Therefore, the first step is to define the temperature, pressure and wavelength range:

```
import astropy.units as u

P = 1000.0 *u.Pa
T = 1000.0 *u.K
wl = (1 * u.um, 10 * u.um)
```

Due to the units provided by the `astropy.units.Quantity` module, *RAPOC* can handle the conversions automatically. To perform the opacity calculations the following function is used: *estimate()*

```
r_estimate = ross.estimate(P_input = P, T_input=T, band=wl, mode='closest')

0.00418 m2 / kg
```

```
p_estimate = plan.estimate(P_input = P, T_input=T, band=wl, mode='closest')

17.9466 m2 / kg
```

The investigated wavelength range may also be expressed as a wavenumbers range or frequencies range. This is achieved by attaching the corresponding units.

There are two estimation modes available in *RAPOC*:

- *closest*: the code estimates the mean opacity for the closest pressure and temperature values found within the input data grid.
- *linear* or *loglinear*: the code estimates the mean opacity by performing an interpolation that makes use of the `scipy.interpolate.griddata()`.

In the second case, *RAPOC* needs to first build a *map()* of the mean opacities in the input pressure and temperature data grid. This may be slow. To make this process faster, *RAPOC* will continue to reuse this map until the user asks for estimates in another wavelength range. Only then would *RAPOC* produce a new map.

RAPOC can also perform estimates of multiple temperatures and pressures at once (e.g. lists and arrays):

```
P = [1, 10 ] *u.bar
T = [500.0, 1000.0, 2000.0] *u.K
wl = (1 * u.um, 10 * u.um)

r_estimate = ross.estimate(P_input = P, T_input=T, band=wl, mode='closest')

[[0.00094728 0.03303942 0.17086334]
 [0.00484263 0.08600765 0.21673218]] m2 / kg
```

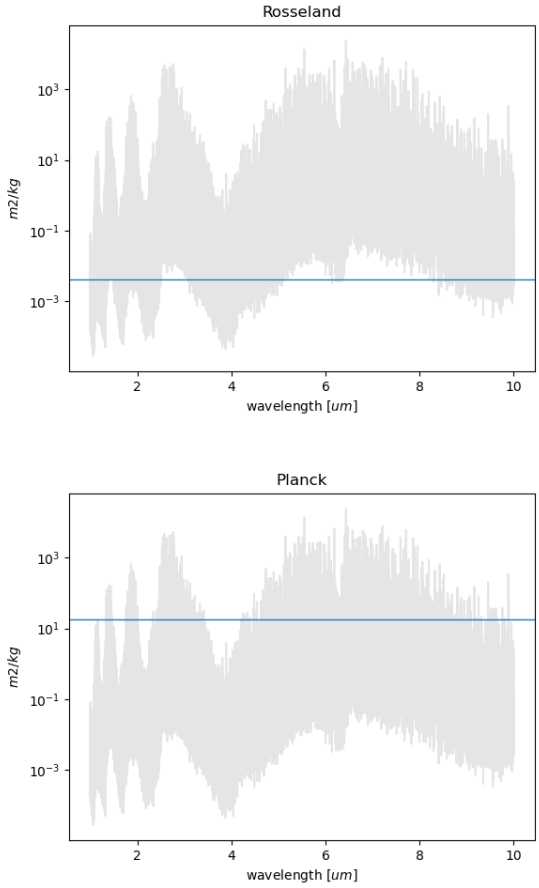
```
p_estimate = plan.estimate(P_input = P, T_input=T, band=wl, mode='closest')

[[27.18478963 17.36108261  8.57782242]
 [27.17086583 17.78923892  8.8249388 ]] m2 / kg
```

2.2.2 Build some plots

RAPOC can build plots of the calculated Rosseland and Planck mean opacities by using *estimate_plot()*.

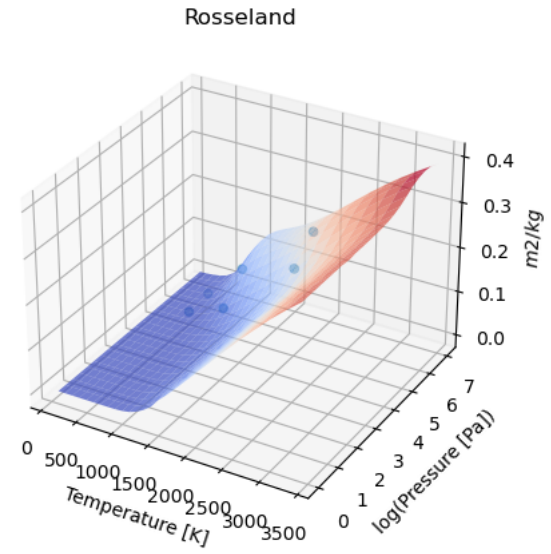
If only a single value has been produced, then to produce plots one can use the following script:

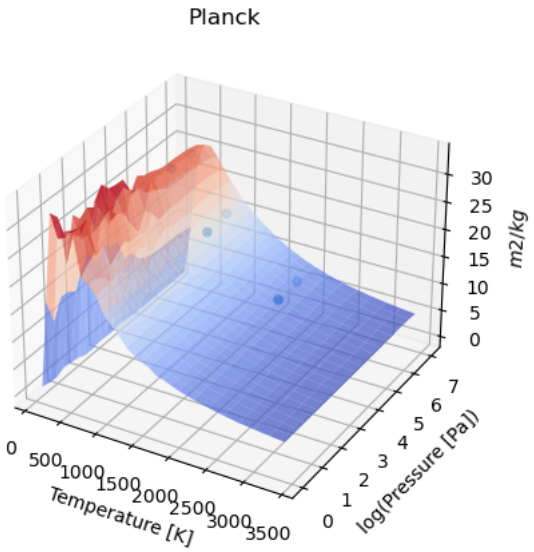
```
P = 1000.0 *u.Pa
T = 1000.0 *u.K
wl = (1 * u.um, 10 * u.um)

fig0, ax0 = ross.estimate_plot(P_input = P, T_input=T, band=wl, mode='closest')
fig1, ax1 = plan.estimate_plot(P_input = P, T_input=T, band=wl, mode='closest')
```



If instead multiple opacities have been calculated, then then the script should be:

```
P = [1, 10 ] *u.bar
T = [500.0, 1000.0, 2000.0] *u.K
wl = (1 * u.um, 10 * u.um)

fig0, ax0 = ross.estimate_plot(P_input = P, T_input=T, band=wl, mode='closest')
fig1, ax1 = plan.estimate_plot(P_input = P, T_input=T, band=wl, mode='closest')
```





Notwithstanding, to force the production of a *map_plot()* with a single value, the following should be used `force_map=True`

2.3 Include Rayleigh scattering

2.3.1 Initialising Rayleigh data

RAPOC can estimate the Rayleigh scattering absorption for a list of atoms, using the tables and equations in Chapter 10 of David R. Lide, ed., CRC Handbook of Chemistry and Physics, Internet Version 2005, hbcponline, CRC Press, Boca Raton, FL, 2005.

To compute the Rayleigh scattering mean opacities, the first step is to produce the Rayleigh data set. This can be done initialising the *Rayleigh* class. This class is similar to a *FileLoader* class: it can be injected into a *Model* as input data, and therefore used as any other input data to estimate the mean opacities.

To initialise the *Rayleigh* it is needed the atom and the wavenumber grid to use to sample the Rayleigh scattering. In the following example we estimate the scattering data for hydrogen in a simple wavenumber grid: 100000, 10000, 1000 cm-1

```
from rapoc import Rayleigh
rayleigh = Rayleigh('H', wavenumber_grid=[100000, 10000, 1000])
```

Another way to initialise the *Rayleigh* is to use an already initialised *Model*. This can be useful is the Rayleigh scattering is to be used along with other absorptions. Let's assume that a *Planck* class has been initialised already. That model can be used to produce a Rayleigh scattering dataset sampled at the same wavenumbers grid.

```
plan = Planck(input_data=input_file)
rayleigh = Rayleigh('H', model=plan)
```

Once the wavenumber grid is produced, either using the *wavenumber_grid* or the *model* keyword, the class produces the opacities data using the *compute_opacities()*.

2.3.2 Estimating Rayleigh mean opacities

The initialised `Rayleigh` can be now used as input data for *Rosseland* and *Planck*

```
ross = Rosseland(input_data=rayleigh)
plan = Planck(input_data=rayleigh)
```

Because the Rayleigh scattering doesn't depend on temperature and pressure, is not sampled in a temperatures and pressures grid as the molecular opacities. Therefore, to estimate its mean opacities the pressure must not be indicated. The temperature, on the contrary, must be indicated because of the Rosseland and Planck equation involving a black body. For the same reason, the estimation mode is forced to *closest*, so should not be indicated by the user. Here is an example

```
T = 1000.0 *u.K
wl = (1 * u.um, 10 * u.um)

r_estimate = ross.estimate(T_input=T, band=wl)

6.725572872420127e-08 m2 / kg
```

```
p_estimate = plan.estimate(T_input=T, band=wl)

6.518714389415313e-09 m2 / kg
```

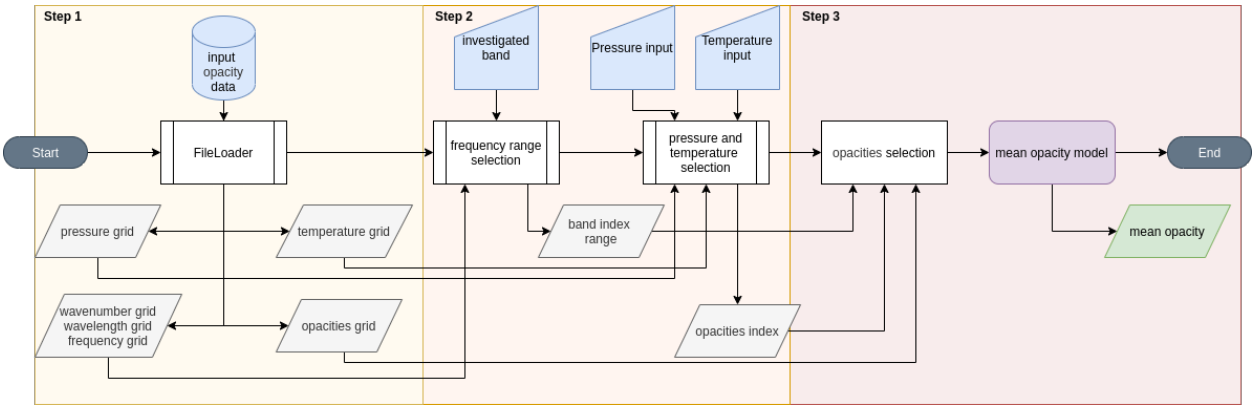
Because, again, the Rayleigh scattering doesn't depend on temperature and pressure, the *map()* is disabled for this kind of data

CHAPTER

THREE

API GUIDE

The *RAPOC* code implements a base `Model` class that loads the input data (consisting of opacities expressed as a function of a grid of pressures, temperatures and a wavelength range, and produces the RPO estimates. From the `Model` class, two other classes are derived that implement the `Rosseland` and `Planck` mean opacity algorithms. The scheme of the code workflow is as follows:



- Yellow box: this is the first step of the code, which consists of reading the input data. In this step *RAPOC* loads the data from the input file. From the input file, *RAPOC* extracts the opacities with their respective pressures, temperatures and wavenumbers.
- Orange box: This is the second step, which involves the analysis of the user's input. Here, the user can specify the wavelength (or wavenumber, or frequency) interval, pressure, and temperature for calculating the RPO values. The code automatically parses the input data and uses it for the final step.
- Red box: in the third and final step, *RAPOC* extracts the specific wavelength-dependent opacities (for the pressure and temperature requested as well as the wavelength, wavenumber or frequency range chosen) and uses them to compute their corresponding RPO values. These values are taken from the complete opacity table that encompasses the total parameter space. The conversion from wavelength-dependent opacities to RPOs is accomplished by using the mean opacity models. This can be either the Rosseland or Planck model. Finally, the code returns the estimate.

3.1 API Content

3.1.1 Planck Model

class rapoc.**Planck**(*input_data*)

Planck model class. This class implements the Planck Opacity model:

$$k = \frac{\int_{\nu_1}^{\nu_2} k_\nu B_\nu(T) d\nu}{\int_{\nu_1}^{\nu_2} B_\nu(T) d\nu}$$

where ν_1 and ν_2 are the investigated frequency band edges, k_ν is the input data opacity at a given frequency, and $B_\nu(T)$ is the black body radiation energy distribution computed at a certain temperature:

$$B_\nu(T) = \frac{2h\nu^3}{c^2} \frac{1}{e^{\frac{h\nu}{k_B T}} - 1}$$

Variables

- **input_data** (*str or dict*) – data file name or data dictionary
- **model_name** (*str*) – model name
- **band** (*tuple*) – investigated band. Is initialised only after the use of `map()`
- **selected_grid** (*astropy.units.Quantity*) – investigated grid. Is initialised only after the use of `map()`
- **mol** (*str*) – molecule name
- **mol_mass** (*astropy.units.Quantity*) – molecular mass
- **pressure_grid** (*astropy.units.Quantity*) – data pressure grid in *si* units
- **temperature_grid** (*astropy.units.Quantity*) – data temperature grid in *si* units
- **wavenumber_grid** (*astropy.units.Quantity*) – data wavenumber grid
- **opacities** (*astropy.units.Quantity*) – data opacities grid in *si* units
- **frequency_grid** (*astropy.units.Quantity*) – data frequency grid
- **wavelength_grid** (*astropy.units.Quantity*) – data wavelength grid

Parameters

input_data (*str or dict*) – data file name or input_data dictionary. If the input is a *str*, the correspondent loader is used (*ExoMolFileLoader*) if the file format is supported. If is *dict*, then *DictLoader* is used.

Raises

IOError – if data format is not supported:

Examples

First we prepare a data dictionary:

```
>>> import numpy as np
>>> data_dict = {'mol': 'H2O',
>>>              'pressure': np.array([1.00000000e+00, 1.00000000e+01, 1.
↪00000000e+02, 1.00000000e+03,
>>>                                    1.00000000e+04, 1.00000000e+05, 1.
↪00000000e+06, 1.00000000e+07]),
>>>              'temperature': np.array([500, 1000, 1500, 2000, 3000]),
>>>              'wavenumber': np.array([100000, 1000])}
>>> ktab = np.ones((data_dict['pressure'].size,data_dict['temperature'].size,data_
↪dict['wavenumber'].size,))
>>> data_dict['opacities'] = ktab
```

Let's build the Planck method using an Exomol file as input

```
>>> from rapoc import Planck
>>> dict_data = 'example.TauREx.h5'
>>> model = Planck(input_data=data_dict)
```

Now the model is ready to be used

opacity_model(*opacities*, *nu*, *T_input*)

This function computes the Planck Opacity model:

$$k = \frac{\int_{\nu_1}^{\nu_2} k_\nu B_\nu(T) d\nu}{\int_{\nu_1}^{\nu_2} B_\nu(T) d\nu}$$

where ν_1 and ν_2 are the investigated frequency band edges, k_ν is the input data opacity at a given frequency, and $B_\nu(T)$ is the black body radiation energy distribution computed at a certain temperature:

$$B_\nu(T) = \frac{2h\nu^3}{c^2} \frac{1}{e^{\frac{h\nu}{k_B T}} - 1}$$

Parameters

- **opacities** (*np.array*) – opacity array. Has the same dimension of the frequency grid *nu*
- **nu** (*np.array*) – frequency grid
- **T_input** (*float*) – temperature

Returns
mean opacity computed from the model

Return type
float

3.1.2 Rosseland Model

class rapoc.**Rosseland**(*input_data*)

Rosseland model class. This class implements the Rosseland Opacity model:

$$\frac{1}{k} = \frac{\int_{\nu_1}^{\nu_2} k_\nu^{-1} u(\nu, T) d\nu}{\int_{\nu_1}^{\nu_2} u(\nu, T) d\nu}$$

where ν_1 and ν_2 are the investigated frequency band edges, k_ν is the input data opacity at a given frequency, and $u(\nu, T)$ is the black body temperature derivative:

$$u(\nu, T) = \frac{\partial B(\nu, T)}{\partial T} = 2\frac{h^2\nu^4}{c^2 k_b}\frac{1}{T^2}\frac{e^{\frac{h\nu}{k_b T}}}{(e^{\frac{h\nu}{k_b T}} - 1)^2}$$

Variables

- **input_data** (*str or dict*) – data file name or data dictionary
- **model_name** (*str*) – model name
- **band** (*tuple*) – investigated band. Is initialised only after the use of `map()`
- **selected_grid** (*astropy.units.Quantity*) – investigated grid. Is initialised only after the use of `map()`
- **mol** (*str*) – molecule name
- **mol_mass** (*astropy.units.Quantity*) – molecular mass
- **pressure_grid** (*astropy.units.Quantity*) – data pressure grid in *si* units
- **temperature_grid** (*astropy.units.Quantity*) – data temperature grid in *si* units
- **wavenumber_grid** (*astropy.units.Quantity*) – data wavenumber grid
- **opacities** (*astropy.units.Quantity*) – data opacities grid in *si* units
- **frequency_grid** (*astropy.units.Quantity*) – data frequency grid
- **wavelength_grid** (*astropy.units.Quantity*) – data wavelength grid

Parameters

input_data (*str or dict*) – data file name or input_data dictionary. If the input is a *str*, the correspondent loader is used (*ExoMolFileLoader*) if the file format is supported. If is *dict*, then *DictLoader* is used.

Raises

IOError – if data format is not supported:

Examples

Let's build the Rosseland method using an Exomol file as input

```
>>> from rapoc import Rosseland
>>> exomolFile = 'example.TauREx.h5'
>>> model = Rosseland(input_data=exomolFile)
```

Now the model is ready to be used

opacity_model(*opacities*, *nu*, *T_input*)

This function computes the Rosseland Opacity model:

$$\frac{1}{k} = \frac{\int_{\nu_1}^{\nu_2} k_\nu^{-1} u(\nu, T) d\nu}{\int_{\nu_1}^{\nu_2} u(\nu, T) d\nu}$$

where ν_1 and ν_2 are the investigated frequency band edges, k_ν is the input data opacity at a given frequency, and $u(\nu, T)$ is the black body temperature derivative:

$$u(\nu, T) = \frac{\partial B(\nu, T)}{\partial T} = 2\frac{h^2\nu^4}{c^2 k_b}\frac{1}{T^2}\frac{e^{\frac{h\nu}{k_b T}}}{(e^{\frac{h\nu}{k_b T}} - 1)^2}$$

Parameters

- **opacities** (*np.array*) – opacity array. Has the same dimension of the frequency grid *nu*
- **nu** (*np.array*) – frequency grid
- **T_input** (*float*) – temperature

Returns
mean opacity computed from the model

Return type
float

3.1.3 Rayleigh Model

class rapoc.**Rayleigh**(*atom*, *wavenumber_grid=None*, *model=None*)

The Rayleigh class estimates the Rayleigh scattering for the indicated atom using the equations from Chapter 10 of David R. Lide, ed., CRC Handbook of Chemistry and Physics, Internet Version 2005, hbcponline, CRC Press, Boca Raton, FL, 2005. These values won't depend on pressure or temperature and therefore only a wavenumbers grid is needed to produce them. The wavenumbers grid can be given by the user or loaded from an already initialised *Model* class (as *Rosseland* or PLanck). The latter solution might be of help in using the Rayleigh scattering along with other opacities.

Parameters

- **atom** (*str*) – name of the considered atom
- **wavenumber_grid** (*list or numpy.array or astropy.units.Quantity*) – data wavenumber grid
- **model** (*Model*) – built model to use to load the wavenumbers grid.

Examples

First we prepare a data wavenumbers grid, then we can produce the Rayleigh data:

```
>>> rayleigh = Rayleigh('H', wavenumber_grid=[100000, 10000, 1000])
```

if we already have a molecular model loaded, as example built from a datafile, we can use it to initialise the Rayleigh class:

```
>>> input_data = Rosseland(input_data=exomol_file)
>>> rayleigh = Rayleigh(atom='Na', model=input_data)
```

Now the Rayleigh data are ready to be used as input data for any RAPOC *Model* class:

```
>>> pl = Planck(input_data=rayleigh)
>>> rs = Rosseland(input_data=rayleigh)
```

compute_opacities()

It computes the opacities for Rayleigh scattering as described in David R. Lide, ed., CRC Handbook of Chemistry and Physics, Internet Version 2005, hbcponline, CRC Press, Boca Raton, FL, 2005. chapter 10 table 1:

$$\alpha(\lambda) = \frac{128\pi^5}{3\lambda^4}\frac{a^2}{m}$$

where a is depending on the atom polarizabiility listed in table 2 chapter 10 of CRC handbook, and m is the atom mass.

Returns
returns the estimated opacity sampled at the indicated wavenumbers.

Return type
astropy.units.Quantity

read_content()

Reads the class content and returns the needed valued for the opacity models.

Returns

- *str* – molecule name
- *astropy.units.Quantity* – molecular mass
- *astropy.units.Quantity* – data pressure grid in si units
- *astropy.units.Quantity* – data temperature grid in si units
- *astropy.units.Quantity* – data wavenumber grid
- *astropy.units.Quantity* – data opacities grid in si units

3.1.4 Base Model

class rapoc.models.model.**Model**(*input_data*)

Base model class.

Variables

- **input_data** (*str or dict*) – data file name or data dictionary
- **model_name** (*str*) – model name
- **band** (*tuple*) – investigated band. Is initialised only after the use of map()
- **selected_grid** (*astropy.units.Quantity*) – investigated grid. Is initialised only after the use of map()
- **mol** (*str*) – molecule name

- **mol_mass** (*astropy.units.Quantity*) – molecular mass
- **pressure_grid** (*astropy.units.Quantity*) – data pressure grid in *si* units
- **temperature_grid** (*astropy.units.Quantity*) – data temperature grid in *si* units
- **wavenumber_grid** (*astropy.units.Quantity*) – data wavenumber grid
- **opacities** (*astropy.units.Quantity*) – data opacities grid in *si* units
- **frequency_grid** (*astropy.units.Quantity*) – data frequency grid
- **wavelength_grid** (*astropy.units.Quantity*) – data wavelength grid

Parameters
input_data (*str or dict*) – data file name or input_data dictionary. If the input is a *str*, the correspondent loader is used (*ExoMolFileLoader*) if the file format is supported. If is *dict*, then *DictLoader* is used.

Raises
IOError – if data format is not supported:

Examples

For the sake of semplicity let's assume we use an ExoMol file as input

```
>>> from rapoc.models import Model
>>> exomolFile = 'example.TauREx.h5'
>>> model = Model(input_data=exomolFile)
```

opacity_model(*opacities*, *nu*, *T_input*)

Computes the mean opacity in the investigated range.

Parameters

- **opacities** (*np.array*) – opacity array. Has the same dimension of the frequency grid *nu*
- **nu** (*np.array*) – frequency grid
- **T_input** (*float*) – temperature

Returns
mean opacity computed from the model

Return type
float

extract_opacities(*T_input*, *band*, *P_input=None*, *units='si'*)

Returns the input_data opacities for given pressure, temperature and band.

Parameters

- **T_input** (*float or np.array*) – temperature to use for the estimation. This should be a *astropy.units.Quantity* with specified units, if not *K* are assumed as units. Can either be a single value or an array.
- **band** (*tuple*) – this should be a tuple of *astropy.units.Quantity* indicating the edges of the band to use in the estimation. The units used reveal the intention to work in wavelengths, wavenumbers or frequencies.

- **P_input** (*float or np.array*) – pressure to use for the estimation. This should be a *astropy.units.Quantity* with specified units, if not *Pa* are assumed as units. Can either be a single value or an array. It can be None in the case of data not depending on pressure, as Rayleigh scattering. Default is None.
- **units** (*str or astropy.units, optional*) – indicates the output units system. If *si* the opacity is returned as m^2/kg, if *cgs* is returned as cm^2/g. Instead of a string, you can also indicate the units using astropy.units. Units is *si* by default.

Returns

- *astropy.units.Quantity* – returns the estimated opacity for the pressures and temperatures indicated. If only one pressure and temperature are indicated, it returns a single Quantity. If *n* pressures and *m* temperatures are indicated, it return a Quantity array of dimensions (n,m)
- *list* – list of wavenumber (or wavelength or frequency) index relative to the investigated band

estimate(*T_input*, *band*, *P_input=None*, *mode='closest'*, *units='si'*)

It estimates the model opacity in the indicated pressure and temperature grid for the indicated band, using the *opacity_model()*.

Parameters

- **T_input** (*float or np.array*) – temperature to use for the estimation. This should be a *astropy.units.Quantity* with specified units, if not *K* are assumed as units. Can either be a single value or an array.
- **band** (*tuple*) – this should be a tuple of *astropy.units.Quantity* indicating the edges of the band to use in the estimation. The units used reveal the intention to work in wavelengths, wavenumbers or frequencies.
- **P_input** (*float or np.array*) – pressure to use for the estimation. This should be a *astropy.units.Quantity* with specified units, if not *Pa* are assumed as units. Can either be a single value or an array. It can be None in the case of data not depending on pressure, as Rayleigh scattering. Default is None.
- **mode** (*str, optional*) – indicates the estimation mode desired. If *closest* the output will be the opacity computed from the data at the nearest pressure on the data grid to the indicated one, and at the nearest temperature on the data grid to the indicated one. If {'linear', 'loglinear'} it's used the `scipy.interpolate.griddata()` with the indicated *kind*. To interpolate a map of opacities over the data grid is computed first using *map()*. Mode is *closest* by default.
- **units** (*str or astropy.units, optional*) – indicates the output units system. If *si* the opacity is returned as m^2/kg, if *cgs* is returned as cm^2/g. Instead of a string, you can also indicate the units using astropy.units. Units is *si* by default.

Returns

returns the estimated opacity for the pressures and temperatures indicated. If only one pressure and temperature are indicated, it returns a single Quantity. If *n* pressures and *m* temperatures are indicated, it return a Quantity array of dimensions (n,m)

Return type

astropy.units.Quantity

Raises

- **NotImplementedError:** – if the indicated mode is not supported
- **astropy.units.UnitConversionError:** – if the input units cannot be converted to *si*

- **AttributeError:** – if the band cannot be interpreted as wavelength, wavenumber or frequency

estimate_plot(*T_input*, *band*, *P_input=None*, *mode='closest'*, *force_map=False*, *fig=None*, *ax=None*, *yscale='log'*, *xscale='linear'*, *grid='wavelength'*)

Produces a plot of the estimates (produced with *estimate()*), comparing the raw data with the result. If a single estimate is to be plotted, this method produces a plot of raw opacities vs wavelength from the opacities (in grey) and the mean opacity estimated. If multiple estimates are be plotted, it produces a 3D plot, with the surface of mean opacity vs temperature and pressure from the data grid (using *map_plot()*) and the interpolated data superimposed.

Parameters

- **T_input** (*float or np.array*) – temperature to use for the estimation. This should be a *astropy.units.Quantity* with specified units, if not *K* are assumed as units. Can either be a single value or an array.
- **band** (*tuple*) – this should be a tuple of *astropy.units.Quantity* indicating the edges of the band to use in the estimation. The units used reveal the intention to work in wavelengths, wavenumbers or frequencies.
- **P_input** (*float or np.array*) – pressure to use for the estimation. This should be a *astropy.units.Quantity* with specified units, if not *Pa* are assumed as units. Can either be a single value or an array. It can be None in the case of data not depending on pressure, as Rayleigh scattering. Default is None.
- **mode** (*str, optional*) – indicates the estimation mode desired. If *closest* the output will be the opacity computed from the data at the nearest pressure on the data grid to the indicated one, and at the nearest temperature on the data grid to the indicated one. If {'linear', 'loglinear'} it's used the `scipy.interpolate.griddata()` with the indicated *kind*. Mode is *closest* by default.
- **force_map** (*bool*) – If True a 3D map plot is generate even for a single estimate. Default is *False*.
- **fig** (*matplotlib.pyplot.figure, optional*) – indicates the figure to use. If *None* a new figure is produced. Default is *None*
- **ax** (*matplotlib.pyplot.axis, optional*) – indicates the axis of the figure to use. If *None* a new axis is produced. Default is *None*
- **yscale** (*str*) – y-axis scale to use. Default is *log*
- **xscale** (*str*) – x-axis scale to use. Default is *linear*
- **grid** (*str*) – x-axis grid format. {*wavelength*, *frequency*, *wavenumber*} are supported. Default is *wavelength*.

Returns

- *matplotlib.pyplot.figure, optional* – figure containing the plot
- *matplotlib.pyplot.axis, optional* – axis containing the plot

Raises

KeyError: – if the indicated grid is not supported

map(*band*)

Returns the mean opacity map for the indicated model.

Parameters

band (*tuple*) – this should be a tuple of *astropy.units.Quantity* indicating the edges of the

band to use in the estimation. The units used reveal the intention to work in wavelengths, wavenumbers or frequencies.

Returns
mean opacity map in *si* units

Return type
np.array

Notes

If the map has been already built for the indicated band, then the method returns the previous map. This speeds up the use of the code in external functions.

map_plot(*band*, *fig=None*, *ax=None*)

Produces a 3D-plot of the model mean opacity map built with *map()*.

Parameters

- **band** (*tuple*) – this should be a tuple of *astropy.units.Quantity* indicating the edges of the band to use in the estimation. The units used reveal the intention to work in wavelengths, wavenumbers or frequencies.
- **fig** (*matplotlib.pyplot.figure, optional*) – indicates the figure to use. If *None* a new figure is produced. Default is *None*
- **ax** (*matplotlib.pyplot.axis, optional*) – indicates the axis of the figure to use. If *None* a new axis is produced. Default is *None*

Returns

- *matplotlib.pyplot.figure, optional* – figure containing the plot
- *matplotlib.pyplot.axis, optinal* – axis containing the plot

3.1.5 ExoMol file loader

class rapoc.loaders.**ExoMolFileLoader**(*filename*, *molecule=None*)

File loader for ExoMol file. It works for opacities file in the *TauREx.h5* format. These file can be downloaded from ExoMol website.

Notes

This class is implicitly called by the model when initialised if the matched input is used.

Examples

Let's build the Rosseland method using an Exomol file as input

```
>>> from rapoc import Rosseland
>>> exomolFile = 'example.TauREx.h5'
>>> model = Rosseland(input_data=exomolFile)
```

Now the model is ready to be used

Parameters

- **filename** (*str*) – data file name
- **molecule** (*str*) – forced molecule name

3.1.6 DACE file loader

class rapoc.loaders.**DACEFileLoader**(*filename*, *molecule=None*)

File loader for DACE files. It works for opacities files in the binary format. These files can be downloaded from DACE website.

The scripts used to read the DACE files are inspired by petitRADTRANS.

Notes

This class is implicitly called by the model when initialised if the matched input is used.

Examples

Let's build the Rosseland method using an DACE files as input. The binary files of the desired molecule should be stored in a *dace_dir*.

```
>>> from rapoc import Rosseland
>>> daceFileList = '/dace_dir'
>>> model = Rosseland(input_data=daceFileList)
```

Now the model is ready to be used

Warning: This method reads all the binary data in the indicated folder: the user must be cautios to indicate a directory that contains only the binary files of the desired molecule.

Parameters

- **filename** (*str*) – data file name
- **molecule** (*str*) – forced molecule name

3.1.7 Base file loader

class rapoc.loaders.**DictLoader**(*input_data*)

Dict loader class. The dictionary should contain the following information under certain keys:

information	default units	supported keys
molecular name		mol
pressure grid	Pa	p, P, pressure, pressure_grid
temperature grid	K	t, T, temperature, temperature_grid
wavenumber grid	$1/cm$	wn, wavenumber, wavenumbers, wavenumbers_grid, wavenumber_grid
opacities	m^2/kg	opacities

If the input data has no units attached, the defaults units are assume. If they have units, a simple conversion is performed by the code to match the default values.

Notes

This class is implicitly called by the model when initialised if the matched input is used.

Examples

First we prepare a data dictionary. Please be aware that this are not representative of any molecule.

```
>>> import numpy as np
>>> data_dict = {'mol': 'None',
>>>              'mol_mass':42,
>>>              'pressure': np.array([1.00000000e+00, 1.00000000e+01, 1.
↪00000000e+02, 1.00000000e+03,
>>>                                    1.00000000e+04, 1.00000000e+05, 1.
↪00000000e+06, 1.00000000e+07]),
>>>              'temperature': np.array([500, 1000, 1500, 2000, 3000]),
>>>              'wavenumber': np.array([100000, 1000])}
>>> opac = np.ones((data_dict['pressure'].size,data_dict['temperature'].size,data_
↪dict['wavenumber'].size,))
>>> data_dict['opacities'] = opac
```

Let's build the Planck method using the dictionary as input

```
>>> from rapoc import Planck
>>> model = Planck(input_data=data_dict)
```

Now the model is ready to be used

Parameters
input_data (*dict*) – input_data dictionary

read_content()

Reads the dict content and returns the needed valued for the opacity models.

Returns

- *str* – molecule name
- *astropy.units.Quantity* – molecular mass
- *astropy.units.Quantity* – data pressure grid in si units
- *astropy.units.Quantity* – data temperature grid in si units
- *astropy.units.Quantity* – data wavenumber grid
- *astropy.units.Quantity* – data opacities grid in si units

3.1.8 Base file loader

class rapoc.loaders.loader.**FileLoader**(*filename*, *molecule=None*)

Base file loaders class.

Parameters

- **filename** (*str*) – data file name
- **molecule** (*str*) – forced molecule name

read_content()

Reads the file content and returns the needed values for the opacity models.

Returns

- *str* – molecule name
- *astropy.units.Quantity* – molecular mass
- *astropy.units.Quantity* – data pressure grid in si units
- *astropy.units.Quantity* – data temperature grid in si units
- *astropy.units.Quantity* – data wavenumber grid
- *astropy.units.Quantity* – data opacities grid in si units

CHAPTER

FOUR

CITE

If you use this code or its results, please cite *RAPOC: the Rosseland and Planck opacity converter* by Mugnai L. V. and Modirrousta-Galian D. (submitted).

CHAPTER

FIVE

MEET THE AUTHORS

For more information on the authors, please see their personal websites:

- Lorenzo V. Mugnai
- Darius Modirrousta-Galian

INDEX