Manual of **ARC**
(Algorithm for Rayleigh and Raman Calculations)

Authors: Nikolaos Siomos
Developers: Nikolaos Siomos, Michael Haimerl, Ioannis Binietoglou
Last Update: 03.09.2024
Version: 1.0.0

Table of Contents

1 Installation

ARC can be cloned through git. In order to do so:

- Git must be installed in the computer
- ARC can be cloned from the stable repository:
- Open a terminal and cd to the folder you want to place ARC to
- run: *git clone*

Alternatively you can just download ARC from the link above.

2 Dependencies

The current version of ARC has been developed and tested in python 3.9. The following python packages should be installed:

- numpy
- scipy
- matplotlib

The recommended ways to install the dependencies and run ATLAS is currently either through an Anaconda environment (tested with Miniconda) or with poetry. Of course, if there is already an existing environment with the aforementioned libraries installed, ARC can be imported as a normal python module.

2.1 Poetry installation

The dependencies are automatically collected with poetry as long as the dependency file pyproject.toml is present in the downloaded ATLAS project folder.

An installation example follows for Linux systems.

- Download and install the latest poetry and add it to your path (instructions here https://python-poetry.org/docs/)
- Create and environment for python 3.9 (if not already created by poetry)
- Go to the ARC project folder and install the dependencies e.g. with

  poetry install
- Before running ARC make sure that the python 3.9 environment is activated.

2.2 Anaconda installation

The dependencies for the current version were collected with conda – version 23.3.1

An installation example follows for Linux systems (it should also work for Windows and MAC systems – if problems arise the users are encouraged to provide feedback to the EARLINET forum):

- Download and install the latest Miniconda for python 3.9
- Activate the **base** environment (usually it is auto-activated by default)
- Create a new environment to install the dependencies of ATLAS (let's call it **arc_*box*** here):
  - conda create -n arc_box python=3.9
- Install all dependencies using conda in the **arc_box** environment
  - conda install -n atlas_box numpy scipy matplotlib
- Before running ALTAS make sure that the **arc_box** environment is activated. It is best to close the terminal after installation of the dependencies and open a new one before import ARC

3 Use of ARC

3.1 Basic Usage

ARC is a python module and it is intended to be imported as such. To start using it, the *arc* class within *__arc_main__.py* must be imported, for example:

```
from __arc_main__ import arc
```

If you don't want to include arc in your project folder, then you can provide the arc folder path, where *__arc_main.py* is placed by default, to the system path by using the *sys* module, for example:

```
import sys

sys.path.insert(0, <path_to_arc_directory>)
```

In order to run the tests, the __test_arc__.py script has to be executed, for example run the following command in a terminal within the arc folder, where *__arc_main.py* is placed by default:

```
python __test_arc__.py
```

Calling the *arc* class creates a python object that includes a number variables and 2 methods: *cross_section* and *mldr*. At least the incident_wavelength argument must be provided. The rest of the arguments are mandatory.

Minimum example 1:

```
from __arc_main__ import arc

rrs = arc(incident_wavelength = 355)

cross_section_355 = rrs.cross_section(cross_section_type = 'full')

print(cross_section_355)
```

This returns the Rayleigh scattering cross-section at 355 nm by summing all polarized and depolarized lines.

Minimum example 2:

```
from __arc_main__ import arc

rrb = arc(incident_wavelength = 355, backscattering = True)

bsc_cross_section_355 = rrb.cross_section(cross_section_type = 'full')

mldr_355 = rrb.mldr(mldr_type = 'full')

print(cross_section_355)

print(mldr_355)
```

This returns the Rayleigh backscattering cross-section and the molecular linear depolarization ratio (MLDR) at 355 nm by summing all polarized and depolarized lines.

Default temperature conditions and molar fractions of the 5 major atmospheric gases (N_2, O_2, Ar, CO_2, H_2O) are applied in the examples above. In the next section we will cover the different input and output parameters of arc in more detail.

3.2 Input parameters

The *arc* class accepts the following arguments as input:

incident_wavelength: float

The wavelength of the radiation incedent to the molecules, in air (nm) - Mandatory

temperature: float

The atmospheric temperature (K). Defaults to 288.15 K (15°C)

molar_fractions: dictionary with the gas names as indexes and

Molar fractions of atmospheric gases.

If not provided the following molar fractions will be used by default:

{'N2' : 0.780796, 'O2' : 0.209448, 'Ar' : 0.009339, 'CO2': 0.000416, 'H2O': 0.}

max_rr_lines: int

Maximum rotational quantum number (number of lines considered per branch)

backscattering: bool

If set to True, backscattering cross-sections will be calculated instead of scattering cross sections. Defaults to False

mode: string

Choose among: rotational_raman and vibrational_raman

rotational_raman: Corresponds to elastic and pure rotational Raman

lidar channel applications.

vibrational_raman_N2: Corresponds to N2 vibrational Raman (V = 1)

lidar channel applications.

vibrational_raman_O2: Corresponds to O2 vibrational Raman (V = 1)

lidar channel applications.

filter_parameters: dictionary

Dictionary entries that describe the interference filter (IF) can be included under the following keys:

transmission_shape: str

Use one of: 'Gaussian', 'Lorentzian', 'Square', 'Custom'.

AOI: float

Angle of incidence (AOI) of the incident light with respect to the optical axis of the IF.

Defaults to 0.

ref_index_IF: float

Effective refractive index of the IF. Defaults to 2.

extra_shift: float

A wavelength extra_shift in nanometers that can be applied to a filter in addition to the AoI extra_shift. It will be ignored if the transmission_shape is not 'Custom'

filter_path: str

The path to an ascii file with the transmission function of the interference filter. The file must have 2 columns and no header. The first corresponds to the wavelength scale. The second to the transmission for each wavelength. It will be ignored if the transmission_shape is not 'Custom'

filter_file_delimiter: str

The delimiter used to parse the filter file. It will be ignored if the transmission_shape is not 'Custom' and the filter_path is not provided. Defaults to ' '

filter_file_header_rows: int

The number of header lines to skip when parsing the filter file. It will be ignored if the transmission_shape is not 'Custom' and the filter_path is not provided. Defaults to 0

wavelengths: 1D float array

An array of wavelength values that correspond to the transmission curve of a filter. Cannot be given as input at the same time with filter_path

transmissions: 1D float array

The array of transmission values per wavelength. Cannot be given as input at the same time with filter_path

central_wavelength: float

The central wavelength of the filter. It will be ignored if the transmission_shape is set to 'Custom'

bandwidth: float

The filter bandwidth in nanometers. It will be ignored if the transmission_shape is set to 'Custom'

peak_transmission: float

The maximum transmission value. It will be ignored if the transmission_shape is set to 'Custom'. Defaults to 1.

The use of **incident_wavelength** and **temperature** are quite straightforward.

The **molar_fractions** dictionary includes the molar fraction of each of the major atmospheric gases. For example, by setting the molar fraction of one of the gases to 1 and the rest to 0 the cross-section and mldr of that specific gas will be returned by the *cross_section* and *mldr* methods.

The **max_J** corresponds to the maximum value of the rotational quantum number J. In practice, it denotes the maximum number of pure rotational or ro-vibrational Raman lines simulated.

The **mode** argument declares whether pure rotational Raman or ro-vibrational Raman calculations will be performed. Pure rotational lines are simulated by default. Setting **mode** to either vibrational_raman_N2 or vibrational_raman_O2 changes the vibrational quantum number *v* to 1 and the ro-vibrational lines of N_2 or O_2 are simulated, respectively.

The **backscattering** arguments declares whether scattering or backscattering cross-sections will be returned. Calculation of the MLDR does not make sense for total scattering and will not be calculated if backscattering is set to False.

The **filter_parameters** argument is the most complex one as it is a dictionary that includes all necessary parameters that define the interference filter (if any). Using this argument does not make sense if the **backscattering** argument is not set to True. There currently 3 ways to define a filter in ARC:

1. Assume a theoretical IF

   This can be achieved by setting the transmission_shape to either 'Gaussian', 'Lorentzian', or 'Tophat'. In that case the central_wavelength and bandwidth variables of the filter must be provided. The transmission curve will be generated from a Gaussian, Lorentzian, or Tophat function and the corresponding central_wavelength and bandwidth variables.

2. Read the IF transmission from an ascii file

   This can be achieved by setting the transmission_shape to either 'Custom'. In that case, the filter_path variable must be provided. It corresponds to the absolute path of the filter ascii file. By default the file should be space-separated and contain 2 columns and no header. The first column should correspond to the wavelength in nm and the second to the transmission values per wavelength (not percent). The transmission will be assumed to be 0 for wavelengths larger or smaller than the ones provided in the file. The filter_file_delimiter variable can be used if the delimiter is not space (e.g. comma), and the filter_file_header_rows can be used to skip any header lines. The ascii file should contain no footer.

3. Provide the IF transmission as a numpy array

   Alternatively, if the filter transmission is already loaded in the memory, it can be provided directly as a numpy array by using the wavelengths and transmissions variables. In that case, wavelengths should correspond to an 1D numpy array containing the wavelength axis values in nm and transmissions to an 1D numpy array containing the corresponding transmission function values per wavelengths (not percent).

Sometimes it is desirable to shift a real filter along the wavelength axis. To do so, the extra_shift variable can be provided, shifting the whole curve to the right or left if a positive or negative shift are provided, respectively. This shift does not make sense for theoretical filters. Any additional shift can be included already to the central_wavelength variable. The extra_shift variable will be ignored if the transmission_shape is set to 'Gaussian', 'Lorentzian', or 'Tophat'.

An apparent shift of the filter transmission curve can occur if the angle of incidence of the radiation rays on the IF is not 0. The AOI variable can be used to simulate this effect. Optionally, the effective refractive index of the IF can be provided with the ref_index_IF variable (2. by default), as it can affect the introduced amount of shift.

3.3 Output

There are three was of exporting information from the generated arc object when the *arc* class is called.

1. Variables stored in the arc object

   Many variables are stored directly in the arc object generated by the *arc* class. They can be exported by typing:

   <class_object>.<variable_name>

   The full list of stored variables is included below.

2. The scattering/effective backscattering cross sections using the cross_section method

   Cross-sections can be calculated by typing:

   <class_object>.cross_section(cross_section_type = <selected_type>, normalize = <True/False>)

   The cross-section value depends on all parameters provided as input to the *arc* class. If an IF was defined in the class with the **filter_parameters** argument, the transmission will be applied for the calculation of the effective backscatter cross-section (see the ARC paper). In addition, by setting the normalize variable to True (False by default), the cross-section will be normalized with the filter transmission at the indecent wavelength, which might be desirable for elastic channels (see the ARC paper for more information). A list of all available cross_section_type can be found below.

3. The molecular linear depolarization ratio using the mldr method

   Cross-sections can be calculated by typing:

   <class_object>.mldr(mldr_type = <selected_type>)

   The MLDR value depends on all parameters provided as input to the *arc* class. If an IF was defined in the class with the **filter_parameters** argument, the transmission will be applied for the calculation of the effective backscatter MLDR (see the ARC paper). A list of all available mldr_type can be found below.

Variables stored in the arc object

all_molecules:

A list of strings containing the names of all major gas components

linear_molecules:

A list of strings containing the names of all major gas components that are linear molecules

filter_transmission:

A function that receives the wavelength as input and returns the transmission at that specific wavelength. It also contains the plot method. Typing:

<arc_object>.filter_transmission.plot()

generates a plot of the filter transmission curve if the filter_parameters argument was provided.

gas_parameters:

A dictionary containing parameters of each molecule relevant to the simulation of the Raman lines.

lambda_pol:

A dictionary that contains the resulting wavelength (in nm) after polarized scattering. It has keys same as the elements of the all_molecules variable.

lambda_depol_line:

A dictionary that contains a numpy array of the resulting wavelength after depolarized scattering for each gas, Raman line, and branch (Stokes, AntiStokes, Unshifted). It has keys in the form of:

- N2_O (N2, AntiStokes branch),
- N2_Q (N2, Stokes branch)
- N2_O (N2, Unshifted branch)

for each element of the linear_molecules variable.

xsection_pol:

A dictionary that contains the scattering/backscattering cross-section from polarized scattering. It has the same keys as the lambda_pol variable.

xsection_depol_line:

A dictionary that contains a numpy array with the scattering/backscattering cross-section from depolarized scattering for each gas, Raman line, and branch (Stokes, AntiStokes, Unshifted). It has the same keys as the lambda_depol_line variable.

Valid cross_section_type values

- **main_line**: Cross section including only the polarized part and the Q branch (e.g. Cabannes line, pure vibrational line) depending on the selected **mode**

- **full**: Cross section including all pure rotational/ro-vibrational lines (e.g. full Rayleigh spectrum, full vibrational spectrum) depending on the selected **mode**
- **polarized**: Cross section including only the polarized part
- **depolarized**: Cross section including only the depolarized (Raman) part
- **O**: Cross section including only the anti-Stokes branch
- **Q**: Cross section including only the unshifted Q branch
- **S**: Cross section including only the Stokes branch
- **wings**: Cross section including only the Stokes and anti-Stokes branches

Valid mldr_type values

- **main_line**: Cross section including only the polarized part and the Q branch (e.g. Cabannes line, pure vibrational line) depending on the selected **mode**
- **full**: Cross section including all pure rotational/ro-vibrational lines (e.g. full Rayleigh spectrum, full vibrational spectrum) depending on the selected **mode**
- **polarized**: Cross section including only the polarized part
- **depolarized**: Cross section including only the depolarized (Raman) part