atmosphere

Article

Py4CAtS—PYthon for Computational ATmospheric Spectroscopy

Franz Schreier *, **Sebastián Gimeno García** †, **Philipp Hochstaffl** and **Steffen Städt**

Deutsches Zentrum für Luft- und Raumfahrt (DLR), Institut für Methodik der Fernerkundung (IMF), 82234 Oberpfaffenhofen, Germany; Sebastian.GimenoGarcia@eumetsat.int (S.G.G.); Philipp.Hochstaffl@dlr.de (P.H.); Steffen.Staedt@dlr.de (S.S.)

* Correspondence: franz.schreier@dlr.de

† Current address: EUMETSAT, 64283 Darmstadt, Germany.

Received: 5 April 2019; Accepted: 6 May 2019; Published: 10 May 2019

Abstract: Radiation is a key process in the atmosphere. Numerous radiative transfer codes have been developed spanning a large range of wavelengths, complexities, speeds, and accuracies. In the infrared and microwave, line-by-line codes are crucial esp. for modeling and analyzing high-resolution spectroscopic observations. Here we present Py4CAtS—PYthon scripts for Computational ATmospheric Spectroscopy, a Python re-implemen-tation of the Fortran Generic Atmospheric Radiation Line-by-line Code GARLIC, where computationally-intensive code sections use the Numeric/Scientific Python modules for highly optimized array processing. The individual steps of an infrared or microwave radiative transfer computation are implemented in separate scripts (and corresponding functions) to extract lines of relevant molecules in the spectral range of interest, to compute line-by-line cross sections for given pressure(s) and temperature(s), to combine cross sections to absorption coefficients and optical depths, and to integrate along the line-of-sight to transmission and radiance/intensity. Py4CAtS can be used in three ways: in the (Unix/Windows/Mac) console/terminal, inside the (I)Python interpreter, or Jupyter notebook. The basic design of the package, numerical and computational aspects relevant for optimization, and a sketch of the typical workflow are presented. In conclusion, Py4CAtS provides a versatile environment for "interactive" (and batch) line-by-line radiative transfer modeling.

Keywords: infrared; radiative transfer; molecular absorption; line-by-line

1. Introduction

Radiative transfer is an important aspect in various branches of physics, esp. in the atmospheric sciences, both for Earth and (exo-)planets [1–3]. Many radiative transfer models have been developed in recent decades (see [4], for an early review and https://en.wikipedia.org/wiki/Atmospheric_radiative_transfer_codes for an up-to-date listing (All links in this paper were checked and active 4 April 2019.)), ranging from simple approximations to complex, often highly optimized codes and usually tailored to a specific spectral domain.

An essential prerequisite for the analysis of data recorded by atmospheric remote sensing instruments as well as for theoretical investigations such as retrieval assessments is a flexible, yet efficient and reliable radiative transfer model (RTM). Furthermore, as the retrieval of atmospheric parameters is in general a nonlinear optimization problem (inverse problem), the retrieval code must be closely connected to the radiative transfer code (forward model).

Radiative transfer depends on the state of the atmosphere (pressure, temperature, composition), the optical properties of the atmospheric constituents (molecules and particles), geometry, and spectral range. Because of the numerous parameters the setup of a radiative transfer calculation can be cumbersome and error prone, and several interfaces have been developed to ease the application. MODO ([5], see also http://www.rese.ch/products/modo/) is a "graphical front-end" to the MODTRAN [6] radiative transfer code, and a Python interface to MODTRAN's predecessor, the low-resolution band model LOWTRAN, is available at https://pypi.org/project/lowtran/. Wilson [7], see also http://www.py6s.rtwilson.com/, has developed a Python interface to the widely used 6S (Second Simulation of the Satellite Signal in the Solar Spectrum, [8]) model.

In the infrared and microwave region, line-by-line (lbl) modeling of molecular absorption is mandatory esp. for high spectral resolution. Furthermore, lbl models are required for the setup and verification of fast parameterized models, e.g., band, k-distribution, or exponential sum fitting models [6,9,10]. Moreover, lbl models are ideal for radiative transfer in planetary atmospheres, where fast models often parameterized for Earth-like conditions are not suitable [11,12].

An essential input for lbl radiative transfer codes are the molecules' optical properties compiled in databases such as HITRAN [13], HITEMP [14], GEISA [15], the JPL and CDMS catalogs [16–18], or ExoMol [19] that list line parameters (center frequency, line strength, broadening parameters etc.) for millions to billions of lines of some dozen molecules. Lbl models are computationally challenging because of the large number of spectral lines to be considered and the large number of wavenumber grid points required for adequate sampling. Nevertheless, lbl models are available for almost half a century, e.g., FASCODE or 4A (Fast Atmospheric Signature Code, Automatized Atmospheric Absorption Atlas, [20,21]). Thanks to the increased computational power (and because of the increased number of high-resolution Earth and planetary spectra) a variety of codes has been developed since these early efforts, e.g., ARTS (Atmospheric Radiative Transfer Simulator, [22–24]), GARLIC (Generic Atmospheric Radiation Line-by-line Code, [25]), GenLN2 [26], KOPRA (Karlsruhe Optimized & Precise Radiative transfer Algorithm, [27]), LblRTM (Line-by-line Radiative Transfer Model, [28]), LinePak [29], RFM (Reference Forward Model, [30]), σ-IASI [31], and VSTAR (Versatile Software for Transfer of Atmospheric Radiation, [32]).

Several web sites offer an Internet access to lbl databases and/or lbl models: "HITRANonline" [33] at http://hitran.org/ allows the downloading of line and other data from the most recent HITRAN version. With the "Information System HITRAN on the Web (HotW)" at http://hitran.tsu.ru (see also http://spectra.iao.ru/) several types of spectra can be simulated using HITRAN data. The HITRAN Application Programming Interface (HAPI) [34] is "a set of routines in Python which aims to provide remote access to functionality and data given by the HITRANonline". A similar service (visualization etc.) for the GEISA database is offered at https://cds-espri.ipsl.upmc.fr/geisa/. The "wavelength search engine" SpectraPlot ([35], see also http://www.spectraplot.com/) is a "web-based application for simulating spectra of atomic and molecular gases" employing, among others, HITRAN and HITEMP. http://www.spectralcalc.com/ provides a web interface to the widely used LinePak [29] model (with extended functionality for subscribed users). Goddard's Planetary Spectrum Generator (PSG, https://psg.gsfc.nasa.gov/, Villanueva et al. [36]) can be used to generate high-resolution spectra of planetary bodies (e.g., planets, moons, comets, exoplanets). Knowledge of the atmospheric transmission is also important for ground-based astronomy observations, and some tools for the correction of telluric absorption are available, e.g., MolecFit ([37], see also https://www.uibk.ac.at/eso/software/molecfit.html.en) and TAPAS (Transmissions Atmosphériques Personnalisées Pour l'AStronomie, [38], online at http://cds-espri.ipsl.fr/tapas/) (both codes use LblRTM). An early attempt to a "virtual lab" providing a web interface to MIRART (the Fortran 77 predecessor of GARLIC, [39]) and FASCODE is described in Ernst et al. [40].

In general, RTMs incl. lbl models work as a kind of "black-box". Given a set of specifications on spectral range, atmospheric conditions, geometry etc. compiled in an input file (or configuration file etc., e.g., the `TAPE5` used by LOWTRAN, MODTRAN, and FASCODE), the program is executed reading this input file (and probably some further auxiliary files, e.g., the lbl database) and one or several spectra (transmission, radiance, ...) are returned (e.g., saved in file(s)). This approach is clearly advantageous when numerous transmission and/or radiance spectra must be generated, e.g., for the operational analysis of thousands or millions of observed spectra. However, to better understand the physics of radiative transfer, inspection of intermediate variables can provide useful insight. For example, the selection of an appropriate spectral range or measurement geometry is crucial for the design and setup of a new measurement system, and analysis and visualization of line parameters, absorption cross sections and coefficients, layer optical depths, etc. can be useful. Methods and tools have been developed for the "automated" definition of "microwindows", i.e., spectral intervals optimized for retrieval of temperature or molecular concentrations from thermal emission infrared observations (e.g., [41–46]). Despite these and similar efforts interactive and flexible RTMs with an easy access to all kind of intermediate variables ("step-by-step", similar to a debugger) appear to be useful, also for pedagogical purposes.

Py4CAtS—Python for Computational Atmospheric Spectroscopy—has been developed with this intention. Since its first release in the mid-1990s [47,48] Python has become increasingly popular for scientific computing [49,50]. Thanks to its numerous extension packages, in particular NumPy/SciPy [51], the "Python-based ecosystem" ([52], https://scipy.org/) is widely considered to be the language of choice in many areas, e.g., atmospheric science and astrophysics [53–55].

Originally, the development of Py4CAtS was motivated by the idea of computational steering [56], i.e., numerically time-consuming tasks such as the lbl evaluation of molecular absorption cross sections are performed by a code (e.g., GARLIC subroutines) written in compiled languages such as Fortran (or C, C++, . . .) that is made accessible from Python (or another scripting language) using a wrapper such as PyFort [57] or F2Py [58]. More precisely, the intention was to call subroutines of GARLIC from Python. Unfortunately porting the wrapper code from one machine to another turned out to be difficult and time-consuming with early versions of PyFort. However, thanks to the increased performance of NumPy a Python implementation of lbl cross sections became feasible, i.e., the interface to GARLIC's subroutines became less important. Accordingly, the further development of GARLIC and Py4CAtS became largely independent, and Py4CAtS is now a full lbl radiative transfer tool kit delivering absorption cross sections and coefficients, optical depths, transmissions, weighting functions, and radiances.

The paper is organized as follows: After a brief review of the basic facts of lbl infrared and microwave (for brevity simply IR in the following) radiative transfer in the following section some numerical and implementation aspects are discussed in Section 2.2. Usage of the code as well as some implementation details are presented in Section 3. The paper continues with a discussion of several aspects in Section 4 before the final conclusion in Section 5. Implementation aspects are described in the Appendix A.

2. Theory and Methods

2.1. Molecular Absorption and Radiative Transfer

Radiative transfer in the IR is essentially described by the Schwarzschild equation and Beer-Lambert law (assuming negligible scattering and local thermal equilibrium in an inhomogeneous atmosphere [1–3]). Given pressure p and temperature T and line parameters from HITRAN, GEISA, etc. the first step of a simulation is to compute the absorption cross section (*xs*) for a particular molecule m

$$k_m(\nu, p, T) \;=\; \sum_l S_{ml}(T)\, g(\nu;\, \hat{\nu}_{ml}, \gamma_{ml}(p, T))\;. \tag{1}$$

where ν is the wavenumber and the sum comprises all relevant lines characterized by position $\hat{\nu}_{ml}$, strength S_{ml}, and broadening parameter γ_{ml}. The line strength $S(T)$ at temperature T is obtained as (note that indices for lines and molecules have been omitted)

$$S(T) \;=\; S(T_0)\, \frac{Q(T_0)}{Q(T)}\, \frac{\exp\left(-E_{\rm i}/k_{\rm B}T\right)}{\exp\left(-E_{\rm i}/k_{\rm B}T_0\right)}\, \frac{1 \,-\, \exp\left(-hc\hat{\nu}/k_{\rm B}T\right)}{1 \,-\, \exp\left(-hc\hat{\nu}/k_{\rm B}T_0\right)}. \tag{2}$$

with reference strength $S(T_0)$ and lower state energy E_i from the database; $Q(T)$ is the product of rotational and vibrational partition functions (for details see [25], subsection 3.4 or [59]) and c, h, and k_B are speed of light, the Planck constant, and the Boltzmann constant, respectively. The combined effect of pressure broadening (corresponding to a Lorentzian profile g_L) and Doppler broadening (corresponding to a Gaussian profile g_G) is described by a convolution resulting in the Voigt line profile [60]

$$g_\mathrm{V}(\nu - \hat{\nu}, \gamma_\mathrm{L}, \gamma_\mathrm{G}) \;=\; g_\mathrm{L} \otimes g_\mathrm{G} \;=\; \int_{-\infty}^{\infty} \mathrm{d}\nu' \; g_\mathrm{L}(\nu - \nu', \gamma_\mathrm{L}) \times g_\mathrm{G}(\nu' - \hat{\nu}, \gamma_\mathrm{G}) \,. \tag{3}$$

The Lorentz and Gauss half widths γ (at half maximum, HWHM) are pressure- and temperature-dependent,

$$\gamma_\mathrm{L} \;=\; \gamma_\mathrm{L}^0 \, \frac{p}{p_0} \left(\frac{T_0}{T}\right)^n \tag{4}$$

$$\gamma_\mathrm{G} \;=\; \hat{\nu} \sqrt{\frac{2 \ln 2 \, k_\mathrm{B} T}{\mu c^2}} \tag{5}$$

where the air-broadening coefficient γ_L^0 and exponent n are given in the database for reference pressure p_0 and temperature T_0, and μ is the molecular mass. Please note that contributions to the Lorentzian width (4) due to self-broadening have been ignored: Except for water at Earth's bottom of atmosphere (BoA), molecular mixing ratios are usually very small, and the self-broadening contribution would be small to negligible. Moreover, the self-broadening coefficient is often close to the foreign broadening coefficient (or even undefined, at least for older versions of HITRAN and GEISA). Clearly, for planetary atmospheres (e.g., Mars or Venus) self-broadening can be an important issue.

The increasing quality of spectroscopic observations has indicated deficiencies of the Voigt profile, and several more advanced profiles have been suggested [61,62]. In addition to the Lorentz, Gauss, and Voigt profile, implementations of the Rautian and speed-dependent Voigt profile [63] are available.

The absorption coefficient (*ac*) is defined as the sum of all molecular cross sections scaled by the molecular number densities n_m (in general depending on altitude z and the corresponding pressure)

$$\alpha(\nu, z, p, T) \;=\; \sum_m n_m(z) \, k_m(\nu, p(z), T(z)) \,. \tag{6}$$

Integration of the absorption coefficient along the line-of-sight gives the optical depth (*od*), i.e., for a plane-parallel atmosphere and a vertical path from altitude z_lo to z_hi

$$\tau(\nu) \;=\; \int_{z_\mathrm{lo}}^{z_\mathrm{hi}} \alpha(\nu, z, p, T) \, \mathrm{d}z \,. \tag{7}$$

This is closely related to the monochromatic transmission describing the ratio of incoming and outgoing radiation

$$\mathcal{T}(\nu) \;=\; \exp(-\tau(\nu)) \,. \tag{8}$$

Finally, the integral form of Schwarzschild's equation gives the (spectral) radiance or intensity (*ri*)

$$I(\nu) \;=\; I_\mathrm{b}(\nu) \, \mathrm{e}^{-\tau_\mathrm{b}(\nu)} \;+\; \int_0^{\tau_\mathrm{b}} B(\nu, T(\tau')) \, \mathrm{e}^{-\tau'} \, \mathrm{d}\tau' \tag{9}$$

$$\;=\; I_\mathrm{b}(\nu) \, \mathrm{e}^{-\tau_\mathrm{b}(\nu)} \;-\; \int_0^{s_\mathrm{b}} B(\nu, T(s')) \, \frac{\partial \mathcal{T}}{\partial s'} \, \mathrm{d}s' \tag{10}$$

where the first term describes an attenuated background contribution I_b at a distance s_b from the observer (corresponding to the optical depth τ_b), e.g., from Earth's (or a planet's) surface in case of nadir-viewing. The source function in the second term is Planck's function depending on temperature,

$$B(\nu, T) = \frac{2hc^2\nu^3}{\exp(hc\nu/k_\mathrm{B}T) - 1} . \tag{11}$$

The finite spectral resolution of measured spectra can be modeled by convolution of the monochromatic transmission (8) and/or radiance (9) with an appropriate spectral response function (SRF, a.k.a. instrument line shape, ILS), e.g.,

$$\tilde{I} = I \otimes \mathrm{SRF} . \tag{12}$$

Weighting functions (*wf*) are an important concept for atmospheric temperature sounding and are a measure of the contribution of a particular atmospheric layer to the radiation seen by an observer, see Equation (10). They are defined by $\frac{\partial \mathcal{T}}{\partial z}$ for a vertical path, or more generally

$$\frac{\partial \mathcal{T}(\nu, s)}{\partial s} = -\mathcal{T}(\nu, s)\, \alpha(\nu, s) \tag{13}$$

for a slant path with $s = z/\cos\theta$ and zenith angle θ (with $\theta = 0$ and $\theta = 180°$ for vertical uplooking and downlooking, respectively).

2.2. *Numerics*

2.2.1. Lbl Molecular Absorption Cross Sections: Multigrid Voigt Function

For the computation of the Voigt profile g_V (3) depending on three variables it is convenient to introduce the Voigt function $K(x, y)$ depending on two variables, essentially the distance x to the center peak and the ratio y of the line widths

$$g_\mathrm{V}(\nu - \hat{\nu}, \gamma_\mathrm{L}, \gamma_\mathrm{G}) = \frac{\sqrt{\ln 2/\pi}}{\gamma_\mathrm{G}} K(x, y) \tag{14}$$

$$K(x, y) = \frac{y}{\pi} \int_{-\infty}^{\infty} \frac{e^{-t^2}}{(x-t)^2 + y^2} \mathrm{d}t \tag{15}$$

with $x = \sqrt{\ln 2}\, (\nu - \hat{\nu})/\gamma_\mathrm{G}$ and $y = \sqrt{\ln 2}\, \gamma_\mathrm{L}/\gamma_\mathrm{G}$. As there is no closed-form solution for this convolution integral, numerical approximations are mandatory. Most modern algorithms consider the complex error function [64,65] whose real part is identical to the Voigt function

$$w(z) \equiv K(x, y) + \mathrm{i}L(x, y) = \frac{\mathrm{i}}{\pi} \int_{-\infty}^{\infty} \frac{e^{-t^2}}{z - t} \mathrm{d}t \qquad \text{with} \quad z = x + \mathrm{i}y . \tag{16}$$

Rational approximations, closely related to continued fractions and defined as the quotient of two polynomials [66,67], have been used successfully for a large variety of functions including the complex error function (e.g., [68–70]). Py4CAtS (like GARLIC) uses an optimized combination [71] of the Humlíček [69] and Weideman [70] algorithms

$$w(z) = \begin{cases} \frac{\mathrm{i}z/\sqrt{\pi}}{z^2 - \frac{1}{2}} & |x| + y > 15 \\ \frac{\pi^{-1/2}}{L - \mathrm{i}z} + \frac{2}{(L - \mathrm{i}z)^2} \sum\limits_{n=0}^{N-1} a_{n+1} Z^n & \text{otherwise} \end{cases} \tag{17}$$

where $Z = \frac{L+\mathrm{i}z}{L-\mathrm{i}z}$ and $L = 2^{-1/4}N^{1/2}$. For $N = 24$ this provides an accuracy better than 10^{-4} everywhere except for very small y and intermediate x that are hardly relevant in practice; for $N = 32$ the relative error $|\Delta K|/K$ is less than 8×10^{-5} for all x, y of interest.

Because of the huge number of Voigt function evaluations even highly optimized complex error function algorithms are insufficient and further optimizations are mandatory. Py4CAtS evaluates the monochromatic cross sections on a uniform (equidistant) wavenumber grid with the spacing estimated as a fraction of the "typical" line widths, i.e., $\delta\nu = \langle\gamma\rangle/n$ with the mean half width $\langle\gamma\rangle$ and a default sampling rate $n = 4$. In case of the Voigt line profile (3), its width is estimated using an approximation due to Whiting [72] (accurate to about one percent)

$$\gamma_{\mathrm{V}} \approx \frac{1}{2}\left(\gamma_{\mathrm{L}} + \sqrt{\gamma_{\mathrm{L}}^2 + 4\gamma_{\mathrm{G}}^2}\right). \tag{18}$$

Please note that $\delta\nu$ and hence the number of grid points and cross section values are dependent on pressure, temperature, and molecular mass (via Equations (4), (5), and (18)), and change with atmospheric level and molecule. These wavenumber steps are clearly appropriate in the line center region; however, in the line wings the profiles are relatively smooth (roughly $g(\nu - \hat{\nu}) \approx 1/(\nu - \hat{\nu})^2$ for the Lorentz or Voigt profile) and a coarser grid would be sufficient.

Py4CAtS uses a two- or three-grid approach [73]. In the two-grid scheme, the Voigt line profile is calculated on the fine grid only in the line center region, and on a coarse grid (typically with spacing $\Delta\nu = 8\delta\nu$) in the entire spectral region. The line profiles are accumulated independently to the fine and coarse grid cross sections, i.e., the interpolation of the coarse grid cross section is done only once after all lines have been summed up. This allows a speed-up roughly in the order of the ratio of fine and coarse grid points. A significant further speed-up can be achieved with three grids of fine, medium, and coarse resolution and exploitation of the asymptotic properties of the profile function.

2.2.2. Integration of the Beer and Schwarzschild Equations

From the very beginning of the code development (MIRART, GARLIC, and Py4CAtS) numerical schemes have been used whenever possible. Hence, in contrast to most lbl radiative transfer codes dividing the atmosphere into a series of layers with constant pressure, temperature, and concentrations (Curtis-Godson approximation), Py4CAtS and GARLIC are level-based and exploit numerical quadrature for Equations (7) and (9). In particular, the optical depth is approximated by the sum of absorption coefficients according to the trapezoid rule (e.g., [66]). For the Schwarzschild Equation (9) both codes use optical depth τ as integration variable and assume that the Planck function varies linearly or exponentially with optical depth within a layer,

$$B(\tau) = b_0 B(\tau_l) + b_1 B(\tau_{l+1}) \qquad \text{with } b_0 = \frac{\tau_{l+1} - \tau}{\tau_{l+1} - \tau_l} \text{ and } b_1 = \frac{\tau - \tau_l}{\tau_{l+1} - \tau_l} \tag{19}$$

$$B(\tau) = B(\tau_l)\,\mathrm{e}^{\beta(\tau - \tau_l)} \qquad \text{with } \beta = \log\big(B(\tau_l)/B(\tau_{l+1})\big)/(\tau_{l+1} - \tau_l) \tag{20}$$

for $\tau_l \le \tau \le \tau_{l+1}$ (corresponding to the altitude interval $[z_l, z_{l+1}]$). Note the slightly different notation for the Planck function's argument here compared to Equation (11) (the optical depth depends on wavenumber and temperature). With both approximations the integral (9) can be computed analytically within a layer.

Because the overall runtime is essentially determined by the evaluation of the cross sections, the speed of level-based and layer-based radiative transfer is equivalent (assuming that the number of levels and layers is comparable). With respect to accuracy, none of the two approaches is superior or inferior; in fact, neither of the intercomparison studies [74–76] identified path integration as an issue. For an in-depth discussion of the "linear-in-τ" vs. "exponential-in-τ" approximations as well as some further quadrature schemes see subsection 4.1 of the GARLIC paper [25].

3. The Package: Usage and Implementation

In Py4CAtS the individual steps of an IR radiative transfer computation are implemented in a series of modules and functions, see Figure 1:

- `higstract`: extract (select) lines of relevant molecules in the spectral range of interest;
- `lbl2xs`: compute lbl cross sections for given pressure(s) and temperature(s): Equation (1);
- `xs2ac`: multiply cross sections with number densities and sum over all molecules: Equation (6);
- `ac2od`: integrate absorption coefficients along the line-of-sight through the atmosphere to the vertical optical depth (7);
- `od2ri`: solve Schwarzschild equation (9), i.e., integrate the Planck function (11) vs. optical depth along the line-of-sight through atmosphere (assuming a plane-parallel, non-scattering atmosphere in local thermal equilibrium).

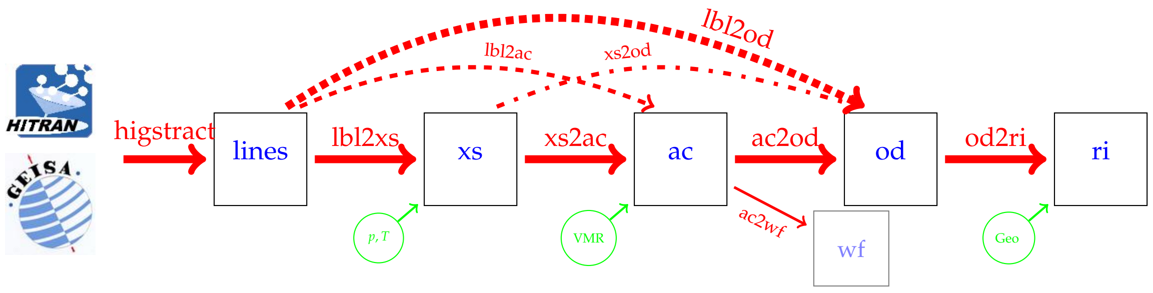

Figure 1. From HITRAN/GEISA via cross sections (xs) and absorption coefficients (ac) to optical depths (od) and radiation intensity (ri). Please note that cross sections are pressure and temperature (pT) dependent, absorption coefficients also depend on composition (VMR), and optical depth and radiation intensity depends on path geometry (geo).

All Python source files along with supplementary files are delivered as a single tarball. Unpacking this file will create four directories with the source files in `src`, data in `data`, documentation in `doc`, and executable scripts in `bin`. Actually, the files in the `bin` directory are just links to the scripts in the `src` directory (e.g., `bin/lbl2xs` is a link to `src/lbl2xs.py`), and in the following subsection it is assumed that the `bin` directory is included in the shell's search path for executables (`PATH` environment variable for Unix-like systems). When Py4CAtS is used inside the IPython interpreter (see Section 3.2), all functions etc. should have been imported using a statement such as `%run /home/schreier/py4cats/src/py4cats.py`. Importing Py4CAtS can be easily done automatically by an appropriate modification of the (I)Python configuration file, e.g., by adding this file to the list of files to run at IPython startup. (In IPython 6 this list is defined by `c.InteractiveShellApp.exec_files` in `.ipython/profile_default/iypthon_config.py`.)

3.1. Running Py4CAtS in the Classical Console

A typical session in a Unix/Linux environment (console/terminal) is shown in Figure 2. After creating a new directory, the first step is to extract the relevant lines from the HITRAN or GEISA data base using `higstract`. (Data are available at hitran.org and http://cds-espri.ipsl.upmc.fr/. Py4CAtS can read the original data file as is, i.e., the 160-byte fixed-width format used since HITRAN 2004 or the 100-byte format of the older HITRAN versions. Regarding GEISA, the format of a single record (corresponding to a "physical line") has changed slightly from version to version, and Py4CAtS parses the file name to select the appropriate reader.) Please note that for this command at least one option is mandatory, i.e., the spectral range (here 50 to 60 cm^{-1}) or a molecule must be specified (for convenience, `higstract -m main ...` can be used to extract line data for the first seven molecules (H_2O, CO_2, ..., O_2) of HITRAN or GEISA.) `higstract` saves the data in several tabular ASCII files, one file per molecule. By default, each file has five columns for line position $\hat{\nu}$, strengths S, energy E, air-broadening parameter γ_L^0, and the exponent n and is named as `H2O.vSEan` etc., where the extension indicates the columns (further parameters, e.g., for self-broadening, can be requested with the format option `-f`).

```
mkdir example;  cd example
higstract -x 50,60 /data/hitran/2000/lines
lbl2xs H2O.vSEan CO2.vSEan O3.vSEan   # one cross section per molecule at HITRAN p, T
lbl2xs -o xSec --pT /data/atmos/mls.xy --cpT 1,2  H2O.vSEan CO2.vSEan O3.vSEan
xs2od -o mls.od  /data/atmos/mls.xy  H2O.xSec CO2.xSec O3.xSec
od2ri -o mls_uplook.ri mls.od             # default zenith angle 0dg, downwelling radiation
od2ri -o mls_nadir.ri -T 294 -z 180 mls.od # observer @ ToA, upwelling rad. incl. surface

# bypassing the cross section and absorption coefficient file I/O
lbl2od -o mls.od  /data/atmos/mls.xy  H2O.vSEan CO2.vSEan O3.vSEan
```

Figure 2. Typical workflow for lbl modeling with Py4CAtS in a Unix-like shell: Hitran/Geisa $\rightarrow$ line parameter extracts $\rightarrow$ cross sections $\rightarrow$ absorption coefficients $\rightarrow$ optical depth $\rightarrow$ radiance / intensity. Output is not shown. All functions support the `-h` option to ask for help, in particular a list of all available options/flags. For some notes on option parsing see Appendix A.7.

The next step is to compute absorption cross sections according to Equation (1). Without any option, cross sections are computed for the reference pressure and temperature used in the line parameter database, i.e., $p_0 = 1013.25\,\text{mb}$ and $T_0 = 296\,\text{K}$ for HITRAN and GEISA. Alternatively, pressure(s) and temperature(s) can be specified with the `-p` and `-T` option. For atmospheric applications it is more convenient to read the data from a file (e.g., "midlatitude summer" (mls) data assuming the first three column list altitudes, pressure, and temperature). Again, the cross sections will be saved in files, one per molecule, using NumPy's pickle format or alternatively ASCII tabular or HITRAN formatted files.

The computation continues to absorption coefficients by summing all cross sections (level-by-level) scaled with the molecular concentrations, cf. Equation (6), and integrating these to the layer (or delta) optical depth, Equation (7). Finally, the radiance seen by an observer at ground and by a spaceborne observer (actually an observer at top of atmosphere, ToA) is computed.

In each step of the calculation the results are written to file(s), and read in for the next step. Obviously, this can be quite time-consuming esp. for ASCII files (reading NumPy pickled files is surprisingly fast). In particular, the cross section files can be very large (depending on the spectral region, the size of the spectral interval, the number of molecules, and atmospheric levels, etc.). For many applications some of the intermediate quantities are not of interest, and the file transfer operations can be omitted, e.g., (see the dashed arrows in Figure 2)

- `lbl2ac` compute lbl cross sections and combine to absorption coefficients;
- `lbl2od` compute lbl cross sections and absorption coefficients, then integrate to optical depth.
- `xs2od` multiply cross sections with densities, sum over molecules, and integrate to optical depth.

In addition to these scripts there are several further Python files that contain functions used by these "main" scripts (e.g., to read or write the data), but can also be executed stand-alone. (Technically, the very first line of all executable scripts is something like `#!/usr/bin/env python` and the last segment of the module starts with `if __name__=='__main__':`.) The scripts `lines.py`, `xSection.py`, `absCo.py`, `oDepth.py` can be used to read, extract subsets in the spectral or altitude regime, summarize, plot, or reformat the respective data. `molecules.py` collects properties (e.g., mass) of the IR relevant molecules and the ID numbers used in the HITRAN and GEISA database. `atmos1D.py` is useful to handle the atmospheric data (p, T, VMRs, ...). The functions in `hitran.py` and `geisa.py` are used by `higstract.py`, but can be used as a "low-level" interface to the respective dataset. Unit conversions are implemented in the `cgsUnit.py` module (note that internally Py4CAtS uses cgs units consistently, see Appendix A.6). Furthermore, `radiance2radiance.py` and `radiance2Kelvin.py` allow the conversion of radiance spectra, e.g., wavenumber $\longleftrightarrow$ wavelength or to equivalent brightness temperature using the inverse of Planck's function (11).

The `lbl2ac`, `lbl2od`, or `xs2od` scripts clearly help to avoid the time-consuming reading and writing of some of the intermediate quantities, however, these scripts turn Py4CAtS back into the "black-box" radiative transfer discussed above.

When executing the series of individual steps, the subsequent steps must re-read the data saved in the previous step(s) from file(s) and check the consistency of the data. For example, the `xs2ac.py` script will test if all cross sections cover the same spectral interval, and if the pressures and temperatures match those in the atmospheric data file. Accordingly, a substantial part of the code is devoted to implement appropriate tests. A prerequisite for these tests is that all relevant attributes are saved in the file along with the spectra; for example, a single cross section is saved along with its pressure, temperature, and wavenumber grid specification.

3.2. *Py4CAtS Used within the (I)Python and Jupyter Shell*

Access to intermediate quantities is especially useful for visualization. In the (I)Python interpreter, graphics packages such as Matplotlib [77] allow the plotting of any kind of array. For plotting as well as for the subsequent processing steps it is important to have all attributes of a spectrum available. However, a function returning an array for the spectrum together with its attributes (e.g., with a call such as `lines_h2o, pRef, tRef = higstract('/data/hitran/86/lines', 'H2O')`) would be cumbersome, and collecting the return values in a dictionary would not significantly improve this. In Py4CAtS this problem is solved by means of subclassed and/or structured NumPy arrays that will be briefly described (in Appendices A.4 and A.5) after a presentation of the typical workflow illustrated in Figure 3.

3.2.1. Atmospheric Data

Knowledge of the atmospheric state, i.e., pressure, temperature, and molecular composition as a function of altitude, is indispensable for lbl modeling and radiative transfer. The Py4CAtS scripts called from the console automatically read these data from a file (along with the line data, cross section data, etc.) This is essentially accomplished by the function `atmRead` of the `atmos1D.py` module, e.g., the command `mls = atmRead('/data/atmos/20/mls.xy')` returns the data of the "midlatitude summer" atmosphere of the AFGL dataset [78] regridded to 20 levels (see the very first input labelled `In[1]:` in Figure 3). `atmRead` essentially expects a tabular ascii file with rows corresponding to the atmospheric levels and columns for altitudes, pressure (and/or air density), temperature, and molecular concentrations. Two comment lines are mandatory: `#what:` followed by the column identifiers and `#units:` followed by the physical units (e.g., "km").

`mls` is an example of a structured array (see Appendix A.5), where the individual profiles can be accessed by their name, e.g., `mls['T']` or `mls['H2O']`, and the data for a specific level l are given by the row index, e.g., `mls[0]` or `mls[-1]` for the bottom and top level, respectively. Please note that "rows" and "columns" can be specified in two ways; for example, the BoA temperature is `mls['T'][0] == mls[0]['T']`.

```
Jupyter QtConsole 4.3.1
Python 3.6.5 (default, Mar 31 2018, 19:45:04) [GCC]

In[1]: # get two mid latitude atmospheres
  ...: mls = atmRead('/data/atmos/20/mls.xy')
  ...: mlw = atmRead('/data/atmos/25/mlw.xy', zToA=50)

Atmos1d: got p, T, air and 7 gases at 20 levels
Atmos1d: got p, T, air and 7 gases at 18 levels

In [2]: # HItran-GeiSa-exTRACT ---> lineListDictionary
   ...: llDict = higstract('/data/geisa/87/lines', (2100,2150), molecule='main')
9771  lines of  5  molecule(s), returning a dictionary

In [3]: # CO cross section at database pressure and temperature
   ...: xs  = lbl2xs(llDict['CO'])
   ...: # cross sections for other pressures or temperatures
   ...: xs_10mb = lbl2xs(llDict['CO'], 1e4)  # p=10mb
   ...: xss = lbl2xs(llDict['CO'], temperature=[200,250,300])
   ...: # a dictionary of x-section lists (for all p, T and gases)
   ...: xssDict = lbl2xs(llDict, mls['p'], mls['T'])

In [4]: # proceed step-by-step
   ...: acList  = xs2ac(mls, xssDict) # absorption coefficients
   ...: dodList = ac2dod(acList)      # delta optical depths

In [5]: # alternatively bypass intermediate quantities, e.g.
   ...: dodList = lbl2od(mls,llDict)  # delta opt.depths

In [6]: # sum/combine optical depths and plot
   ...: odPlot([dodList[0], dodList[1]])  # the bottom layers
   ...: odPlot(dodList[0]+dodList[1])     # and their sum

In [7]: # radiation intensity seen by uplooking observer at BoA
   ...: radUp = dod2ri(dodList)
   ...: # and downlooking observer at ToA (incl. surface @ 294K)
   ...: radNadir = dod2ri(dodList, 180, mls['T'][0])
```

Figure 3. Typical workflow of an IPython session (output not shown except for first two commands).

All data are stored internally in cgs units, e.g., pressure is converted to $\mathrm{dyn/cm^2} = \mathrm{g/cm/s^2}$. Molecular concentrations are stored as number densities (i.e., if the data file contains volume mixing ratios (VMR), these are converted to densities by multiplication with the air number density $n = p/(kT)$). VMR can be obtained with the `vmr` function, e.g., `vmr(mls)`. Further convenience functions of the `atmos1D.py` module include `vcd` to integrate the (molecular and air) number densities along a vertical path through the atmosphere to the "Vertical Column Density" ($N = \int n(z)\,\mathrm{d}z$ with default limits given by top and bottom of atmosphere, z_{ToA} and z_{BoA}), `cmr` for the "Column Mixing Ratio", i.e., the ratio of the molecular VCDs and the air VCD.

The `atmRegrid (mls, zNew, ...)` function allows interpolation of the atmospheric data to a new altitude grid. If the limits of the new grid exceed the old limits, `atmRegrid` prints a warning. The actual results are depending on the chosen interpolation scheme; in case of the default linear interpolation NumPy's `interp` function simply repeats the very first or last value at BoA or ToA, respectively.

Data from different files can be combined with the `atmMerge` function, e.g.,
`combiAtm = atmMerge (mlw, traceGases)` (here the second data set "`traceGases`" is likely comprising only concentration profiles that can be read with the `vmrRead` function.) If the two data sets are given on different altitude grids, profiles from the second set are interpolated to the grid of

the first set. If a profile is defined in both data, then by default the second is ignored unless the flag `replace` is "switched on".

The data can be easily visualized using the standard Matplotlib [77] functions (e.g., `plot(mls['T'],mls['z'])`), but for convenience Py4CAtS provides a function `atmPlot` that expects a single structured array or a list thereof as first argument. (For some general remarks on visualization see Appendix A.2.) For example, `atmPlot(mls)` shows temperature vs. altitude, and `atmPlot([mls,mlw], 'O3', 'mb')` compares ozone profiles of the midlatitude summer and winter atmospheres with pressure as vertical axis. Finally, the function `atmSave` can be used to write the data to file (see also Appendix A.1).

3.2.2. Line Data

For the following presentation assume that you are interested in carbon monoxide (CO) retrievals from the thermal IR observations as done by IASI (e.g., [79,80]) or AIRS [81].
`llDict = higstract('/data/hitran/2012/lines', (2100,2150))` returns about 26 thousand lines in a dictionary of 22 line lists, more precisely structured arrays, one array for each molecule with transitions in this spectral range around the CO fundamental band around 2143 cm^{-1}. Actually, these arrays are subclassed NumPy arrays (with type `lineArray`) holding some extra information as attributes, e.g., `llDict['CO'].p` and `llDict['CO'].t` gives HITRAN's reference pressure $1013.25 \times 10^3\,\text{dyn/cm}^2$ and temperature 296.0 K, respectively. If lines of a single molecule are extracted (e.g., `higstract('/data/geisa/2003/lines', molecule='CO')`), a single `lineArray` is returned. The option `molecule='main'` returns the line parameters of the "main" molecules only, i.e., the molecules with ID numbers ≤ 7 of the HITRAN and GEISA database.

Similar to the atmospheric data, the line data can be plotted using Matplotlib's functions, or, more conveniently, with Py4CAtS' function, e.g., `atlas(llDict)`. The `atlas` function (named after Park et al. [82]) can be used with a single `lineArray` or a list or dictionary of `lineArrays` and displays line strength vs. position by default.

3.2.3. Cross Sections

In the next step the line parameters can be used to compute molecular cross sections, see the `In[3]:` block in Figure 3. `xs = lbl2xs(llDict['CO'])` returns the cross section of CO in the spectral range around 2140 cm^{-1} for the database (here HITRAN) reference pressure and temperature. A single cross section is stored in a subclassed NumPy array (i.e., `type(xs)` $\rightarrow$ `xsArray`) with "attributes" such as lower and upper wavenumber bound, pressure, temperature, and molecule stored in further items, e.g., `xs.x` or `xs.p`. As a default, the Voigt profile is considered. Please note that the cross sections are evaluated on a uniform wavenumber grid (with spacing defined by the mean line width, see Section 2.2.1), so it is sufficient (and more memory efficient) to save the very first and very last grid point only.

Cross sections for different pressures and temperatures are obtained by specifying the second or third function argument of `lbl2xs`. Please note that the data type returned by `lbl2xs` in these examples is different, i.e., the type is depending on the number of p, T pairs and the type of the line data (a single or a dictionary of `lineArray`, essentially the number of molecules). In the very first example (CO and one p, T) in Figure 3 a single subclassed NumPy array `xsArray` is returned, whereas a list of `xsArray`'s is returned for a list of p, T pairs and a single molecule. Finally, the last example gives a dictionary of lists of `xsArray`'s, each list for a single molecule and a dictionary entry for each molecule.

The `xSection.py` module has a function `xsPlot` to visualize the cross sections, e.g., `xsPlot(xss)`. This function works recursively (cf. Appendix A.3), i.e., it can be called with a single `xsArray`, a list thereof, or a dictionary of (lists of) `xsArray`'s. Figure 4 demonstrates the combined use of the `atlas` and `xsPlot` functions exploiting Matplotlib's function `twinx` to share the common wavenumber axis. The functions `xsInfo` and `xsSave` can be used with a single cross section array, a list of cross sections, or a dictionary of lists to summarize the cross sections' properties or to write the data to file(s); Reading cross section data from file(s) is possible with the `xsRead` function.

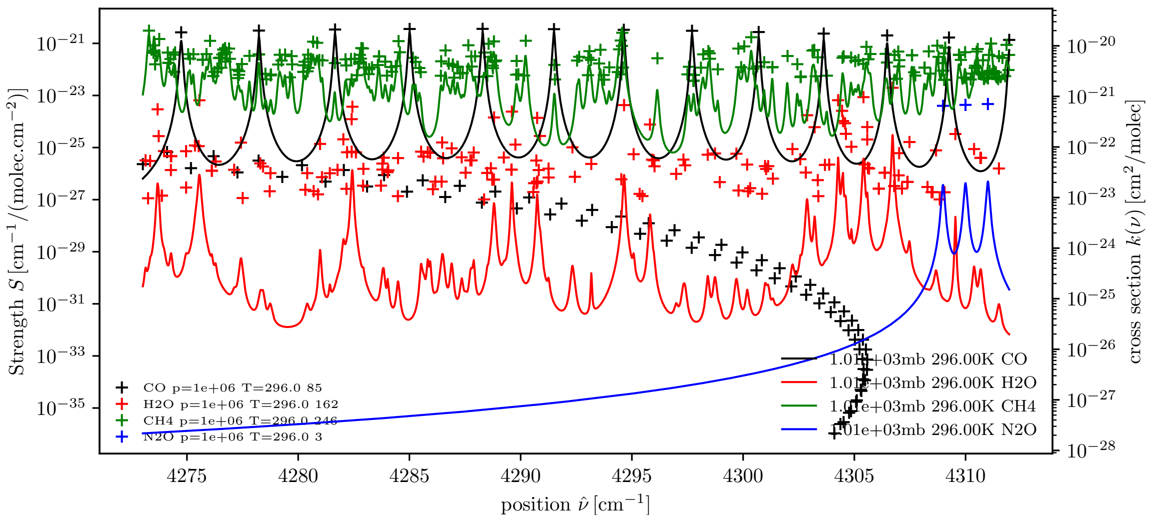

Figure 4. Combination of line atlas and xsPlot. This example has been generated with
`dll = higstract('/data/hitran/2000/lines',(4273,4312), 'main') # dict of line lists`
`xss = lbl2xs(dll) # dict of cross sections` .
`atlas(dll); twinx(); xsPlot(xss)`

3.2.4. Absorption Coefficients

Given cross sections of some molecules on a set of p, T levels along with the atmospheric data, in particular the molecular number densities, the absorption coefficient (6) for all levels are generated with `acList = xs2ac (mls, xssDict)`. The list contains a spectrum for each atmospheric p, T level, where each spectrum is stored in a subclassed NumPy array: `type(acList[0])` $\rightarrow$ `acArray` similar to the cross sections, i.e., with attributes stored as, e.g., `ac.x` and `ac.z` for the wavenumber range and altitude, respectively. Please note that the number of levels in the atmospheric data set (here `mls`) and the lengths of the cross section lists in the `xssDict` dictionary must be identical. Furthermore, all molecules with cross section data must be contained in the atmospheric data (but there can be some "unused" molecules in the atmospheric data set).

The absorption coefficients can be plotted with the standard Matplotlib functions, but Py4CAtS also has a function to make this easier: `acPlot(acList)`. The function `acInfo(acList)` prints essential information about the absorption coefficients (Actually it is a loop calling the corresponding `info` method of `acArray`, i.e., `for ac in acList: ac.info()`).

The data can be saved to file (tabular ascii) with the standard NumPy `savetxt` or Py4CAtS' `awrite` function. The `acSave` function automatically saves the absorption coefficients along with the atmospheric data, and the `acRead` function allows reading of the data (incl. the associated atmosphere) back from file, e.g., `absCo = acRead(acFile)`. Both `acSave` and `acRead` also support HITRAN formatted files or Python/NumPy's internal pickle format.

3.2.5. Optical Depths

The next step is to integrate the absorption coefficients along the (vertical) path through the atmosphere using the function `ac2dod` (see `In[4]` in Figure 3). Similar to cross sections and absorption coefficients this returns a list of `(nLevels-1)` subclassed NumPy arrays `odArray`, where each list member is essentially the delta / differential / layer optical depth spectrum along with its attributes lower and upper altitudes, pressures, and temperatures (and the wavenumber interval, too).

The optical depths instances can be combined by addition or subtraction, e.g., the delta optical depths of the first (bottom) two layers can be added by `dodList[0]+dodList[1]` (see Figure 5).

The main purpose of the `__add__` special method is to combine the optical depths from neighboring layers, but it can also be used to sum the optical depths of different molecules in one layer.

```
# delta optical depth list
dodl = lbl2od(mls, dictOfLineLists)

# the first two layers and their sum
odPlot([dodl[0], dodl[1],
        dodl[0]+dodl[1]])
# also plot total optical depth
odPlot(dod2tod(dodl))
```

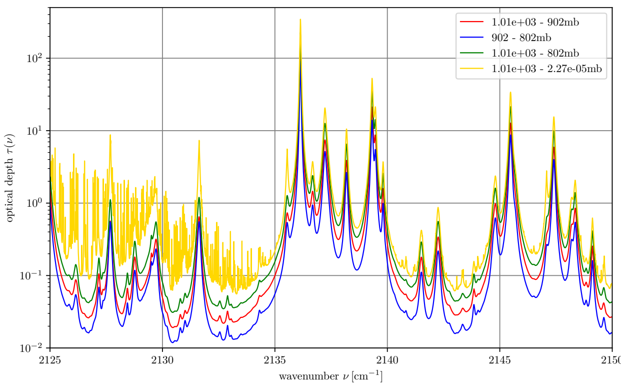

Figure 5. Computing and combining optical depths.

Please note that the `ac2dod` function (and the `xs2dod`, `lbl2od`, ... function discussed below) only return delta optical depths (dod), further functions can be used to convert to cumulative or total optical depth. Summation of all layer optical depths with the `dod2tod` function delivers the total optical depth and `dod2cod` returns the (ac)cumulated optical depth. By default, the accumulation is starting with the very first layer (usually at BoA) and the very last element of the generated list should be the total optical depth, whereas `cod = dod2cod(dodList,True)` starts accumulating with the very last layer and the very first `cod[0]` corresponds to the total optical depth.

A quick-look of optical depth(s) can be generated with the `odPlot` function, and the data (incl. the attributes) can be saved to file using `odSave` using ASCII, netcdf, or pickle format. Later, the optical depth data can be read from file into a (new) IPython session with `oDepth = odRead (odFile)`.

The `oDepthOne` function returns the distance s_1 from the (uplooking or downlooking) observer to the point, where the optical depth is one, $\tau(\nu, s_1) = 1.0$, corresponding to a transmission that has decreased to $\mathcal{T} = 1/e$. This distance should roughly correspond to the location of the weighting function maximum.

3.2.6. Weighting Functions

`wgtFct = ac2wf(acList[, angle, zObs])` computes weighting functions according to Equation (13) for an observer at altitude `zObs` looking in direction `angle` (default 180°) and returns a subclassed 2D NumPy array of type `wfArray` (with `wgtFct.shape = (len(vGrid), len(sGrid))`). By default, the observer is assumed to be at ToA or BoA for viewing angles larger or smaller than 90°. The attributes define the wavenumber interval, path distance grid `sGrid` (relative to the observer (in cm), i.e., from ToA to BoA in case of a downlooking nadir view), observer altitude, and viewing angle. Optionally `ac2wf` also allows the treatment of finite field-of-view effects with an extra argument `FoV` to set the type and width (HWHM, in degree) (e.g., `FoV='Gauss 7.5'`).

Alternatively, given the delta/layer optical depths the weighting functions can be approximated by finite differencing using the `dod2wf(dodList,zObs,angle)` function, but starting from the absorption coefficient is much more reliable. For weighting functions of a horizontal path (zenith angle $\theta = 90°$) see the first remark in Section 4.2.

The function `wfSave(wgtFct, ...)` and `wfRead(wfFile, ...)` can be used to save and read the data, and `wfPlot(wgtFct[, wavenumber, ...])` provides a simple visualization tool. If (a) specific wavenumber(s) are given, a 2D plot is presented, otherwise a contour plot is generated. An example of weighting functions for microwave temperature sounding in the region of the oxygen rotation band is given in Figure 6.

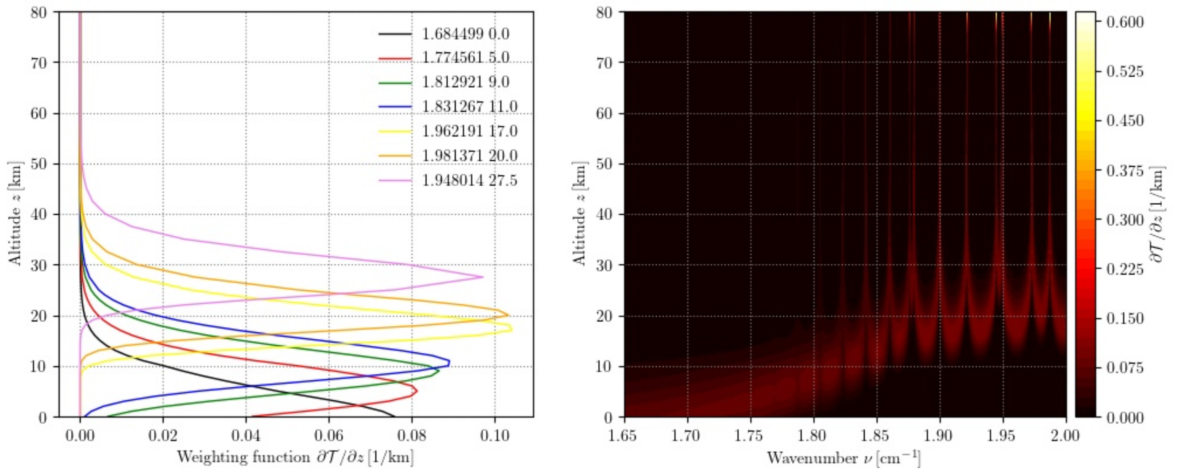

Figure 6. Combination of wfPlot. The labels in the left plot correspond to the central wavenumbers (in cm^{-1}) for each of the seven SSM/T channels shown in Fig. 7.13 of Liou [2]. The second number indicates the altitude grid point (in km) next to the maximum. This example has been generated with

```
sas = atmRead('/data/atmos/50/subarcticSummer.xy', zToA=80)
dll = higstract('/data/hitran/2000/lines',(0,10), 'main'); del dll['CO']
acList = lbl2ac(sas, dll, (1.65,2))
wgtFct = ac2wf(acList, 180)
ssmtFreqs=array([50.5, 53.2, 54.35, 54.9, 58.825, 59.4, 58.4,])*1e9/c
subplot(121); wfPlot(wgtFct, wavenumber=ssmtFreqs)
subplot(122); wfPlot(wgtFct, nLevels=50)
```

.

3.2.7. Radiance/Intensity

The `dod2ri` function evaluates the Schwarzschild integral (9) and returns the radiance or intensity, again a subclassed NumPy array `riArray` with attributes for wavenumber interval, altitude, pressure, and temperature minimum/maximum, observer zenith angle, and background temperature, see the last block in Figure 3. Without optional arguments, the radiance seen by an uplooking observer at the surface (BoA) is computed, whereas `dod2ri (dodList, 180.0, mls['T'][0])` gives the radiance for a nadir-viewing observer looking down from ToA with an angle of 180.0° (relative to the zenith angle) to Earth; the third argument specifies the surface temperature T_b (here the BoA temperature of the midlatitude summer (mls) atmosphere) that is used to evaluate a Planck background contribution in (9) with $I_b(\nu) = B(\nu, T_b)$.

Please note that `dod2ri` does not have any argument to specify the observer altitude, i.e., it computes the radiance at BoA or ToA for an angle smaller or larger than 90° (a horizontal path with angle 90° is not implemented). If you want to model the radiance, say, for an airborne observer downlooking from 10 km and have a list of layer optical depths for an atmosphere with a uniform altitude grid of 1 km (hence layer thickness 1 km), supply a list of the first ten optical depths only, i.e., `dod2ri(dodList[:10],180)`. (See also the remark on limitations in Section 4.6.)

A further Boolean optional argument can be given to switch to the "B exponential-in τ" approximation instead of the default "B linear-in τ", see Section 2.2.2.

To plot and save the radiance spectrum (along with the wavenumber grid) in a file use the `riPlot` and `riSave` functions, respectively. To convolve the radiance spectrum I with a spectral response function according to (12), a special method `convolve` has been implemented, e.g., `radBox1 = radiance.convolve()` uses the default "box" with a half width 1.0 cm^{-1}. Likewise, `radGauss2 = radiance.convolve(2.0,'G')` uses a Gaussian response function with HWHM 2.0 cm^{-1}. The `convolve` special method is also available for the `odArray` and `wfArray` classes (in the first case the convolution operates on the corresponding transmission).

3.2.8. Shortcuts

If some of the intermediate quantities are not required, it is possible to go directly from line and atmospheric data to optical depths using `dodList = lbl2od (mls, llDict)`. Likewise, cross sections and absorption coefficients can be bypassed with the `lbl2ac` and `xs2dod` functions. Please note that `lbl2ac` and `lbl2od` "inherit" most options accepted by `lbl2xs` or `xs2ac`.

4. Discussion

4.1. Selection of Spectral Range, Contributions from Line Wings

To compute cross sections, absorption coefficients, and optical depths for some spectral range $\nu_{\text{lo}} \ldots \nu_{\text{hi}}$, all lines in an extended spectral range $\nu_{\text{lo}} - \delta \ldots \nu_{\text{hi}} + \delta$ should be considered, where δ is typically some wavenumbers (cm^{-1}). However, unless specified as fourth optional argument `xLimits` of the `lbl2xs` function or as option `-x` of the command line script `lbl2xs.py` (and similarly for the `lbl2ac` and `lbl2od` functions), the cross sections, absorption coefficients, and optical depths returned by Py4CAtS are computed on a uniform grid in the interval $[\nu_{\text{first}}, \nu_{\text{last}}]$ where the lower and upper limits correspond to the position of the very first and last line returned by `higstract`. As the `higstract` and `lbl2xs` scripts are completely independent, this extension is not done automatically. The impact of line wing contributions on cross sections is demonstrated in Figure 7.

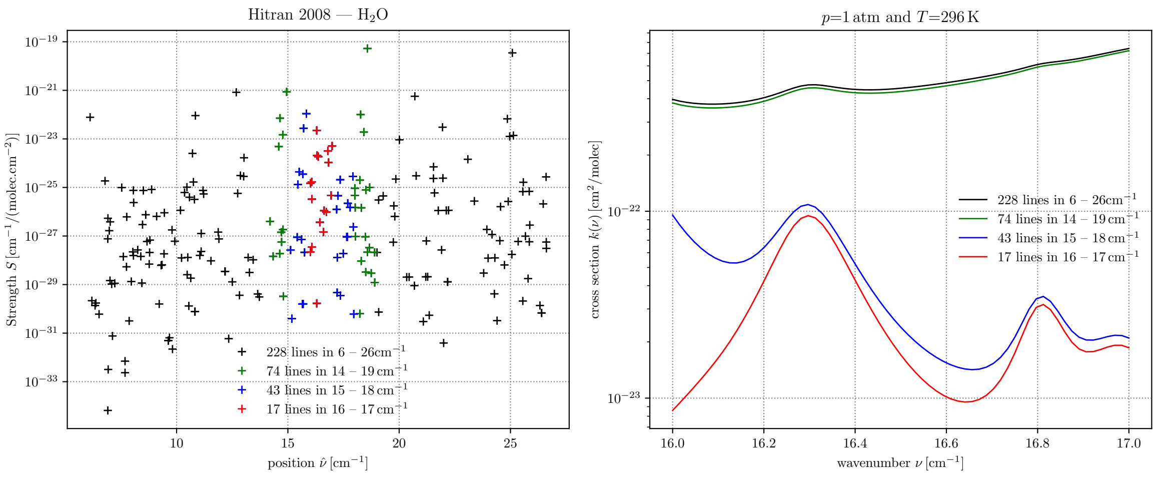

Figure 7. Impact of line wings on H_2O cross section in the ODIN [83] 501 GHz channel. A series of cross sections has been computed taking into account more and more lines to the left and right of the 16 to 17 cm^{-1} window.

4.2. Optical Depths, Transmissions, and Weighting Functions for a Horizontal View

The functions `ac2dod` or `lbl2od` do not have an angle as argument, so the optical depth returned is always the vertical optical depth through the atmosphere. If transmission (or weighting functions) for a horizontal path (i.e., zenith angle 90°, hence a homogeneous atmosphere) are needed, the transmission $\mathcal{T}(\nu, s) = \exp(-\alpha(\nu)l)$ can be readily evaluated as a function of path length l given an appropriate absorption coefficient α. Likewise, the weighting functions are easily computed according to (13) as the product of transmission $\mathcal{T}(\nu, s)$ times the absorption coefficient α for some lengths l.

4.3. Arbitrary Observer Positions

An observer "inside" a layer, i.e., with the observer altitude different from any atmospheric altitude grid point, is not supported by Py4CAtS. However, if one needs an observer at a specific altitude (e.g., 3.14159 km), one can interpolate the atmospheric profiles to a new grid including this point and proceed as usual.

4.4. GARLIC vs. Py4CAtS

As indicated above, Py4CAtS has originally been a (partial) re-implementation of GARLIC. GARLIC has been thoroughly verified by intercomparisons with other lbl codes such as ARTS and KOPRA (e.g., [74–76]). Furthermore, it has been recently validated by intercomparison with spectra of the ACE-FTS instrument [84–86]. Figure 8 shows an intercomparison of GARLIC and Py4CAtS brightness temperature spectra. For the monochromatic spectra differences are mostly below one Kelvin except for a few spikes near strong lines. The convolved spectra agree within some hundredth Kelvin.

Please note that in principle the radiance spectrum could have been generated with a single statement `dod2ri(lbl2od(atmRead('mls.xy', higstract('lines',(2100,2150)), 180., mls['T'][0]);` however, to properly account for line wing contributions (cf. discussion in Section 4.1) the spectral interval has been specified explicitly.

Despite the efficiency of NumPy's array processing GARLIC is significantly faster than Py4CAtS. In GARLIC the computation of molecular cross sections, absorption coefficients, and radiances is exploiting the multi-core architecture of modern CPUs, i.e., parallelization by means of OpenMP (see section 3.3 of the GARLIC paper [25]). This allows about a factor 50 speed-up for the Fortran-OpenMP implementation compared to the NumPy implementation on an eight-core desktop for the radiance spectra shown here.

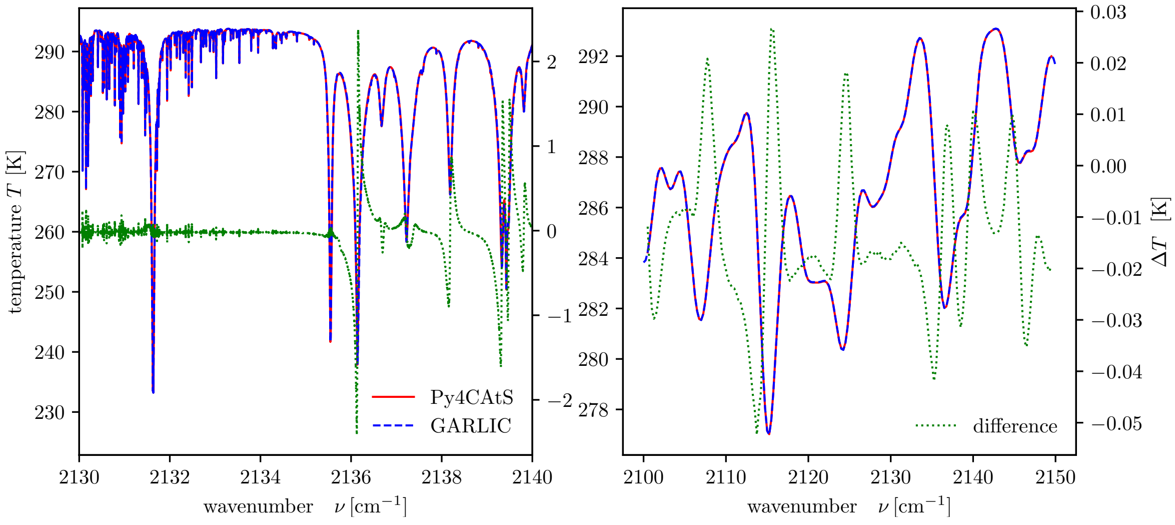

Figure 8. Intercomparison of equivalent brightness temperature spectra with corresponding spectra generated with GARLIC and their difference (axis on right). Midlatitude summer atmosphere, downlooking observer at ToA with viewing zenith angle 180°. (**Left**) monochromatic spectra; (**Right**) radiance convolved with a Gaussian response function with HWHM = 1 cm^{-1}. The Py4CAtS spectra have been generated with

```
vLimits = Interval(2100.0,2150.0)
mls = atmRead('/data/atmos/50/mls.xy', zToA=100)
dll = higstract('/data/hitran/2000/lines',vLimits+20, 'main') # dict. of line lists
radNadir = dod2ri(lbl2od(mls,dll,vLimits+5),180.,mls['T'][0]) # monochrom radiance   .
radNadirG = radNadir.convolve(1.0,'G') # Gauss spectral response
btNadir = radiance2Kelvin(radNadir.grid(),radNadir) # equ brightness temperature
btNadirG = radiance2Kelvin(radNadirG.grid(),radNadirG)
```

4.5. Batch Processing

Given line and atmospheric data, a radiance spectrum is essentially generated by the sequence `atmRead`, `higstract`, `lbl2od`, and `dod2ri`. Likewise, atmospheric transmission can be modeled with the fourth and last step `exp(-dod2tod(...))`. If a series of synthetic spectra must be modeled, a combination of these functions into a single function might be convenient. However, such a function is currently not provided in Py4CAtS because the appropriate implementation is clearly depending on the actual task: For an assessment of the impact of spectroscopic data or spectral range on atmospheric radiances, a function combining `higstract - lbl2od - dod2ri` would be adequate (with atmospheric data ingested only once), whereas the impact of atmospheric data on radiances could be studied with the combination of `atmRead - lbl2od - dod2ri`. Furthermore, the main modeling task can easily be performed with a single statement such as `dod2ri(lbl2od(atmData,llDict),...)`. Indeed, this can also be easily used for retrieval purposes, e.g., by a combination with the nonlinear least squares solvers of the `scipy.optimize` package.

4.6. Current limitations—What Py4CAtS Cannot Do

- Spherical atmospheres: modeling radiance and/or transmission for limb sounding is not possible; Py4CAtS is assuming a plane-parallel atmosphere. Please note that the quadrature schemes of Section 2.2.2 also work for limb geometry.
- Continuum or collision-induced absorption [87] contributions to molecular absorption are not considered. To some extent this is a rather controversial aspect esp. for the water continuum, and several codes and/or data are available (e.g., [88–93]).
- Jacobians: Several tools have been developed for automatic differentiation of Python code (see http://www.autodiff.org), so derivatives of spectra w.r.t. atmospheric parameters etc. could be implemented similar to the approach used in GARLIC [94]. This is currently not foreseen.
- Other line parameter databases (see also http://hitran.org/links/): In addition to HITRAN/HITEMP and GEISA, further databases have been developed such as ExoMol [19] and databases dedicated to specific spectral regions (e.g., JPL and CDMS catalogs, Pickett et al. [16], Endres et al. [18]), specific molecules (e.g., [95–97]), or satellite missions [96,98]. Some of these databases have a format similar to HITRAN and GEISA and an appropriate reader could be readily implemented, whereas an implementation of other databases would require some more effort because of their different organization. Please note that a Python script `ExoMol_to_HITRAN.py` has been developed by the ExoMol consortium, see https://github.com/xnx/ExoMol_to_HITRAN.
- Predefined cross sections: Both HITRAN and GEISA include a large collection of IR (and UV) cross sections esp. for heavy molecules that can be relevant for atmospheric absorption. Reading and further processing of these data is not yet implemented in Py4CAtS.
- Scattering: So far only the Schwarzschild equation with thermal emission as source (i.e., Planck function) is supported. However, the optical depths can be used as input for any multiple scattering solver (e.g., DISORT, [99]), see libRadtran [100,101].
- Line shapes beyond Voigt: some advanced profiles for modeling line-mixing, Dicke narrowing, or speed-dependence [61,62] have been implemented recently.
- Py4CAtS stores only the "core" line parameters required for cross section modeling. In contrast, HAPI [34] can also keep track of line assignments, transition IDs etc. However, Py4CAtS has been developed with atmospheric spectroscopy as target application, but not molecular spectroscopy.

5. Conclusions

Py4CAtS, a Python package developed for "computational atmospheric spectroscopy", has been presented and its usage in a Unix-like console or within the (I)Python interpreter/notebook (recommended) has been demonstrated. Starting with line data from HITRAN or GEISA and atmospheric data, the scripts and functions allow the computation of cross sections, absorption coefficients, optical depths, weighting functions, and radiances. In particular, they also provide easy access to all intermediate quantities esp. for visualization.

When the development of Py4CAtS started, lbl modeling with Python appeared to be quite ambitious. However, thanks to NumPy and its dramatic performance improvements, atmospheric radiative transfer modeling with Python can now be done with reasonable speed. In fact, our recent benchmark tests of Voigt and complex error function algorithms indicated that NumPy implementations are not significantly slower than Fortran implementations [102].

As emphasized above, the intention for Py4CAtS has not been a highly efficient and accurate lbl radiative transfer code, and the package is not considered to be a competitor for the codes mentioned in the introduction. Nevertheless, despite the speed limitations of interpreters such as Python and the omission of limb geometry, continuum, or scattering Py4CAtS is believed to be attractive because of the flexibility and ease of use. The future development of Py4CAtS will certainly be driven by further optimizations and extensions of its functionality.

Supplementary Materials: Py4CAtS is free software and can be redistributed and/or modified under the terms of the GNU General Public License as published by the Free Software Foundation. A tarball of the Python source files is available at our department's server at https://atmos.eoc.dlr.de/tools/Py4CAtS/. Py4CAtS is also registered at the Astrophysics Source Code Library (ASCL) with the permalink http://ascl.net/1905.002. The sources have been thoroughly checked with Pep8, PyFlakes, PyChecker, and PyLint. In addition to the sources, the model atmospheres of the Anderson et al. [78] compilation are available in various resolutions (number of altitude levels), and an excerpt of the HITRAN 1986 database [103]. Furthermore, a directory `doc` provides a tutorial/users' guide along with extensive documentation of the individual modules. The html files have been produced with `pydoc3`.

Author Contributions: Conceptualization and methodology, F.S.; Software, F.S. with support from all co-authors; Validation, all authors; Writing—original draft preparation, F.S.; Writing—review and editing, all authors.

Funding: Development of PyCAtS has been partially supported by several projects, especially the ESA-ESTEC study ESASlight (contract No. AO/1-5433/07/NL/HE). Furthermore, funding by the World Data Center for Remote Sensing of the Atmosphere (WDC-RSAT) and in the DFG project SCHR 1125/3-1 is gratefully acknowledged.

Acknowledgments: Thomas Trautmann and Mathias Milz have used Py4CAtS in lectures and seminars at the Universität Leipzig (Germany) and at the Luleå University of Technology (Kiruna, Sweden), respectively. Thanks also go to Thomas Trautmann and Mourad Hamidouche for careful reading of the manuscript. Finally, numerous students and colleagues gave valuable feedback and recommendations, most notably Dmitry Efremenko, Mourad Hamidouche, Michael Hess, Michał Szajkowski, Haoran Xu, Jian Xu (DLR-IMF), Mareike Kenntner (DLR-IPA), and Claudia Emde and Bernhard Mayer (LMU-MIM).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are frequently used in this manuscript:

BoA	Bottom of Atmosphere
GARLIC	Generic Atmospheric Radiation Line-by-line Infrared Code
HWHM	half width half maximum
IR	infrared
lbl	line-by-line
Py4CAtS	PYthon scripts for Computational ATmospheric Spectroscopy
ToA	Top of Atmosphere
ac	absorption coefficient
od	optical depth
ri	radiance intensity
wf	weighting function
vmr	volume mixing ratio
xs	cross section

Appendix A. Implementation

In the first three subsections some common aspects of reading, writing, and plotting of the data are discussed. Special data types implemented in Py4CAtS are described in Appendices A.4 and A.5. Finally, three modules that should be of interest not only for "computational atmospheric spectroscopy" are presented.

Appendix A.1. Input/Output

Atmospheric data, cross sections, absorption coefficients, optical depths, weighting functions, and radiance spectra can be written to data files with the functions `atmSave`, `xsSave`, `acSave`, `odSave`, `wfSave`, and `riSave`. Likewise, the data are read from file using `atmRead`, `xsRead`, `acRead`, `odRead`, wfRead, and riRead. In all cases tabular ASCII files are supported, in some cases Python's / NumPy's pickle format or netcdf I/O is also available. Cross section and absorption coefficient files can also use the HITRAN format. (The HITRAN cross section format is extensively described at https://hitran.org/docs/cross-sections-definitions/. In brief, each "portion" starts with a header line defining the molecule, wavenumber range, number of data in this set, temperature (K), pressure (Torr) etc. Following this header, the "values are arranged in records containing ten values of fields of ten for each cross-section".)

For completeness, the higstract function to read the HITRAN and GEISA database (and extract some lines) should be mentioned here, too. The extracted lines can be saved to file with save_lines_core for the "core parameters" only (i.e., position, strength etc.) or save_lines_orig for the original format (Please note that there is no tool to convert data from HITRAN to GEISA format or back). To read a set of line data files (HITRAN/GEISA extracts of core parameters) use read_line_file that returns a single "lineArray" for a single file (molecule) or a dictionary thereof.

All routines saving data in ASCII format use the awrite function from the aeiou module, see Appendix A.8 for details.

Appendix A.2. Visualization

The functions atmPlot, atlas, xsPlot, acPlot, odPlot, riPlot, and wfPlot can be used to plot atmospheric profiles (default temperature vs. altitude, with z (default, or p) as vertical abscissa), spectroscopic line data (default strength vs. position), molecular absorption cross sections and coefficients, optical depths, radiance/intensity, and weighting functions. Please note that these functions can be called with a single array or a list of arrays (exploiting Python's recursive capabilities, see Appendix A.3). The name atlas goes back to the *Atlas of absorption lines from 0 to 17,900* cm^{-1} by Park et al. [82] that served as pictorial representation of the HITRAN 86 database.

Please note that these functions serve to provide quicklooks of the various spectra etc. and are not designed for highly fancy, publication-ready plots. However, the source code of these functions can be exploited as a starting point for more sophisticated plots. And x or y axis labels and limits, legend (entries, positions, ...), title, and curve colors, markers, styles and widths can be changed interactively in IPython/Matplotlib.

Appendix A.3. Recursive Functions

Some functions exploit Python's recursive capabilities to make their use as flexible as possible. In particular, lbl2xs can be called with

- a single lineArray holding the line parameters (position, strengths, ...) of a single molecule and a single (p, T) pair (that defaults to STP 1 atm, 296 K);
- a dictionary or list of lineArray's and a single (p, T) pair;
- a single lineArray and a list/array of pressures and/or a list/array of temperatures (if both p and T are arrays (or lists), their length must be identical!);
- a dictionary (list) of lineArray's and (a list/array of) pressure(s) and temperature(s).

Likewise, xsPlot can be called with a single cross section xsArray, or with a (nested) list or dictionary of xsArray's. Similarly, an odArray instance or a list of optical depths can be visualized using odPlot, and acPlot and riPlot work in the same way. Furthermore, atmPlot and atmInfo accept a single or a list of atmospheric data. Finally, ac2wf is called recursively in case of a finite field-of-view.

Appendix A.4. The Subclassed NumPy Arrays

The spectra of molecular cross sections, absorption coefficients, (layer, cumulative, and total) optical depths, weighting functions, and radiances are stored in subclassed NumPy arrays to hold extra information as attributes, e.g., the minimum and maximum wavenumber is stored in xs.x, ac.x, and od.x, respectively (technically the x attribute is an instance of the Interval class defined in the `pairTypes.py` module, see Appendix A.9). Pressure and temperature of the cross section and absorption coefficient are single floats for the corresponding atmospheric level, whereas for optical depths they are `PairOfFloats` corresponding to the atmospheric layer.

In addition to these attributes, `xsArray` defines several methods, e.g., (note the parentheses!)

`xs.info()` — print "essential" information;
`xs.dx()` — compute the grid point spacing (essentially `xs.x.size()/(len(xs)-1)`);
`xs.grid()` — returns the uniform wavenumber grid array;
`xs.regrid(n)` — interpolate to a denser uniform grid (usually n is larger than `len(xs)`);
`xs.__eq__(other)` — compare two cross sections using the == operator, i.e., `xs1==xs2` returns `True` if the wavenumber intervals, pressure, temperature, and the spectra itself agree (approximately).

The same methods are also defined for `acArray`, `odArray`, and `riArray`, where in addition there is also a method `truncate` returning the spectrum in a smaller wavenumber interval. Furthermore, a method `convolve` has been implemented in the `riArray` class to smooth the radiance with a box, triangle, or Gaussian spectral response function.

For `odArray` there are also `__add__` and `__sub__` methods to add and subtract optical depths, see Figure 5. These combinations can only be performed if both spectra are defined in the same wavenumber interval, i.e., `od1.x` and `od2.x` are identical. In general, the two optical depths are given on different wavenumber grids, so the coarser spectrum is regridded to the resolution of the denser spectrum first. The `__mul__` method can be used to scale optical depths with a (float) number, e.g., to account for a slant path `od/cosdg(60)`.

The core parameters (position, strength, width, ...) of the lines extracted from the HITRAN or GEISA databases are also saved internally in a subclassed array `lineArray`. However, in contrast to the arrays mentioned above (that all have just a single dimension) this has "rows and columns", where the rows correspond to the spectral lines/transitions, and the columns correspond to center wavenumber $\hat{\nu}_l$, line strength S_l, etc. To make these columns easily accessible, `lineArray` is a subclassed structured array (see next section Appendix A.5) with attributes holding information about molecule and reference pressure and temperature.

Appendix A.5. Structured Arrays

As described in the NumPy User Guide http://docs.scipy.org/doc/numpy/user/basics.rec.html, "These arrays permit one to manipulate the data by the structs or by fields of the struct". The main difference to standard NumPy arrays is that one can access the "columns" of these arrays by names (strings) instead of numbers, similar to dictionary entries.

Py4CAtS uses structured arrays for the atmospheric data (see Section 3.2.1) and the line parameters (see Section 3.2.2, actually `lineArray` is a subclassed *and* structured NumPy array). Several utility functions are collected in the `struc_array.py` module: `loadStrucArray` reads tabular ASCII files and automatically assign names to the fields (columns) given the information in the file header or as an optional argument to the function. Further functions allow the changing of these field names, to insert additional fields, to extract fields, or to delete a field.

Appendix A.6. Conversion of Physical Units: The `cgsUnits` Module

The function `cgs` from the `cgsUnits.py` module can be used to convert physical quantities to or from the cgs base unit, e.g., `cgs('kg')` $\rightarrow$ `1000.` or `cgs('!km')` $\rightarrow$ `1e − 5` or `cgs('mb!atm', 1013.25)` $\rightarrow$ `1.0` (where the exclamation mark separates the original (input) and final (output) unit). Please note that the optional second argument "`data`" can be a float, list of floats, or a NumPy array.

Two further modules `radiance2radiance.py` and `radiance2Kelvin.py` help to convert radiances. With the function `radiance2radiance` one can change the power and/or area and/or spectral unit of radiances, e.g., nW $\leftrightarrow$ erg/s or wavenumber $\leftrightarrow$ frequency $\leftrightarrow$ wavelength. Likewise, `radiance2Kelvin` allows the conversion of radiances to equivalent brightness temperatures using the "inverse" of the Planck function (11).

Appendix A.7. The Option Parser Module `command_parser.py`

When development of the Py4CAtS tools started, Python only offered the `getopt` module providing only limited functionality. Parsing the arguments and options given on the (Unix/Linux) shell command line essentially comprises a series of common tasks, e.g., type checking and conversion, so these extra steps were finally implemented in a new module `command_parser.py` building on top of `getopt.py`. Later on, a new module `optparse` had been added to the Python Standard Library, which is now superseded by the newer and more advanced `argparse` module.

Although `argparse` has some nice features currently not available in `command_parser`, it also has one serious deficiency related to range checks of the given input. In case of integer or float input arguments/options it is frequently required to check if the number falls within a certain range, e.g., pressures and temperatures should be positive, or the percentage concentration should be in the range $(0.0, 100.0)$. `argparse` only can check if the input is in a given list of possibilities, e.g., `diceNumber in [1,2,3,4,5,6]`, but a statement such as `200<=temperature<=300` cannot be used. (Several solutions are discussed in the web, but none of them appears attractive.) And more sophisticated checks such as `constraint='all([digit.strip().isdigit() for digit in split(columns,",")])'` are impossible.

Appendix A.8. Input/Output Utilities: The `aeiou.py` *Module*

A set of functions for some common tasks as reading and parsing the comments in the file header are collected in this module. In addition, there is the `awrite` function that serves as a "slightly better" version of NumPy's `savetxt` function: the format option is more intelligent, and instead of a single `header` string to be written to the file header `awrite` also accepts a list of strings for the header. Most importantly, there is only a single mandatory argument: the data array to save/write. If no output file is given, `awrite` prints the output on the screen (`sys.stdout`). Accordingly, the sequence of arguments is changed from `savetxt(fname,data,...)` to `awrite(data,fname=None,...)`. Furthermore, `awrite` makes it easier to save several NumPy arrays (all with the same number of rows), for example `awrite ([xGrid, yValues, aMatrix], 'allInOne.file')`.

Appendix A.9. The `pairTypes.py` *Module*

This module defines several classes for `Interval`, `pairOfInts`, and `PairOfFloats`. The `Interval` is frequently used in Py4CAtS, e.g., for the wavenumber region of interest: `xLimits=Interval(10.,20.)`.

References

1. Goody, R.; Yung, Y. *Atmospheric Radiation—Theoretical Basis*, 2nd ed.; Oxford University Press: Oxford, UK, 1989.
2. Liou, K.N. *An Introduction to Atmospheric Radiation*; Academic Press: Orlando, FL, USA, 1980.
3. Zdunkowski, W.; Trautmann, T.; Bott, A. *Radiation in the Atmosphere—A Course in Theoretical Meteorology*; Cambridge University Press: Cambridge, UK, 2007.
4. Wiscombe, W. Atmospheric Radiation: 1975–1983. *Rev. Geophys. Space Phys.* **1983**, *21*, 997. [CrossRef]
5. Schläpfer, D. MODO: An interface to MODTRAN for the simulation of imaging spectrometry at-sensor signals. In Proceedings of the Tenth Jet Propulsion Laboratory Airborne Earth Science Workshop, Pasadena, CA, USA, 27 February–2 March 2001; Green, R., Ed.; Volume JPL-02-1, pp. 343–350.
6. Berk, A.; Anderson, G.; Acharya, P.; Bernstein, L.; Muratov, L.; Lee, J.; Fox, M.; Adler-Golden, S.; Chetwynd, J.; Hoke, M.; et al. MODTRAN 5: A reformulated atmospheric band model with auxiliary species and practical multiple scattering options: Update. In Proceedings of the Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XI, Orlando, FL, USA, 28 March–1 April 2005; Shen, S., Lewis, P., Eds.; Volume 5806, pp. 662–667.
7. Wilson, R. Py6S: A Python interface to the 6S radiative transfer model. *Comput. Geosci.* **2013**, *51*, 166–171. [CrossRef]
8. Vermote, E.; Tanré, D.; Deuzé, J.; Herman, M.; Morcette, J.J. Second Simulation of the Satellite Signal in the Solar Spectrum, 6S: An Overview. *IEEE Trans. Geosci. Remote Sens.* **1997**, *35*, 675–686. [CrossRef]

9. Lacis, A.; Oinas, V. A Description of the Correlated *k* Distribution Method for Modeling Nongray Gaseous Absorption, Thermal Emission, and Multiple Scattering in Vertically Inhomogeneous Atmospheres. *J. Geophys. Res.* **1991**, *96*, 9027–9063. [CrossRef]
10. Wiscombe, W.; Evans, J. Exponential-sum fitting of radiative transmission functions. *J. Comput. Phys.* **1977**, *24*, 416–444. [CrossRef]
11. Heng, K.; Marley, M. Radiative Transfer for Exoplanet Atmospheres. In *Handbook of Exoplanets*; Deeg, H.J., Belmonte, J.A., Eds.; Springer International Publishing: New York, NY, USA, 2018; pp. 2137–2152.
12. Madhusudhan, N. Atmospheric Retrieval of Exoplanets. In *Handbook of Exoplanets*; Deeg, H.J., Belmonte, J.A., Eds.; Springer: New York, NY, USA, 2018.
13. Gordon, I.E.; Rothman, L.S.; Hill, C.; Kochanov, R.V.; Tan, Y.; Bernath, P.F.; Birk, M.; Boudon, V.; Campargue, A.; Chance, K.V.; et al. The HITRAN2016 molecular spectroscopic database. *J. Quant. Spectrosc. Radiat. Transf.* **2017**, *203*, 3–69. [CrossRef]
14. Rothman, L.; Gordon, I.; Barber, R.; Dothe, H.; Gamache, R.; Goldman, A.; Perevalov, V.; Tashkun, S.; Tennyson, J. HITEMP, the high-temperature molecular spectroscopic database. *J. Quant. Spectrosc. Radiat. Transf.* **2010**, *111*, 2139–2150. [CrossRef]
15. Jacquinet-Husson, N.; Armante, R.; Scott, N.A.; Chédin, A.; Crépeau, L.; Boutammine, C.; Bouhdaoui, A.; Crevoisier, C.; Capelle, V.; Boonne, C.; et al. The 2015 edition of the GEISA spectroscopic database. *J. Mol. Spectrosc.* **2016**, *327*, 31–72. [CrossRef]
16. Pickett, H.; Poynter, R.; Cohen, E.; Delitsky, M.; Pearson, J.; Müller, H. Submillimeter, millimeter, and microwave spectral line catalog. *J. Quant. Spectrosc. Radiat. Transf.* **1998**, *60*, 883–890. [CrossRef]
17. Müller, H.; Schlöder, F.; Stutzki, J.; Winnewisser, G. The Cologne Database for Molecular Spectroscopy, CDMS: A useful tool for astronomers and spectroscopists. *J. Mol. Struct.* **2005**, *742*, 215–227. [CrossRef]
18. Endres, C.P.; Schlemmer, S.; Schilke, P.; Stutzki, J.; Müller, H.S. The Cologne Database for Molecular Spectroscopy, CDMS, in the Virtual Atomic and Molecular Data Centre, VAMDC. *J. Mol. Spectrosc.* **2016**, *327*, 95–104. [CrossRef]
19. Tennyson, J.; Yurchenko, S.N.; Al-Refaie, A.F.; Barton, E.J.; Chubb, K.L.; Coles, P.A.; Diamantopoulou, S.; Gorman, M.N.; Hill, C.; Lam, A.Z.; et al. The ExoMol database: Molecular line lists for exoplanet and other hot atmospheres. *J. Mol. Spectrosc.* **2016**, *327*, 73–94. [CrossRef]
20. Clough, S.; Kneizys, F.; Rothman, L.; Gallery, W. Atmospheric transmittance and radiance: FASCOD1B. *Proc. SPIE* **1981**, *277*, 152–166.
21. Scott, N.; Chédin, A. A Fast Line-by-Line Method for Atmospheric Absorption Computations: The Automatized Atmospheric Absorption Atlas. *J. Appl. Meteorol.* **1981**, *20*, 802–812. [CrossRef]
22. Buehler, S.A.; Eriksson, P.; Kuhn, T.; von Engeln, A.; Verdes, C. ARTS, the atmospheric radiative transfer simulator. *J. Quant. Spectrosc. Radiat. Transf.* **2005**, *91*, 65–93. [CrossRef]
23. Eriksson, P.; Buehler, S.A.; Davis, C.; Emde, C.; Lemke, O. ARTS, the atmospheric radiative transfer simulator, version 2. *J. Quant. Spectrosc. Radiat. Transf.* **2011**, *112*, 1551–1558. [CrossRef]
24. Buehler, S.A.; Mendrok, J.; Eriksson, P.; Perrin, A.; Larsson, R.; Lemke, O. ARTS, the atmospheric radiative transfer simulator—Version 2.2, the planetary toolbox edition. *Geosci. Model Dev.* **2018**, *11*, 1537–1556. [CrossRef]
25. Schreier, F.; Gimeno García, S.; Hedelt, P.; Hess, M.; Mendrok, J.; Vasquez, M.; Xu, J. GARLIC—A General Purpose Atmospheric Radiative Transfer Line-by-Line Infrared-Microwave Code: Implementation and Evaluation. *J. Quant. Spectrosc. Radiat. Transf.* **2014**, *137*, 29–50. [CrossRef]
26. Edwards, D. Atmospheric transmittance and radiance calculations using line–by–line computer models. In Proceedings of the Modelling of the Atmosphere, Orlando, FL, USA, 4–8 April 1988; Volume 928, pp. 94–116.
27. Stiller, G.; von Clarmann, T.; Funke, B.; Glatthor, N.; Hase, F.; Höpfner, M.; Linden, A. Sensitivity of trace gas abundances retrievals from infrared limb emission spectra to simplifying approximations in radiative transfer modelling. *J. Quant. Spectrosc. Radiat. Transf.* **2002**, *72*, 249–280. [CrossRef]
28. Clough, S.; Shephard, M.; Mlawer, E.; Delamere, J.; Iacono, M.; Cady-Pereira, K.; Boukabara, S.; Brown, P. Atmospheric radiative transfer modeling: A summary of the AER codes. *J. Quant. Spectrosc. Radiat. Transf.* **2005**, *91*, 233–244. [CrossRef]
29. Gordley, L.; Marshall, B.; Chu, D. LINEPAK: Algorithms for modeling spectral transmittance and radiance. *J. Quant. Spectrosc. Radiat. Transf.* **1994**, *52*, 563. [CrossRef]

30. Dudhia, A. The Reference Forward Model (RFM). *J. Quant. Spectrosc. Radiat. Transf.* **2017**, *186*, 243–253. [CrossRef]
31. Amato, U.; Masiello, G.; Serio, C.; Viggiano, M. The σ–IASI code for the calculation of infrared atmospheric radiance and its derivatives. *Environ. Model. Softw.* **2002**, *17*, 651–667. [CrossRef]
32. Bailey, J.; Kedziora-Chudczer, L. Modelling the spectra of planets, brown dwarfs and stars using VStar. *Mon. Not. R. Astron. Soc.* **2012**, *419*, 1913–1929. [CrossRef]
33. Hill, C.; Gordon, I.E.; Kochanov, R.V.; Barrett, L.; Wilzewski, J.S.; Rothman, L.S. HITRANonline: An online interface and the flexible representation of spectroscopic data in the HITRAN database. *J. Quant. Spectrosc. Radiat. Transf.* **2015**, *177*, 4–14. [CrossRef]
34. Kochanov, R.; Gordon, I.; Rothman, L.; Wcisło, P.; Hill, C.; Wilzewski, J. HITRAN Application Programming Interface (HAPI): A comprehensive approach to working with spectroscopic data. *J. Quant. Spectrosc. Radiat. Transf.* **2016**, *177*, 15–30. [CrossRef]
35. Goldenstein, C.; Miller, V.; Spearrin, R.; Strand, C. SpectraPlot.com: Integrated spectroscopic modeling of atomic and molecular gases. *J. Quant. Spectrosc. Radiat. Transf.* **2017**, *200*, 249–257. [CrossRef]
36. Villanueva, G.; Smith, M.; Protopapa, S.; Faggi, S.; Mandell, A. Planetary Spectrum Generator: An accurate online radiative transfer suite for atmospheres, comets, small bodies and exoplanets. *J. Quant. Spectrosc. Radiat. Transf.* **2018**, *217*, 86–104. [CrossRef]
37. Smette, A.; Sana, H.; Noll, S.; Horst, H.; Kausch, W.; Kimeswenger, S.; Barden, M.; Szyszka, C.; Jones, A.M.; Gallenne, A.; et al. Molecfit: A general tool for telluric absorption correction. *Astron. Astrophys.* **2015**, *576*, A77. [CrossRef]
38. Bertaux, L.; Lallement, R.; Ferron, S.; Boonne, C.; Bodichon, R. TAPAS, a web-based service of atmospheric transmission computation for astronomy. *Astron. Astrophys.* **2014**, *564*, A46. [CrossRef]
39. Schreier, F.; Böttger, U. MIRART, A Line-By-Line Code for Infrared Atmospheric Radiation Computations incl. Derivatives. *Atmos. Ocean. Opt.* **2003**, *16*, 262–268.
40. Ernst, T.; Rother, T.; Schreier, F.; Wauer, J.; Balzer, W. DLR's Virtual Lab: Scientific Software just a mouse click away. *Comput. Sci. Eng.* **2003**, *5*, 70–79. [CrossRef]
41. von Clarmann, T.; Echle, G. Selection of Optimized Microwindows for Atmospheric Spectroscopy. *Appl. Opt.* **1998**, *37*, 7661–7669. [CrossRef] [PubMed]
42. Echle, G.; von Clarmann, T.; Dudhia, A.; Flaud, J.M.; Funke, B.; Glatthor, N.; Kerridge, B.; López-Puertas, M.; Martin-Torres, F.; Stiller, G. Optimized Spectral Microwindows for Data Analysis of the Michelson Interferometer for Passive Atmospheric Sounding on the Environmental Satellite. *Appl. Opt.* **2000**, *39*, 5531–5540. [CrossRef]
43. Dudhia, A.; Jay, V.; Rodgers, C. Microwindow selection for high-resolution-sounders. *Appl. Opt.* **2002**, *41*, 3665–3673. [CrossRef] [PubMed]
44. Rabier, F.; Fourrié, N.; Chafai, D.; Prunet, P. Channel Selection Methods for Infrared Atmospheric Sounding Interferometer Radiances. *Quart. J. R. Met. Soc.* **2002**, *128*, 1011–2027. [CrossRef]
45. Crevoisier, C.; Chedin, A.; Scott, N.A. AIRS channel selection for CO_2 and other trace-gas retrievals. *Quart. J. R. Met. Soc.* **2003**, *129*, 2719–2740. [CrossRef]
46. Kuai, L.; Natraj, V.; Shia, R.; Miller, C.; Yung, Y. Channel selection using information content analysis: A case study of CO_2 retrieval from near infrared measurements. *J. Quant. Spectrosc. Radiat. Transf.* **2010**, *111*, 1296–1304. [CrossRef]
47. Dubois, P.; Hinsen, K.; Hugunin, J. Numerical Python. *Comput. Phys.* **1996**, *10*, 262–267. [CrossRef]
48. Watters, A.; van Rossum, G.; Ahlstrom, J.C. *Internet Programming with Python*; M & T Books: New York, NY, USA, 1996.
49. Oliphant, T. Python for Scientific Computing. *Comput. Sci. Eng.* **2007**, *9*, 10–20. [CrossRef]
50. Langtangen, H.P. *Python Scripting for Computational Science*, 3rd ed., Texts in Computational Science and Engineering; Springer: New York, NY, USA, 2008; Volume 3.
51. van der Walt, S.; Colbert, S.; Varoquaux, G. The NumPy Array: A Structure for Efficient Numerical Computation. *Comput. Sci. Eng.* **2011**, *13*, 22–30. [CrossRef]
52. Pérez, F.; Granger, B.; Hunter, J. Python: An Ecosystem for Scientific Computing. *Comput. Sci. Eng.* **2011**, *13*, 13–21. [CrossRef]
53. Lin, J.W.B. Why Python is the next wave in earth sciences computing. *Bull. Am. Met. Soc.* **2012**, *93*, 1823–1824. [CrossRef]

54. Raspaud, M.; Hoese, D.; Dybbroe, A.; Lahtinen, P.; Devasthale, A.; Itkin, M.; Hamann, U.; Rasmussen, L.Ø.; Nielsen, E.; Leppelt, T.; et al. PyTroll: An Open-Source, Community-Driven Python Framework to Process Earth Observation Satellite Data. *Bull. Am. Met. Soc.* **2018**, *99*, 1329–1336. [CrossRef]
55. Astropy Collaboration; Robitaille, T.P.; Tollerud, E.J.; Greenfield, P.; Droettboom, M.; Bray, E.; Aldcroft, T.; Davis, M.; Ginsburg, A.; Price-Whelan, A.; et al. Astropy: A community Python package for astronomy. *Astron. Astrophys.* **2013**, *558*, A33.
56. Parker, S.; Johnson, C.; Beazley, D. Computational Steering Software Systems and Strategies. *IEEE Comput. Sci. Eng.* **1997**, *4*, 50–59. [CrossRef]
57. Dubois, P.; Yang, T.Y. Extending Python with Fortran. *Comput. Sci. Eng.* **1999**, *1*, 66–73. [CrossRef]
58. Peterson, P. F2PY: A tool for connecting Fortran and Python programs. *Int. J. Comput. Sci. Eng.* **2009**, *4*, 296–305. [CrossRef]
59. Norton, R.; Rinsland, C. ATMOS data processing and science analysis methods. *Appl. Opt.* **1991**, *30*, 389–400. [CrossRef]
60. Armstrong, B. Spectrum Line Profiles: The Voigt Function. *J. Quant. Spectrosc. Radiat. Transf.* **1967**, *7*, 61–88. [CrossRef]
61. Varghese, P.; Hanson, R. Collisional narrowing effects on spectral line shapes measured at high resolution. *Appl. Opt.* **1984**, *23*, 2376–2385. [CrossRef]
62. Tennyson, J.; Bernath, P.; Campargue, A.; Császár, A.; Daumont, L.; Gamache, R.; Hodges, J.; Lisak, D.; Naumenko, O.; Rothman, L.; et al. Recommended isolated-line profile for representing high-resolution spectroscopic transitions (IUPAC Technical Report). *Pure Appl. Chem.* **2014**, *86*, 1931–1943. [CrossRef]
63. Schreier, F. Computational Aspects of Speed-Dependent Voigt Profiles. *J. Quant. Spectrosc. Radiat. Transf.* **2017**, *187*, 44–53. [CrossRef]
64. *NIST Digital Library of Mathematical Functions*; Online Companion to [65]; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2010.
65. Olver, F.; Lozier, D.; Boisvert, R.; Clark, C. (Eds.) *NIST Handbook of Mathematical Functions*; Print Companion to [64]; Cambridge University Press: New York, NY, USA, 2010.
66. Ralston, A.; Rabinowitz, P. *A First Course in Numerical Analysis*, 2nd ed.; McGraw–Hill Book Company: New York, NY, USA, 1978.
67. Cuyt, A.; Petersen, V.; Verdonk, B.; Waadeland, H.; Jones, W. *Handbook of Continued Fractions for Special Functions*; Springer: New York, NY, USA, 2008.
68. Humlíček, J. An efficient method for evaluation of the complex probability function: The Voigt function and its derivatives. *J. Quant. Spectrosc. Radiat. Transf.* **1979**, *21*, 309–313. [CrossRef]
69. Humlíček, J. Optimized computation of the Voigt and complex probability function. *J. Quant. Spectrosc. Radiat. Transf.* **1982**, *27*, 437–444. [CrossRef]
70. Weideman, J. Computation of the Complex Error Function. *SIAM J. Num. Anal.* **1994**, *31*, 1497–1518. [CrossRef]
71. Schreier, F. Optimized Implementations of Rational Approximations for the Voigt and Complex Error Function. *J. Quant. Spectrosc. Radiat. Transf.* **2011**, *112*, 1010–1025. [CrossRef]
72. Whiting, E. An empirical approximation to the Voigt profile. *J. Quant. Spectrosc. Radiat. Transf.* **1968**, *8*, 1379–1384. [CrossRef]
73. Schreier, F. Optimized evaluation of a large sum of functions using a three-grid approach. *Comput. Phys. Commun.* **2006**, *174*, 783–802. [CrossRef]
74. Schreier, F.; Milz, M.; Buehler, S.A.; von Clarmann, T. Intercomparison of three microwave/infrared high resolution line-by-line radiative transfer codes. *J. Quant. Spectrosc. Radiat. Transf.* **2018**, *211*, 64–77. [CrossRef]
75. von Clarmann, T.; Höpfner, M.; Funke, B.; López-Puertas, M.; Dudhia, A.; Jay, V.; Schreier, F.; Ridolfi, M.; Ceccherini, S.; Kerridge, B.; et al. Modeling of Atmospheric Mid–Infrared Radiative Transfer: The AMIL2DA Algorithm Intercomparison Experiment. *J. Quant. Spectrosc. Radiat. Transf.* **2003**, *78*, 381–407. [CrossRef]
76. Melsheimer, C.; Verdes, C.; Buehler, S.A.; Emde, C.; Eriksson, P.; Feist, D.; Ichizawa, S.; John, V.; Kasai, Y.; Kopp, G.; et al. Intercomparison of General Purpose Clear Sky Atmospheric Radiative Transfer Models for the Millimeter/Submillimeter Spectral Range. *Radio Sci.* **2005**, *40*, RS1007. [CrossRef]
77. Hunter, J. Matplotlib: A 2D Graphics Environment. *Comput. Sci. Eng.* **2007**, *9*, 90–95. [CrossRef]
78. Anderson, G.; Clough, S.; Kneizys, F.; Chetwynd, J.; Shettle, E. *AFGL Atmospheric Constituent Profiles (0–120 km)*; Technical Report TR-86-0110; AFGL: Hanscom AFB, MA, USA, 1986.

79. Fortems-Cheiney, A.; Chevallier, F.; Pison, I.; Bousquet, P.; Carouge, C.; Clerbaux, C.; Coheur, P.F.; George, M.; Hurtmans, D.; Szopa, S. On the capability of IASI measurements to inform about CO surface emissions. *Atmos. Chem. Phys.* **2009**, *9*, 8735–8743. [CrossRef]
80. George, M.; Clerbaux, C.; Bouarar, I.; Coheur, P.F.; Deeter, M.; Edwards, D.; Francis, G.; Gille, J.; Hadji-Lazaro, J.; Hurtmans, D.; et al. An examination of the long-term CO records from MOPITT and IASI: Comparison of retrieval methodology. *Atmos. Meas. Tech.* **2015**, *8*, 4313–4328. [CrossRef]
81. McMillan, W.; Barnet, C.; Strow, L.; Chahine, M.; McCourt, M.; Warner, J.; Novelli, P.; Korontzi, S.; Maddy, E.; Datta, S. Daily global maps of carbon monoxide from NASA's Atmospheric Infrared Sounder. *Geophys. Res. Lett.* **2005**, *32*, L11801. [CrossRef]
82. Park, J.; Rothman, L.; Rinsland, C.; Richardson, D.; Namkung, J. *Atlas of Absorption Lines from 0 to 17,900 cm*$^{-1}$; Reference Publication 1188; NASA: Washington, DC, USA, 1987.
83. Murtagh, D.; Frisk, U.; Merino, F.; Ridal, M.; Jonsson, A.; Stegman, J.; Witt, G.; Eriksson, P.; Jiménez, C.; Megie, G.; et al. An overview of the Odin atmospheric mission. *Can. J. Phys.* **2002**, *80*, 309–319. [CrossRef]
84. Hughes, R.; Bernath, P.; Boone, C. ACE infrared spectral atlases of the Earth's atmosphere. *J. Quant. Spectrosc. Radiat. Transf.* **2014**, *148*, 18–21. [CrossRef]
85. Bernath, P.F. The Atmospheric Chemistry Experiment (ACE). *J. Quant. Spectrosc. Radiat. Transf.* **2017**, *186*, 3–16. [CrossRef]
86. Schreier, F.; Städt, S.; Hedelt, P.; Godolt, M. Transmission Spectroscopy with the ACE-FTS Infrared Spectral Atlas of Earth: A Model Validation and Feasibility Study. *Mol. Astrophys.* **2018**, *11*, 1–22. [CrossRef]
87. Richard, C.; Gordon, I.; Rothman, L.; Abel, M.; Frommhold, L.; Gustafsson, M.; Hartmann, J.M.; Hermans, C.; Lafferty, W.; Orton, G.; et al. New section of the HITRAN database: Collision-Induced Absorption (CIA). *J. Quant. Spectrosc. Radiat. Transf.* **2012**, *113*, 1276–1285. [CrossRef]
88. Clough, S.; Kneizys, F.; Davies, R. Line Shape and the Water Vapor Continuum. *Atmos. Res.* **1989**, *23*, 229–241. [CrossRef]
89. Tipping, R.; Ma, Q. Theory of the water vapor continuum and validations. *Atmos. Res.* **1995**, *36*, 69–94. [CrossRef]
90. Green, P.D.; Newman, S.M.; Beeby, R.J.; Murray, J.E.; Pickering, J.C.; Harries, J.E. Recent advances in measurement of the water vapour continuum in the far-infrared spectral region. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.* **2012**, *370*, 2637–2655. [CrossRef] [PubMed]
91. Mlawer, E.; Payne, V.; Moncet, J.L.; Delamere, J.; Alvarado, M.; Tobin, D. Development and recent evaluation of the MT-CKD model of continuum absorption. *Philos. Trans. R. Soc. Lond. Ser. A* **2012**, *370*, 2520–2556. [CrossRef] [PubMed]
92. Shine, K.; Ptashnik, I.; Rädel, G. The Water Vapour Continuum: Brief History and Recent Developments. *Surv. Geophys.* **2012**, *33*, 535–555. [CrossRef]
93. Shine, K.; Campargue, A.; Mondelain, D.; McPheat, R.; Ptashnik, I.; Weidmann, D. The water vapour continuum in near-infrared windows—Current understanding and prospects for its inclusion in spectroscopic databases. *J. Mol. Spectrosc.* **2016**, *327*, 193–208. [CrossRef]
94. Schreier, F.; Gimeno García, S.; Vasquez, M.; Xu, J. Algorithmic vs. finite difference Jacobians for infrared atmospheric radiative transfer. *J. Quant. Spectrosc. Radiat. Transf.* **2015**, *164*, 147–160. [CrossRef]
95. Tashkun, S.A.; Perevalov, V.I.; Teffo, J.L.; Bykov, A.D.; Lavrentieva, N.N. CDSD-1000, the high-temperature carbon dioxide spectroscopic databank. *J. Quant. Spectrosc. Radiat. Transf.* **2003**, *82*, 165–196. [CrossRef]
96. Nikitin, A.; Lyulin, O.; Mikhailenko, S.; Perevalov, V.; Filippov, N.; Grigoriev, I.; Morino, I.; Yoshida, Y.; Matsunaga, T. GOSAT-2014 methane spectral line list. *J. Quant. Spectrosc. Radiat. Transf.* **2015**, *154*, 63–71. [CrossRef]
97. Tashkun, S.; Perevalov, V.; Lavrentieva, N. NOSD-1000, the high-temperature nitrous oxide spectroscopic databank. *J. Quant. Spectrosc. Radiat. Transf.* **2016**, *177*, 43–48. [CrossRef]
98. Brown, L.; Gunson, M.; Toth, R.; Irion, F.; Rinsland, C.; Goldman, A. 1995 Atmospheric Trace Molecule Spectroscopy (ATMOS) Linelist. *Appl. Opt.* **1996**, *35*, 2828–2848. [CrossRef]
99. Stamnes, K.; Tsay, S.C.; Wiscombe, W.; Jayaweera, K. Numerically Stable Algorithm for Discrete–Ordinate–Method Radiative Transfer in Multiple Scattering and Emitting Layered Media. *Appl. Opt.* **1988**, *27*, 2502–2509. [CrossRef] [PubMed]
100. Mayer, B.; Kylling, A. Technical note: The libRadtran software package for radiative transfer calculations—Description and examples of use. *Atmos. Chem. Phys.* **2005**, *5*, 1855–1877. [CrossRef]

101. Emde, C.; Buras-Schnell, R.; Kylling, A.; Mayer, B.; Gasteiger, J.; Hamann, U.; Kylling, J.; Richter, B.; Pause, C.; Dowling, T.; et al. The libRadtran software package for radiative transfer calculations (version 2.0.1). *Geosci. Model Dev.* **2016**, *9*, 1647–1672. [CrossRef]
102. Schreier, F. The Voigt and complex error function: Humlíček's rational approximation generalized. *Mon. Not. R. Astron. Soc.* **2018**, *479*, 3068–3075. [CrossRef]
103. Rothman, L.; Gamache, R.; Goldman, A.; Brown, L.; Toth, R.; Pickett, H.; Poynter, P.; Flaud, J.M.; Camy-Peyret, C.; Barbe, A.; et al. The HITRAN database: 1986 edition. *Appl. Opt.* **1987**, *26*, 4058. [CrossRef]