Py4CAtS

PYthon for Computational ATmospheric Spectroscopy (line-by-line)

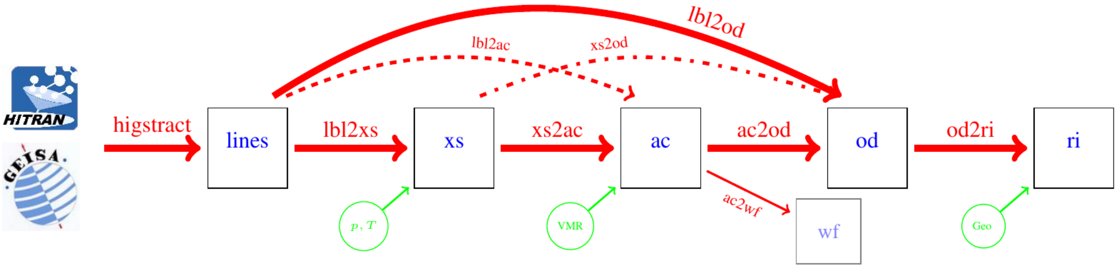
Franz Schreier

DLR — Remote Sensing Technology Institute
Oberpfaffenhofen, 82234 Wessling, Germany

April 4, 2019

Abstract

Py4CAtS — PYthon scripts for Computational ATmospheric Spectroscopy is a Python re-implementation of the Fortran infrared radiative transfer code GARLIC, where compute-intensive code sections utilize 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 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 two ways, from the Unix/Linux (or Windows/Mac) console/terminal or inside the (i)python interpreter. The basic design of the package, numerical and computational aspects relevant for optimization, and a sketch of the typical workflow are presented.



Contents

List of Figures

1 Introduction

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 high resolution radiative transfer code. Furthermore, as the retrieval of atmospheric parameters is in general a nonlinear optimization problem (inverse problem), the retrieval code has to be closely connected to the radiative transfer code (forward model).

Although a variety of general purpose high resolution radiative transfer models has been developed in the past decades, nb. FASCODE [Clough et al., 1988] and GENLN2 [Edwards, 1988], a new code has been found to be desirable because implementation of these sophisticated line-by-line (lbl) programs in retrieval algorithms is generally a non–trivial task. Furthermore derivatives with respect to the unknown profiles are often not available or at least difficult to access (more recent developments such as KOPRA Stiller et al. [2002] or ARTS Buehler et al. [2005] providing analytical derivatives were not available then).

Given the variety of applications at DLR–IMF a new code has been designed for arbitrary observation geometry and instrumental field-of-view (FoV) and instrumental line shape (ILS) (a.k.a. spectral response function, SRF). The original implementation MIRART(Modular InfraRed Atmospheric Radiative Transfer), written in Fortran 77, has been developed with emphasis on efficient and reliable numerical algorithms and a modular approach appropriate for simulation and/or retrieval. More recently this has been translated to modern Fortran 90/2003 as GARLIC (Generic Atmospheric Radiation Line-by-line Infrared Code) [Schreier et al., 2014].

Concurrently a version of MIRART/GARLIC written in Python Langtangen [2004] has been developed — named Py4CAtS for "*Py*thon for *C*omputational *At*mospheric *S*pectroscopy". Although highly optimized codes written in compiled languages such as Fortran or C/C++ are indispensable for operational processing, radiative transfer tools developed in script/interpreter languages such as Python, IDL/GDL, or MatLab/Octave/SciLab are an interesting alternative. Despite the reduced execution speed, script based tools are attractive because they allow for "rapid prototyping", can be executed on a large variety of platforms, and provide easy access to intermediate quantities, hence facilitating visualization and better understanding of the physics Lin [2012].

2 Physical Basics of Infrared Radiative Transfer

2.1 Schwarzschild Equation and Beer's Law

In the infrared and microwave spectral range the intensity (radiance) I at wavenumber ν received by an instrument at $s = 0$ can be described by the integral form of the equation of radiative transfer (neglecting scattering and assuming local thermodynamic equilibrium) [Liou, 1980, Goody and Yung, 1989, Zdunkowski et al., 2007]

$$I(\nu) = I_\mathrm{b}(\nu)\mathcal{T}(\nu;\infty) - \int_0^{s_\mathrm{b}} \mathrm{d}s' \; B(\nu, T(s')) \, \frac{\partial \mathcal{T}(\nu;s')}{\partial s'} \tag{1a}$$

$$= I_\mathrm{b}(\nu)\mathcal{T}(\nu;\infty) + \int_0^{\tau_\mathrm{b}} \mathrm{d}\tau \; B(\nu, T(\tau)) \exp(-\tau) \tag{1b}$$

where I_b is a background contribution (e.g., solar radiation at the top of the atmosphere in case of uplooking or limbviewing geometry, or surface emission in case of nadir viewing geometry) and B is the Planck function at temperature T,

$$B(\nu, T) \;=\; 2hc^2\nu^3 \Big/ \Big(e^{hc\nu/k_BT} - 1\Big) \;, \tag{2}$$

with c, h, k_B denoting speed of light, Planck constant, and Boltzmann constant, respectively. The partial derivative in Eq. (1a) is called the *weighting function*[1], see Eq. (38). Note that despite the minus sign the second term describing the atmospheric thermal emission is a radiation source, i.e. a positive contribution.[2]

The monochromatic transmission $\mathcal{T}$ (relative to the observer) is given according to Beer's law by

$$\begin{aligned} \mathcal{T}(\nu; s) &= \mathrm{e}^{-\tau(\nu;s)} && (3) \\ &= \exp\left[-\int_0^s \alpha(\nu, s')\, \mathrm{d}s'\right] \;, && (4) \\ \alpha(\nu; s) &= \sum_m k_m(\nu, s)\; n_m(s) + \alpha^{(\mathrm{c})}(\nu, s) && (5) \end{aligned}$$

where τ is the optical depth, α is the volume absorption coefficient, k_m and n_m are the absorption cross section and density of molecule m, and $\alpha^{(\mathrm{c})}$ the continuum absorption coefficient. Note that the absorption cross section is a function of (altitude dependent) pressure and temperature, but for brevity the condensed notation $k(\nu, z) = k\big(\nu, p(z), T(z)\big)$ has been used. In (1) we have assumed an uplooking or limb viewing path geometry, but (1) is easily rewritten to other slant path geometries. It should also be noted that the instrumental influence on the measured spectrum has been neglected.

2.2 Molecular Absorption

In general the molecular cross section is obtained by summing over the contributions from many lines,

$$k_m(\nu, z) \;=\; \sum_l S_l^{(m)}(T(z))\; g(\nu;\, \hat{\nu}_l^{(m)}, \gamma_l^{(m)}(p(z), T(z))) \;. \tag{6}$$

In the infrared and microwave spectral range molecular absorption is due to radiative transitions between rotational and ro–vibrational states of the molecules. A single spectral line is characterized by its position $\hat{\nu}$, line strength S, and line width γ, where the transition wavenumber (or frequency) is determined by the energies E_i, E_f of the initial and final state, $|i\rangle$, $|f\rangle$,

$$\hat{\nu} \;=\; \frac{1}{hc}\,(E_\mathrm{f} - E_\mathrm{i}) \tag{7}$$

For an individual line the cross section is the product of the temperature dependent line strength $S(T)$ and a normalized line shape function $g(\nu)$ describing the broadening mechanism, $k(\nu, z) = S(T(z)) \cdot g\big(\nu, p(z), T(z)\big)$. In the atmosphere the combined effect of pressure broadening (corresponding to a Lorentzian line shape) and Doppler broadening (corresponding to a Gaussian line shape) can be represented by a Voigt line profile.

---

[1] The term *weighting function* is frequently "misused" for the Jacobian, i.e. the partial derivatives of the radiance (or transmission in case of absorption spectroscopy) with respect to the unknows to be retrieved (the so-called "state vector")

[2] In fact this might be confusing, however, the minus in Eq. (1a) "compensates" the minus in (4). Moreover, the sign of the second term also "somehow" is depending on the path intergration variable: in case of a downlooking spaceborne observer one could use altitude z instead of distance $s = z_\mathrm{obs} - z$ to the observer.

2.2.1 Line strength and partition functions

The monochromatic absorption cross section for a single line is defined as the product of the line strength S and a normalized line profile function g essentially determined by line broadening,

$$k(\nu;\hat{\nu},S,\gamma) \;=\; S \,\cdot\, g(\nu;\hat{\nu},\gamma) \qquad \text{with} \quad \int\limits_{-\infty}^{+\infty} g \,\mathrm{d}\nu \;=\; 1 \;. \tag{8}$$

For electric dipole transitions the line strength is determined by the square of the temperature dependent matrix element of the electric dipole moment and by further factors accounting for the partition function, Boltzmann-distribution, and stimulated emission,

$$S(T) \;=\; \frac{8\pi^3}{3hc}\,\frac{g_i I_a}{Q(T)}\;\hat{\nu}\;e^{-E_i/kT}\left[1-e^{-hc\hat{\nu}/kT}\right]\;R_{if}\cdot 10^{-36} \tag{9}$$

here g_i is the degeneracy of the nuclear spin of the lower energy state, I_a is the relative abundance of the isotope[3], $Q(T)$ is the total partition sum, R_{if} is the transition probability given by the matrix element of the electric dipole operator $R_{if} = |\langle f|\boldsymbol{D}|i\rangle|^2$. A similar expression is found for the line strength of magnetic quadrupole transitions. In both cases the ratio of line strength at two different temperatures is given by

$$S(T) \;=\; S(T_0)\;\times\;\frac{Q(T_0)}{Q(T)}\;\frac{\exp\left(-E_i/kT\right)}{\exp\left(-E_i/kT_0\right)}\;\frac{1\;-\;\exp\left(-hc\hat{\nu}/kT\right)}{1\;-\;\exp\left(-hc\hat{\nu}/kT_0\right)}\;. \tag{10}$$

$Q(T)$ is the product of the rotational and vibrational partition functions, $Q = Q_{\mathrm{rot}}\cdot Q_{\mathrm{vib}}$, whose temperature dependance are calculated from

$$Q_{\mathrm{rot}}(T) \;=\; Q_{\mathrm{rot}}(T_0)\left(\frac{T}{T_0}\right)^{\beta}\;, \tag{11}$$

$$Q_{\mathrm{vib}}(T) \;=\; \prod_{i=1}^{N}\left[1-\exp(-hc\omega_i/kT)\right]^{-d_i}\;, \tag{12}$$

where β is the temperature coefficient of the rotational partition function, and N is the number of vibrational modes with wavenumbers ω_i and degeneracies d_i. Data required to calculate the vibrational partition sums have been taken from Norton and Rinsland [1991].

2.2.2 Pressure (collision) broadening — Lorentz profile

In case of pure pressure broadening the cross section for a single radiative transition is essentially given by a Lorentzian line profile

$$g_{\mathrm{L}}(\nu) \;=\; \frac{\gamma_{\mathrm{L}}/\pi}{(\nu-\hat{\nu})^2+\gamma_{\mathrm{L}}^2}\;. \tag{13}$$

The Lorentz half width (at half maximum, HWHM) γ_{L} is proportional to pressure p and decreases with increasing temperature. In case of a gas mixture with total pressure p and partial pressure p_s of the absorber molecule the total width is given by the sum of a self broadening contribution due to collisions between the absorber molecules and an air-broadening contribution due to collisions with other molecules,

$$\gamma_{\mathrm{L}}(p,p_s,T) \;=\; \left(\gamma_{\mathrm{L}}^{(0,\mathrm{air})}\,\frac{p-p_{\mathrm{s}}}{p_0}\;+\;\gamma_{\mathrm{L}}^{(0,\mathrm{self})}\,\frac{p_{\mathrm{s}}}{p_0}\right)\;\times\;\left(\frac{T_0}{T}\right)^{n}\;. \tag{14}$$

---

[3] In the HITRAN– and GEISA databases the abundances of the Earth atmosphere are used.

The exponent n specifying the dependence of temperature is so far known for only a few transitions of the most important molecules. The kinetic theory of gases (collision of hard spheres) yields the classical value $n = \frac{1}{2}$. The self-broadening coefficient $\gamma_\mathrm{L}^{(\mathrm{self})}$ is so far known for only a few transitions and will otherwise be set to the air-broadening coefficient $\gamma_\mathrm{L}^{(\mathrm{air})}$ (mostly specified for N_2 and/or O_2), i.e.

$$\gamma_\mathrm{L}(p,T) \;=\; \gamma_\mathrm{L}^{(\mathrm{air})}\,\frac{p}{p_0} \;\times\; \left(\frac{T_0}{T}\right)^n \tag{15}$$

Typical values of air-broadening coefficients are $\gamma_\mathrm{L} \approx 0.1p\ [\mathrm{cm}^{-1}/\mathrm{atm}]$ (see Tab. 2 in Rothman et al. [1987]).

Van Vleck-Weisskopf and van Vleck-Huber lineshapes Two variants of the Lorentz line profile widely used in the microwave regime are

$$g_\mathrm{vvw}(\nu) \;=\; \left(\frac{\nu}{\hat\nu}\right)^2 \left(\frac{\gamma_\mathrm{L}/\pi}{(\nu-\hat\nu)^2+\gamma_\mathrm{L}^2} \;+\; \frac{\gamma_\mathrm{L}/\pi}{(\nu+\hat\nu)^2+\gamma_\mathrm{L}^2}\right) , \tag{16}$$

and

$$g_\mathrm{vvh}(\nu) \;=\; \left(\frac{\nu}{\hat\nu}\right) \left(\frac{\nu\ \tanh\frac{hc\nu}{2kT}}{\hat\nu\ \tanh\frac{hc\hat\nu}{2kT}}\right) \left(\frac{\gamma_\mathrm{L}/\pi}{(\nu-\hat\nu)^2+\gamma_\mathrm{L}^2} \;+\; \frac{\gamma_\mathrm{L}/\pi}{(\nu+\hat\nu)^2+\gamma_\mathrm{L}^2}\right) . \tag{17}$$

Note that the van Vleck-Huber profile is the default line shape used in FASCODE [Clough et al., 1988] and its successor LBLRTM [Clough et al., 2005]. Furthermore, $\tanh(x) \approx x$ for small arguments x.

2.2.3 Doppler broadening

The thermal motion of the molecules leads to Doppler broadening of the spectral lines, which is described by a Gaussian line shape

$$g_\mathrm{D}(\nu) \;=\; \frac{1}{\gamma_\mathrm{D}}\left(\frac{\ln 2}{\pi}\right)^{1/2} \cdot \exp\left[-\ln 2\left(\frac{\nu-\hat\nu}{\gamma_\mathrm{D}}\right)^2\right] . \tag{18}$$

The half width (HWHM) is essentially determined by the line position $\hat\nu$, the temperature T, and the molecular mass m,

$$\gamma_\mathrm{D} \;=\; \hat\nu\,\sqrt{\frac{2\ln 2\ kT}{mc^2}} . \tag{19}$$

For a typical atmospheric molecule one finds

$$\gamma_\mathrm{D} \approx 6\cdot 10^{-8}\,\hat\nu\sqrt{T\,[K]} \qquad \text{for} \quad m \approx 36\,\mathrm{amu}.$$

2.2.4 Combined pressure and Doppler broadening

The combined effects of both broadening mechanisms can be modelled by convolution, i.e., a Voigt line profile

$$\begin{aligned} g_\mathrm{V}(\nu-\hat\nu,\gamma_\mathrm{L},\gamma_\mathrm{D}) \equiv&\ g_\mathrm{L}\otimes g_\mathrm{D} \\ =&\ \int_{-\infty}^{\infty} \mathrm{d}\nu'\ g_\mathrm{L}(\nu-\nu';\hat\nu,\gamma_\mathrm{L}) \,\times\, g_\mathrm{D}(\nu'-\hat\nu;\hat\nu,\gamma_\mathrm{D}) . \end{aligned} \tag{20}$$

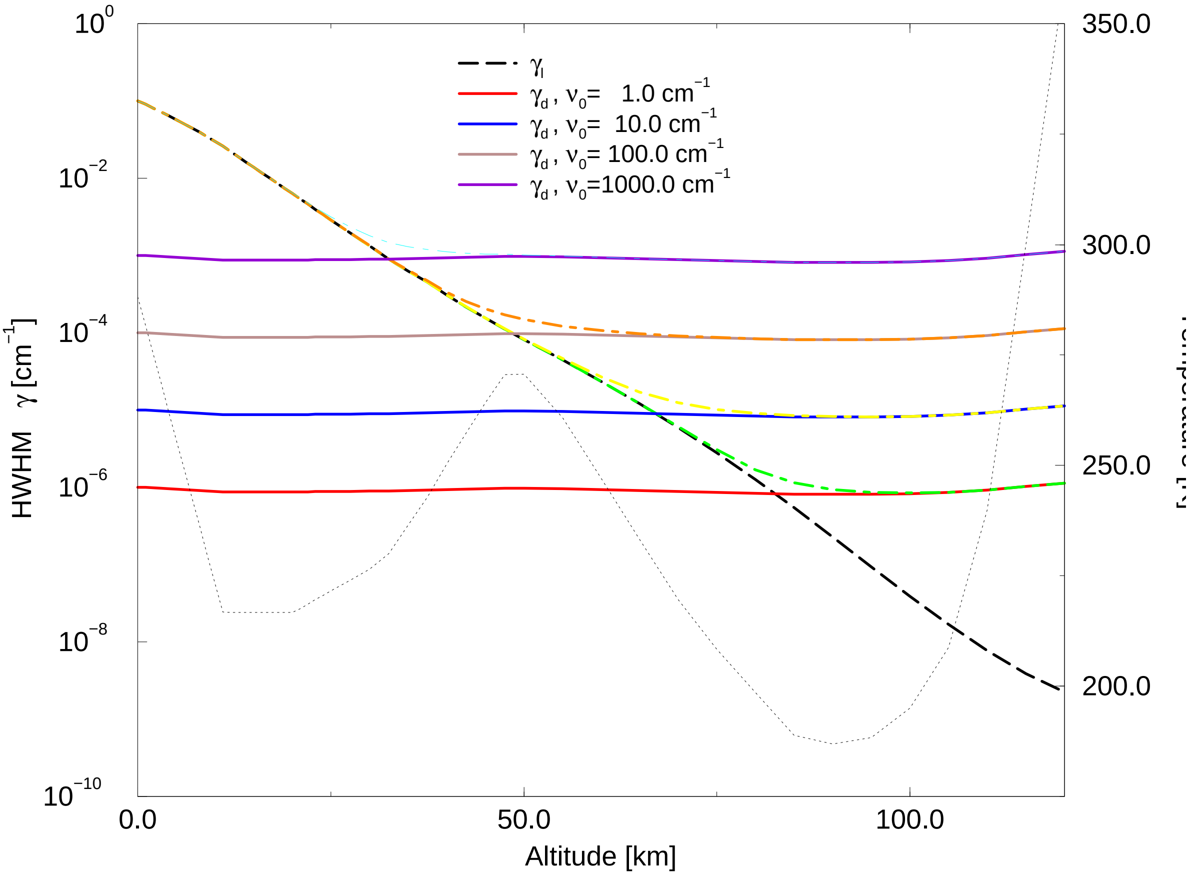


Figure 1: Half widths (HWHM) for Lorentz-, Doppler- and Voigt-Profile as a function of altitude for a variety of line positions $\hat{\nu}$. The Lorentz width is essentially proportional to pressure and hence decays approximately exponentially with altitude. In contrast the Doppler width is only weakly altitude dependent. In the troposphere lines are generally pressure broadened, the transition to the Doppler regime depends on the spectral region. The dotted line indicated atmospheric temperature. (Pressure and temperature: US Standard atmosphere, molecular mass $36 amu$)

Several empirical approximations for the half width (HWHM) of a Voigt line (defined by $g_V(\hat{\nu} \pm \gamma_V) = \frac{1}{2} g_V(\hat{\nu})$) have been developed [Olivero and Longbothum, 1977]. For the approximation

$$\gamma_V = \frac{1}{2} \left(c_1 \gamma_\mathrm{L} + \sqrt{c_2 \gamma_\mathrm{L}^2 + 4\gamma_D^2} \right) \qquad \text{with} \quad c_1 = 1.0692, \; c_2 = 0.86639 \tag{21}$$

a accuracy of 0.02% has been specified, with $c_1 = c_2 = 1$ the accuracy is in the order of one percent. A comparison of Lorentzian, Doppler, and Voigt half width is given in Fig. 1.

3 Algorithms

NOTE: For a more thorough and up-to-date discussion of algorithmic aspects see the GARLIC paper [Schreier et al., 2014].

3.1 Numerical Aspects — Computational Challenges

The computational challenge of high resolution atmospheric radiative transfer modelling is due to several facts. The summation in Eq. (6) has to include all relevant lines contributing to the spectral interval considered. In many line–by–line codes a cutoff wavenumber of $25\,\mathrm{cm}^{-1}$ from line center is frequently employed for truncation of line wings. Note that the widely used HITRAN and GEISA spectroscopic databases [Rothman et al., 2013, Jacquinet-Husson et al., 2008] list more than some million lines of about 40 molecules in the microwave, infrared, to ultraviolet regime, whereas the JPL spectral line catalog [Pickett et al., 1998] covering the submillimeter, millimeter, and microwave only has almost 2 million entries.

Furthermore the wavenumber grid has to be set in accordance with the line widths γ, i.e. the grid spacing is typically chosen in the order of $\delta\nu \approx \gamma/4$. Typical line widths due to pressure broadening are in the order of $\gamma(p) \approx (p/p_0)\,0.1\,\mathrm{cm}^{-1}$ with $p_0 = 1013\,\mathrm{mb}$. In the atmosphere the pressure decays approximately exponentially with altitude z, and the line width decreases accordingly until Doppler broadening (proportional to line position and the square root of the temperature over molecular mass ratio) becomes dominant (cf. Fig. 1). Hence, for an altitude of $z = 100\,\mathrm{km}$ with a pressure $p \approx 10^{-4}\,\mathrm{mb}$ the number of spectral grid points required for a spectral interval $\Delta\nu = 1\,\mathrm{cm}^{-1}$ in the microwave is in the order of $1/(0.1 \times 10^3/10^{-4}) = 10^6$. For a spectral interval of width $\Delta\nu = 10\,\mathrm{cm}^{-1}$ in the region of the CO_2 ν_2 band around $500\,\mathrm{cm}^{-1}$ the number of spectral grid points is in the order of 10^5.

A variety of approaches has been developed to speed–up the calculation and an essential difference between different line–by–line codes is the choice of the line profile approximation, wavenumber grid, and interpolation. Some of the algorithms are specifically designed for the individual functions to be calculated, e.g., the Clough and Kneizys [1979] algorithm used in FASCODE [Clough et al., 1988]: The Lorentzian (or Voigt function) is decomposed using three or four even quartic functions, each of them is then calculated on its individual grid. (A similar technique using quadratic functions has been developed by Uchiyama [1992].) GENLN2 [Edwards, 1988] performs the line–by–line calculation in two stages, i.e., the entire spectral interval of interest is first split in a sequence of “wide meshes”; contributions of lines with their center in the current wide mesh interval are computed on a fine mesh, and the contribution of other lines is computed on the wide mesh. Fomin [1995] defines a series of grids and evaluates line wing segments of larger distance to the line center on increasingly coarse grids. Sparks [1997] also uses a series of grids with $2^k + 1$ grid points ($k = 1, 2, \ldots,$ where the coarsest grid with 3 points spans the entire region) and uses a function decomposition similar to ours.

3.2 Voigt profile and Voigt function

The convolution of a Lorentz and a Gauss profile, commonly known as the Voigt profile, is important in many branches of physics, nb. atomic and molecular spectroscopy, atmospheric radiative transfer [Armstrong, 1967].

It is convenient to define the Voigt function $K(x,y)$ normalized to $\sqrt{\pi}$,

$$K(x,y) \;=\; \frac{y}{\pi} \int_{-\infty}^{\infty} \frac{\mathrm{e}^{-t^2}}{(x-t)^2 + y^2}\,\mathrm{d}t \;, \tag{22}$$

where the dimensionless variables x, y are defined in terms of the distance from the center position, $\nu - \hat{\nu}_0$, and the Lorentzian and Gaussian half–widths γ_L, γ_G:

$$x \;=\; \sqrt{\ln 2}\,\frac{\nu - \hat{\nu}}{\gamma_\mathrm{G}} \qquad \text{and} \qquad y \;=\; \sqrt{\ln 2}\,\frac{\gamma_\mathrm{L}}{\gamma_\mathrm{G}}\;. \tag{23}$$

The Voigt function represents the real part of the complex function

$$W(z) \;\equiv\; K(x,y) \,+\, \mathrm{i}L(x,y) \;=\; \frac{\mathrm{i}}{\pi} \int_{-\infty}^{\infty} \frac{e^{-t^2}}{z-t}\,\mathrm{d}t \qquad \text{with} \qquad z = x + \mathrm{i}y, \tag{24}$$

that, for $y > 0$, is identical to the complex error function (probability function) defined by [Abramowitz and Stegun, 1964]

$$w(z) \;=\; \mathrm{e}^{-z^2} \left(1 \;+\; \frac{2\mathrm{i}}{\sqrt{\pi}} \int_0^z \mathrm{e}^{t^2}\,\mathrm{d}t\right) \;=\; \mathrm{e}^{-z^2} \Big(1 \;-\; \mathrm{erf}(-\mathrm{i}z)\Big)\;. \tag{25}$$

The complex error function satisfies the differential equation

$$w'(z) \;=\; -\,2z \cdot w(z) \;+\; \frac{2\mathrm{i}}{\sqrt{\pi}} \tag{26}$$

and the series and asymptotic expansions (where Γ is the gamma function)

$$w(z) = \sum_{n=0}^{\infty} \frac{(\mathrm{i}z)^n}{\Gamma\left(\frac{n}{2}+1\right)} \tag{27}$$

$$w(z) = \frac{\mathrm{i}}{\pi} \sum_{k=0}^{\infty} \frac{\Gamma\left(k+\frac{1}{2}\right)}{z^{2k+1}}\;. \tag{28}$$

Unfortunately, none of these functions can be evaluated in closed analytical form and a large number of numerical algorithms have been developed in the past [Schreier, 1992]. Most modern algorithms for the Voigt function employ approximations for the complex error function. Actually this approach has further advantages, in particular it simultaneously provides derivatives of these functions, required for, e.g., sensitivity analysis or optimization. Furthermore, the complex error function can be used if more sophisticated line profiles, e.g., the Rautian for collisional narrowing, or line mixing effects have to be computed.

Rational approximations are known to give accurate and efficient algorithms for a large class of functions, and also have been successfully used to approximate the complex error function, e.g., Hui et al. [1978], Humlíček [1979, 1982], Weideman [1994].

$$w(z) \;=\; \frac{P(\tilde{z})}{Q(\tilde{z})} \;=\; \frac{\sum\limits_{m=0}^{M} a_m \tilde{z}^m}{\sum\limits_{n=0}^{M+1} b_n \tilde{z}^n} \qquad \text{where} \quad \tilde{z} = y - \mathrm{i}x. \tag{29}$$

Because of the asymptotic behaviour of the complex error function $w \sim 1/z$), the degree of the nominator and denominator polynomials are constrained by $N = M + 1$. It should be noted, that for atmospheric spectroscopy applications the Lorentz to Gauss width ratio varies over many orders of magnitude, i.e., $10^{-7} < y < 10^4$ (see Figures 2 and 3 in [Schreier, 2011]). On the other hand, $K(x,y)$ and $w(z)$ are especially difficult to evaluate for small $y < 1$, and in most algorithms the x, y plane (or the first quadrant $x, y \geq 0$ because of the symmetry relations) is divided in several regions and appropriate methods are utilized, e.g., a series approximation for small x, y and an asymptotic approximation for large x, y.

The Hui et al. [1978] and Weideman [1994] rational approximations appear to be quite tempting as they provide a single approximation applicable to the entire x, y plane. However, the Hui et al. 1978 algorithm (with $M = 6$) has significant accuracy problems for small y

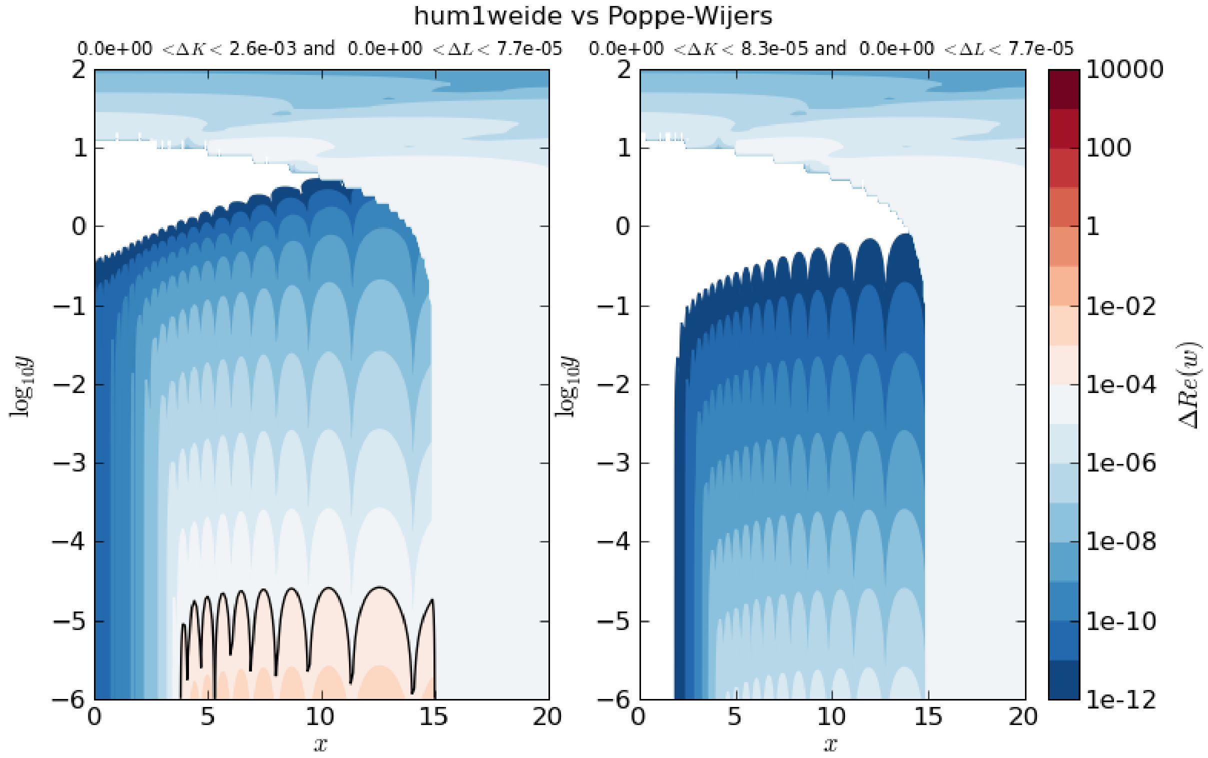


Figure 2: Relative accuracy of the combined Humliček I and Weideman approximation (left $N = 24$ and right $N = 32$).

and medium x, and the Weideman 1994 approximations requires a large number of terms to achieve sufficient accuracy for small y, making it computationally less efficient. The Humlíček [1982] code (or variations thereof, e.g., Kuntz [1997], Imai et al. [2010]) has been selected by several authors, but its performance depends on the compiler's efficiency to handle nested conditional branches.

In order to avoid complicated `if` constructs for the calculation of the complex error function, `lbl2xs` and `lbl2od` use an optimized combination of the Humlíček [1982] and Weideman 1994 algorithms [Schreier, 2011]

$$w(z) \;=\; \begin{cases} \frac{\mathrm{i}z/\sqrt{\pi}}{z^2-\frac{1}{2}} & |x|+y>15, \\ \frac{\pi^{-1/2}}{L-\mathrm{i}z} \;+\; \frac{2}{(L-\mathrm{i}z)^2}\sum_{n=0}^{N-1} a_{n+1}Z^n & \text{otherwise}, \end{cases} \tag{30}$$

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 $y < 10^{-5}$ and $4 < x < 15$; for $N = 32$ the relative error $|\Delta K|/K$ is less than $8 \cdot 10^{-5}$ for all x, y, cf. Fig. 2.

3.3 Multigrid algorithm

The problem is the efficient computation of a superposition of similar functions $f_l(x)$ over a large region of its independent variable x,

$$F(x_i) \;=\; \sum_{l=1}^{L} f_l(x_i) \qquad \text{for} \quad x_{\mathrm{lo}} \equiv x_0 < x_1 < \cdots < x_i < \cdots < x_n \equiv x_{\mathrm{hi}}. \tag{31}$$

Frequently the functions $f_l(x)$ vary rapidly only in a small region of the entire domain, but the evaluation of $F(x)$ covers a large x–interval where the individual f_l is mostly smooth. However, accurate modelling of the function sum requires appropriate sampling of the x–grid, i.e., the grid interval size δx has to be chosen small enough to resolve the details of $f_l(x)$ in the regions of strong variability. Thus, for an uniform/equidistant grid the spacing δx is determined by the fine structure of the f_l's.

Computing $f_l(x)$ on a uniform, appropriately dense grid over the entire region of interest is obviously not very efficient when f_l is smooth everywhere except for a small subinterval of $[x_{\mathrm{lo}}, x_{\mathrm{hi}}]$. The calculation is significantly accelerated when f_l is decomposed into rapidly and slowly varying contributions, where the fast part has to be computed on a fine grid in the region of strong variability only and the smooth part is computed on a coarse grid covering the entire interval of interest. Furthermore, if the smooth part is a continuous function of x over the entire interval $[x_{\mathrm{lo}}, x_{\mathrm{hi}}]$, the sum in (31) can be performed separately for the rapidly and slowly varying contributions,

$$F^{\mathrm{fast}}(x) \;=\; \sum_l f_l^{\mathrm{fast}}(x) \qquad \text{where} \quad x \in \{x_0, \ldots, x_n\} \qquad \text{(fine grid)} \tag{32}$$

$$F^{\mathrm{smooth}}(X) \;=\; \sum_l f_l^{\mathrm{smooth}}(X) \qquad \text{where} \quad X \in \{X_0, \ldots, X_N\} \qquad \text{(coarse grid)} \tag{33}$$

and the interpolation to the fine grid x is required only once after the entire sum has been evaluated,

$$F(x) \;=\; F^{\mathrm{fast}}(x) \;+\; \mathcal{I}\left[F^{\mathrm{smooth}}(X)\right](x)\ . \tag{34}$$

Here $\mathcal{I}$ denotes an interpolation operator, i.e., $\mathcal{I}\left[F^{\mathrm{smooth}}\right](x)$ is the interpolated sum of smooth contributions (available at the coarse grid X) at the fine grid point x. In order to guarantee an efficient interpolation, an equidistant set of N coarse grid points $X_0, X_1, \ldots, X_N$ with spacing ΔX satisfying $\Delta X/\delta x = n/N$ integer will be used (furthermore $X_0 = x_0$ and $X_N = x_n$). For convenience ratios of power two will be used, i.e., $n/N = 2^m$. Clearly, the larger the ratio, the larger the computational speed–up. However, for very large coarse grid spacings, errors due to inadequate sampling of the smooth contribution to f become too big. For our applications ratios $n/N = 4$ and $n/N = 8$ have turned out to provide a reasonable compromise between speed and accuracy.

Thus the problem of efficient calculation of the sum (31) has been transformed into the problem of splitting off the smooth part of each f_l, i.e.,

$$f_l(x) \;=\; f_l^{\mathrm{smooth}}(x) + f_l^{\mathrm{fast}}(x) \tag{35}$$

The simplest choice of the smooth function that automatically satisfies the constraints of continuity is to use the function f_l itself as smooth function f_l^{smooth}, too. Note that the sum of the smooth contributions F^{smooth} is interpolated to the fine grid and then added to the sum of the fine grid, quickly varying contributions. In order to compensate for the interpolated smooth contributions in the regions of strong variability, the quickly varying contribution is defined as

$$f_l^{\mathrm{fast}}(x) \;=\; f_l(x) - \mathcal{I}\left[f^{\mathrm{smooth}}\right](x) \qquad \text{for } x \text{ in center region.} \tag{36}$$

Note that f_l^{fast} or its first derivative may have discontinuities. In Fig. 3 this decomposition is shown for the Lorentzian line shape.

The speed–up that can be achieved with the two–grid algorithm developed in the previous subsection is essentially determined by the ratio of grid points on the fine and coarse

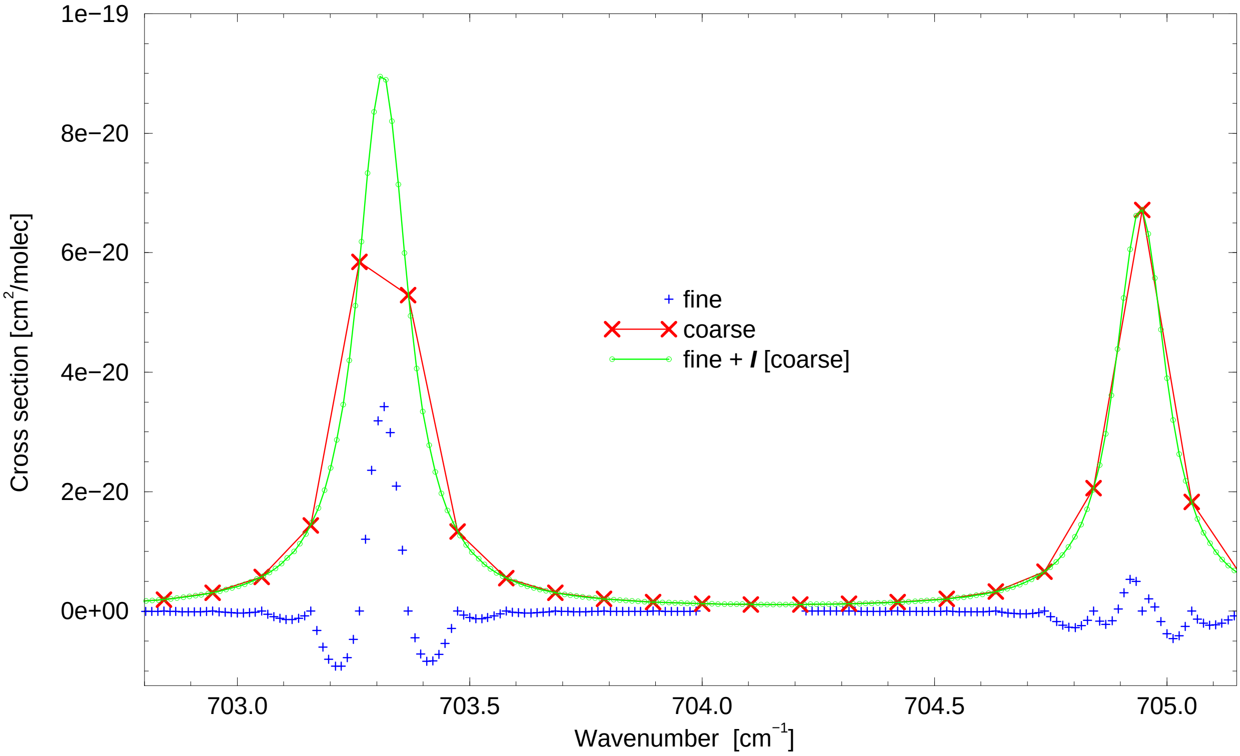


Figure 3: Lorentzian line profile function (13) decomposed in a slowly varying contribution (evaluated on a coarse grid) and a rapidly varying contribution evaluated on a fine grid near the center only (Two–point Lagrange interpolation). The example corresponds to CO_2 cross sections at 1013.25 mb and 296 K; for clarity only the strongest lines have been included.

grid, i.e. with $n/N = 4$ or $n/N = 8$ only a small computational gain is possible. A significant acceleration can be achieved by using further grids with increasing grid point spacing. However, for our applications to spectral modelling the computational overhead required to control a series of grids turned out to partly compensate the speed–up provided by very coarse grids, and simply using three grids turned out to be efficient [Schreier, 2006].

3.4 Path Quadrature

To evaluate the optical depth τ according to (3) it is necessary to compute the integral of the absorption coefficient $\alpha(z)$ along the (vertical or slanted) path (Py4CAtS considers a plane-parallel atmosphere, so `lbl2od`, `dod2ri`, … are restricted to up- and down-looking viewing geometries, and limb viewing is not supported.)

The trapezoidal rule is perhaps the most basic and important Newton-Cotes formula for numerical evaluation of an integral of a function $y(x)$,

$$\int_{x_0}^{x_n} y(x)\,\mathrm{d}x = \frac{1}{2}\sum_{i=1}^{n}(y_i + y_{i-1})(x_i - x_{i-1}) \; . \tag{37}$$

Clearly the assumption behind the trapezoid rule, i.e. a linear polynomial interpolating the function y in the subintervals $[x_{i-1}, x_i]$, is hardly justified in view of the exponential dependence of air number density or the presence of steep gradients of, e.g., the water concentration profile. Nevertheless, the trapezoid rule is the most robust and fastest quadrature rule, and it is the default rule used here.

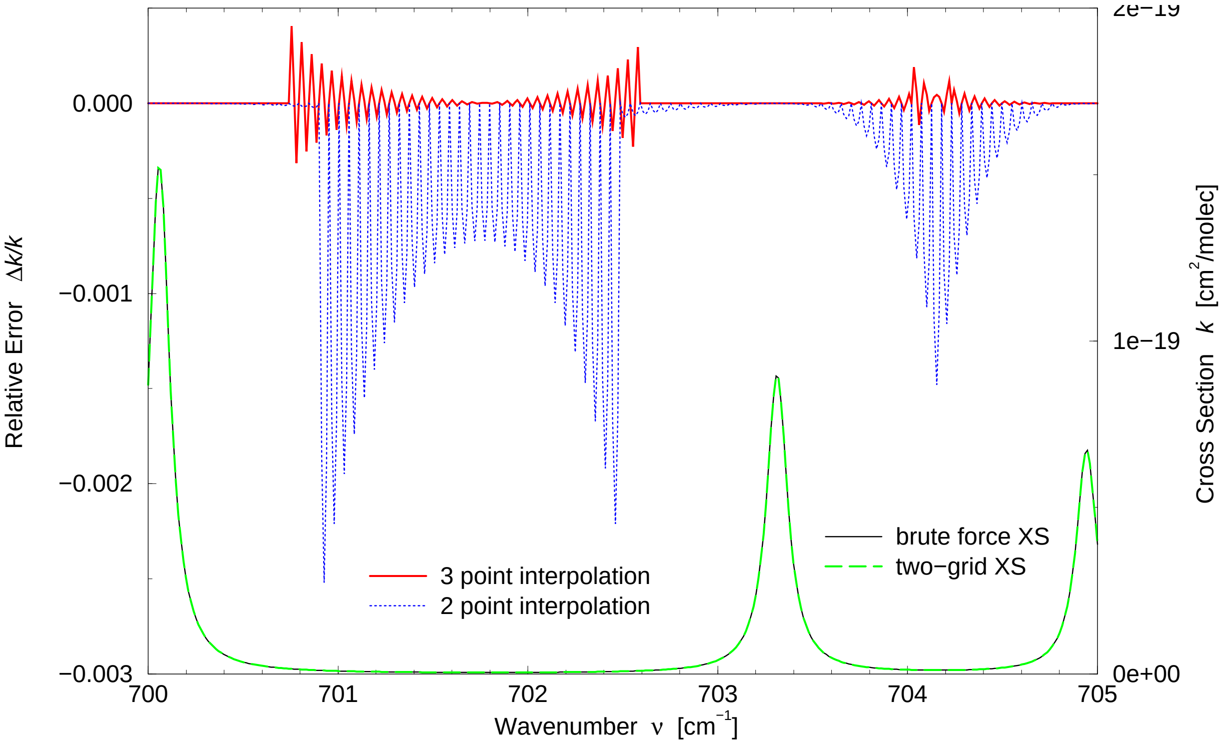


Figure 4: Error of the sum of some Lorentzians (13) evaluated with a two–grid approximation and linear and quadratic Lagrange interpolation. For two–point interpolation the fine grid was used within $\nu_l \pm 12\gamma$ around the line center. whereas for three–point interpolation the fine grid extension was $\nu_l \pm 10\gamma$. The sum of Lorentzians (cross section, "XS") evaluated with the "brute force" and with the two–grid–algorithm are indistinguishable. (Same example as in Fig. 3.)

As an alternative, `lbl2od` offers the possibility to use Simpson's rule (using `integrate.simps` from SciPy), the next higher Newton-Cotes formula assuming a quadratic interpolating polynomial. Note that especially for the difference or cumulative optical depth this can be very time-consuming.

For the evaluation of the Schwarzschild equation Py4CAtS uses optical depth τ as integration variable as in (1b), and the integral along the line-of-sight is approximated by a sum over all layers. The default quadrature scheme assumes that the Planck function seen as function of optical depth varies linearly with τ within a layer, i.e.

$$B(\tau) = \frac{\tau - \tau_l}{\tau_{l+1} - \tau_l} B(\tau_{l+1}) + \frac{\tau_{l+1} - \tau}{\tau_{l+1} - \tau_l} B(\tau_l)$$

with $\tau_l \le \tau < \tau_{l+1}$. As an alternative, a "B exponential in optical depth" approach can be used where the Planck function is approximated by $B(T(\tau)) = B(T(\tau_l))\,\mathrm{e}^{\beta(\tau-\tau_l)}$ with $\beta = \log\big(B(\tau_l)/B(\tau_{l+1})\big)/(\tau_{l+1} - \tau_l)$. See the GARLIC paper for more details.

4 Verification and Validation

Verification essentially is a check, if the (mathematical and/or physical) model of the real word or nature is implemented correctly, whereas the aim of validation is to demonstrate that the (correctly implemented code) models the real world correctly [Calder et al., 2004, Einarsson et al., 2005].

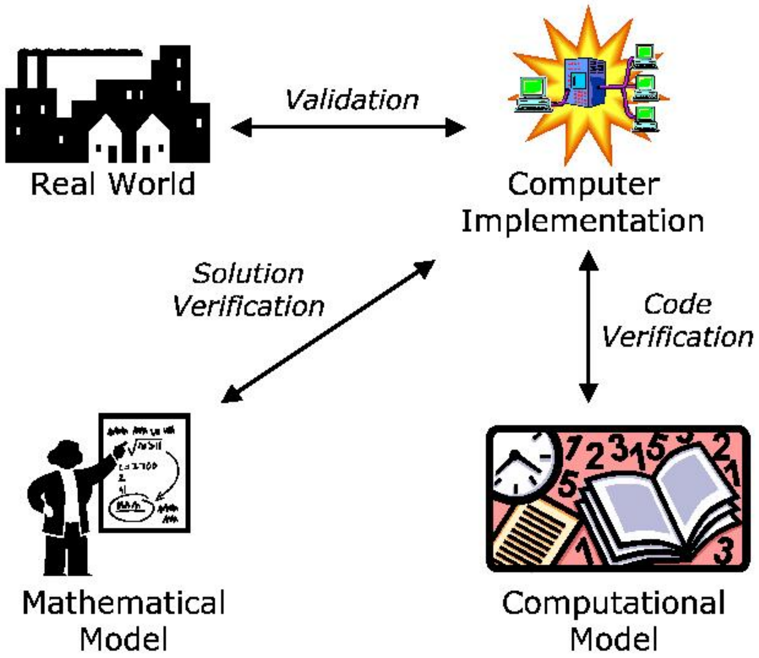


Figure 5: A model of computational validation (This is Figure 2.2 from Boisvert et al. [2005].)

The standard approach to verification of LbL codes relies on cross checks against similar codes. MIRART/GARLIC participated in three extensive intercomparisons. For a more extensive discussion, along with a comparison of various path intergration schemes and intercomparisons with real observed data (validation) see the recent GARLIC paper [Schreier et al., 2014].

4.1 AMIL2DA

In order to assess the consistency of level 2 data generated from measurements by the MIPAS Fourier transform limb emission spectrometer onboard the ENVISAT satellite, the AMIL2DA project aimed at careful comparison and characterization of algorithms and data analysis stategies used by different European groups. An essential step of this project was a cross comparison of the radiative transfer forward models to be used as part of the group's MIPAS data processing [von Clarmann et al., 2002]. The intercomparison was organized as a series of exercises, starting from simple settings proving basic functionalities and proceeding to more complex and realistic scenarios. Accordingly the first exercises considered the transmission of a single N_2O line for different pressures and temperatures, hence testing line shape computation and line strength conversion. In a second set of exercises radiance spectra for a limb viewing geometry with instrumental effects have been intercompared. Figure 6 shows a comparison of a limb emission spectrum, revealing deviations well below one percent.

4.2 IRTMW01

A major objective of IRTMW01 was the intercomparison of radiative transfer codes in the microwave spectral domain [Melsheimer et al., 2005]. Similar to the AMIL2DA intercomparison it was organized in a series of progressively more sophisticated “cases”, starting with an assessment of Voigt line shape and molecular absorption coefficient calculations. As for

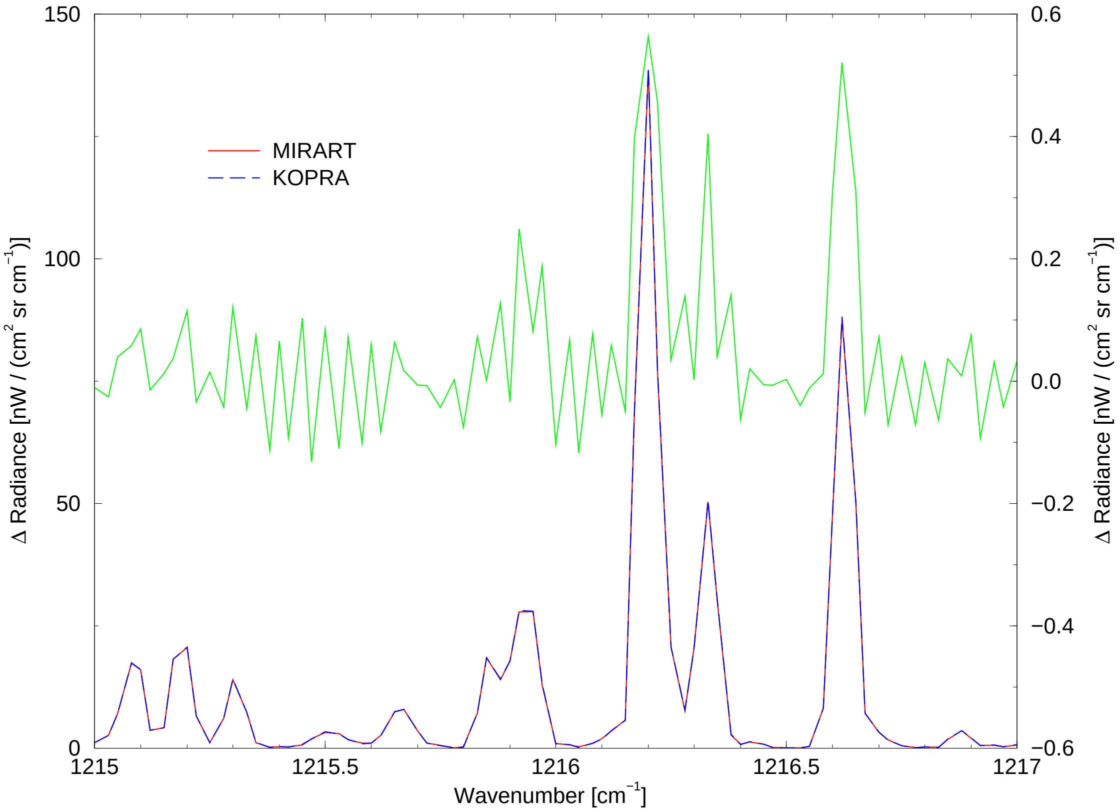


Figure 6: AMIL2DA Forward model intercomparison (Exercise 20):
KOPRA line–by–line code [Stiller et al., 2002] and MIRART.
Limb view with tangent altitude 40 km, apodized FTS instrument line shape, finite field–of–view, H_2O, CO_2, O_3, N_2O, and CH_4; CKD–continuum [Clough et al., 1988].

the corresponding AMIL2DA exercises MIRART exhibited slight deviations for spectra at temperatures different from the database reference temperature, that have been attributed to the use of different line strengths conversion approaches.

The purpose of case 3 was to check the correct implementation of the radiative transfer algorithm, nb., the solution of the integrals in Eqs. (1) and (3). In order to allow to discriminate different sources of possible deviations between the models, absorption coefficients $\alpha(\nu, z)$ have been pre–calculated by the University of Bremen group and used as common input. Case 4 was aiming to test the entire computational chain of the codes including LbL calculation, continuum corrections, and path quadrature. Geometries and instrument settings were identical to case 3, thus changes from case 3 spectra to case 4 spectra have to come from differences in the input data or from differences in the cross section and absorption coefficient calculations.

The intercomparison was performed for different geometries, and for ideal monochromatic spectra as well as ILS (instrument line shape) and FoV (field-of-view) convolved spectra. Figure 7 shows the results for the uplooking geometry: Whereas case 3 spectra do not yield visible differences, slight deviations show up in case 4 for small zenith angles. Similar results were also found for the case 3 and case 4 down looking and limb viewing exercises.

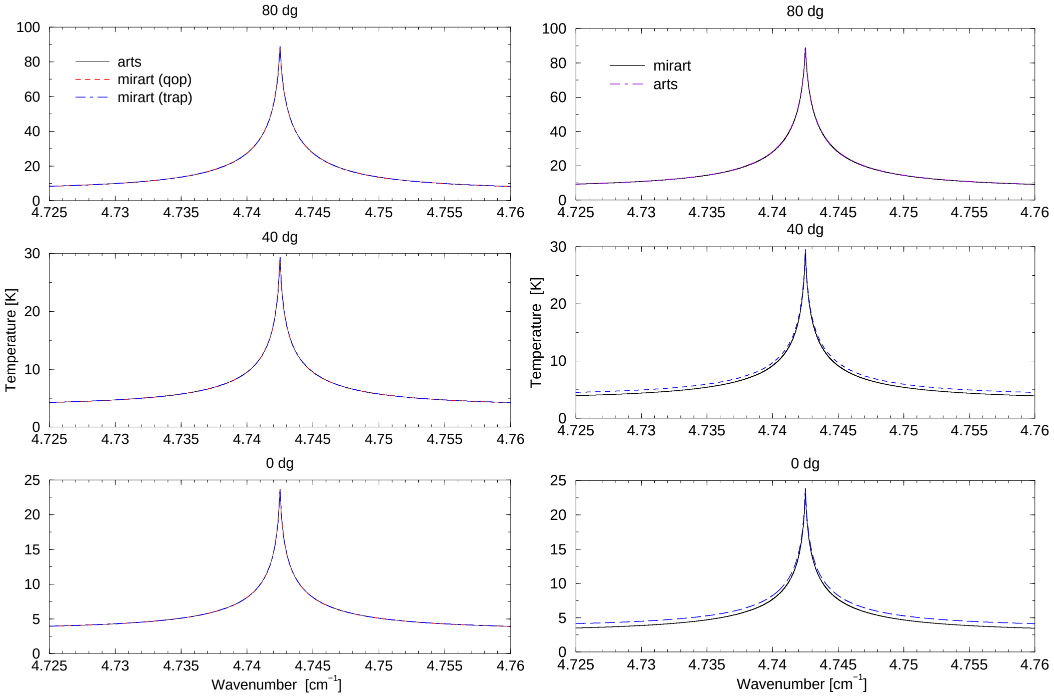


Figure 7: IRTMW01 intercomparison: case 3 (left) and 4 (right) up–looking:
ARTS line–by-line code (University of Bremen, Buehler et al. [2005]) vs. MIRART.
O_3 and O_2, perfect antenna (i.e. infinitesimal FoV), single side band receiver with Gaussian ILS function with half width 0.25 MHz.

4.3 ARTS – GARLIC – KOPRA

An intercomparison of three line-by-line (lbl) codes developed independently for atmospheric sounding — ARTS, GARLIC, and KOPRA — has been performed for a thermal infrared nadir sounding application assuming a HIRS-like (High resolution Infrared Radiation Sounder) setup. Radiances for the HIRS infrared channels and a set of 42 atmospheric profiles from the "Garand dataset" [Garand et al., 2001] have been computed. Except for a few channels and/or atmospheres, the codes generally agree quite well with deviations less than one Kelvin. Averaging over all atmospheres (Fig. **??**), discrepancies are smaller than half a Kelvin except for a few channels, mostly due to the choice of the continuum model used. Further results are presented in Schreier et al. [2018a].

4.4 ACE-FTS

Effective height transit spectra of Earth have been generated by combining representative limb transmission spectra [Hughes et al., 2014] observed by the ACE-FTS [Bernath et al., 2005, Bernath, 2017] solar occultation instrument. These spectra have been degraded to moderate and low resolution and compared with spectra computed with GARLIC using HITRAN (or GEISA) spectroscopic data. Inclusion or exclusion of molecules considered in the modeling allowed to study their impact on the transit spectra. The main infrared absorbers water, carbon dioxide, ozone, nitrous oxide, and methane can be clearly identified in the effective height spectra. Furthermore, nitric acid is very prominent around 900 cm^{-1}, and the main constituents of Earth's atmosphere, molecular oxygen and nitrogen, are also important for modeling the spectra. To further reduce the discrepancies, heavy molecules had to be considered, too. In particular, the "technosignatures" CFC11 and CFC12 are

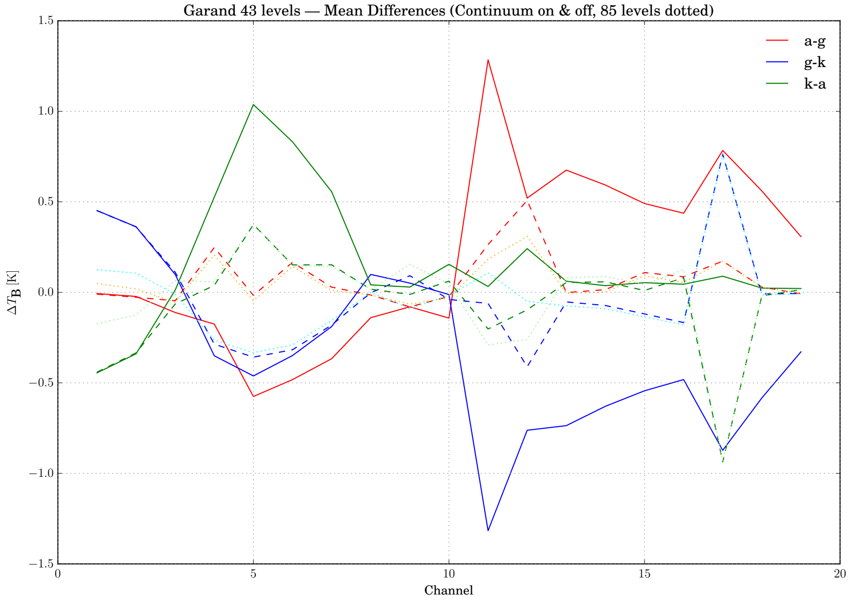


Figure 8: ARTS-GARLIC-KOPRA mean brightness temperature differences: Solid lines: 43 level atmosphere with continuum; dashed lines: without continuum; dotted lines: 85 level atmosphere without continuum.

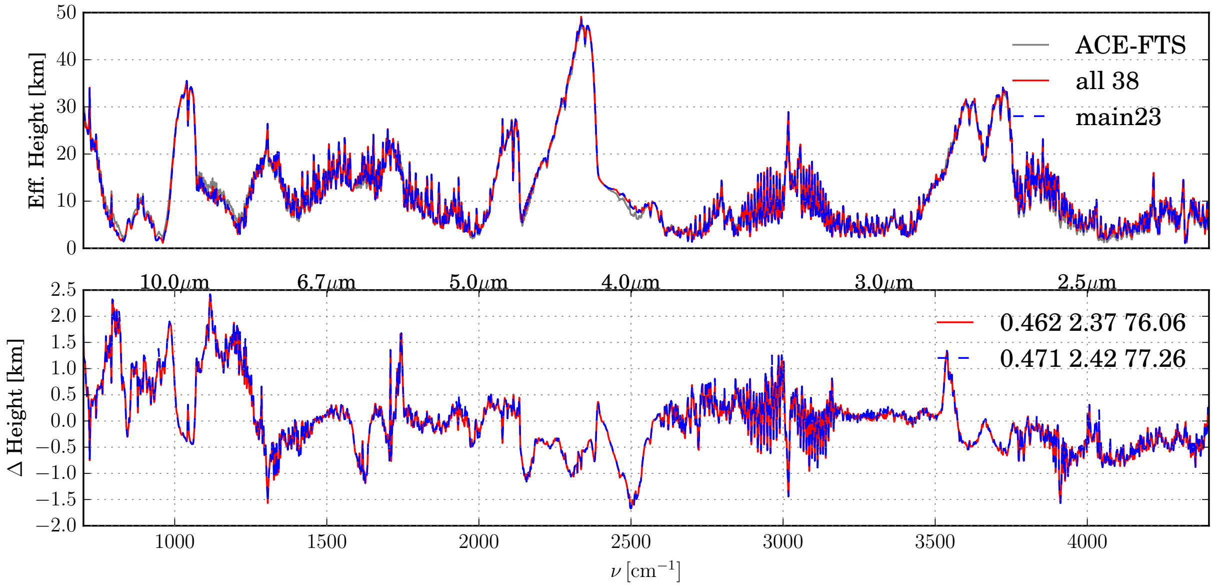


Figure 9: Comparison of GARLIC effective height spectra (for 38 and 23 molecules) with the "atlas" spectrum observed by the ACE-FTS instrument. The numbers in the legend are the mean, maximum, and norm residuum of observation vs. model. (Sub)-arctic winter, spectral response Gaussian with HWHM $\Gamma = 1\,\mathrm{cm}^{-1}$.

visible in the moderate and low resolution spectra, cf. Fig. 9. The best matching model has a mean residuum of 0.4 km and a maximum difference of 2 km to the measured effective height. For details see Schreier et al. [2018b].

4.5 GARLIC vs. Py4CAtS

Fig. 10 shows an intercomparison of GARLIC and Py4CAtS radiance spectra. The Py4CAtS modeling essentially comprises the `lbl2od` and `dod2ri` steps with an additional convolution step `radNadirGauss = radNadir.convolve(1.0,'gauss')`.

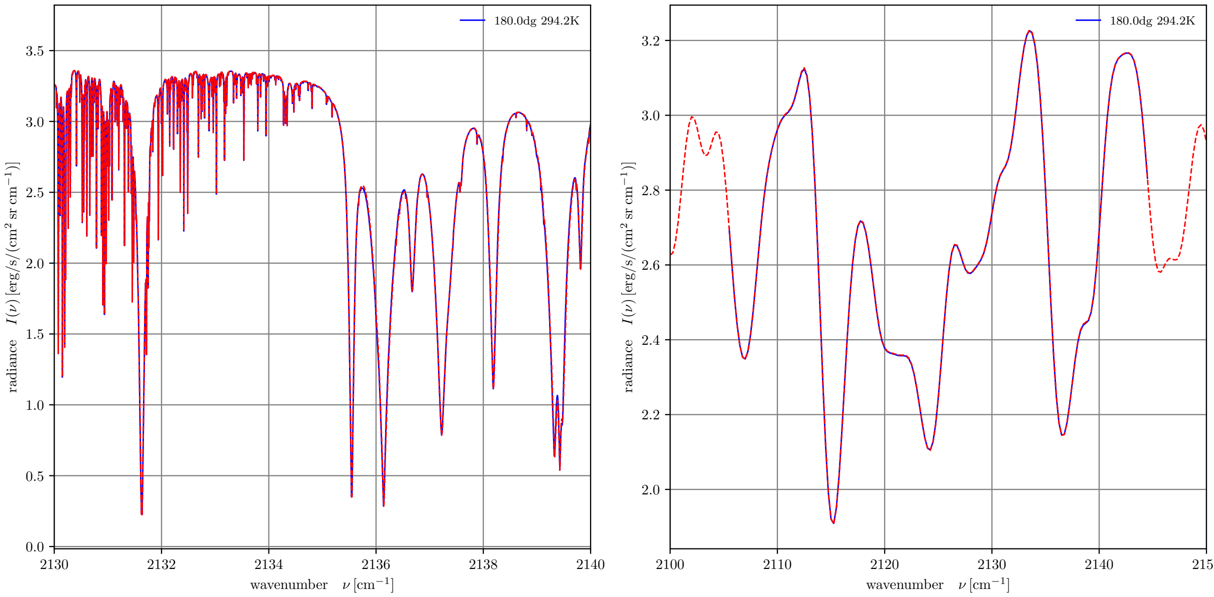


Figure 10: Intercomparison of radiance spectra with corresponding spectra generated with GARLIC. Midlatitude summer atmosphere, downlooking observer at ToA with viewing zenith angle 180°. Left: monochromatic spectra; Right: radiance convolved with a Gaussian response function with HWHM = $1\,\mathrm{cm}^{-1}$.

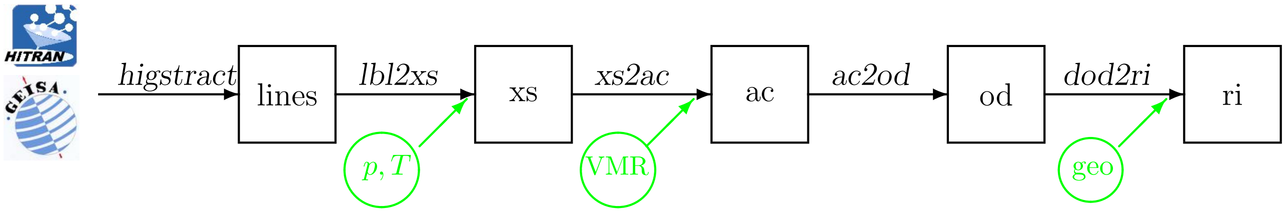
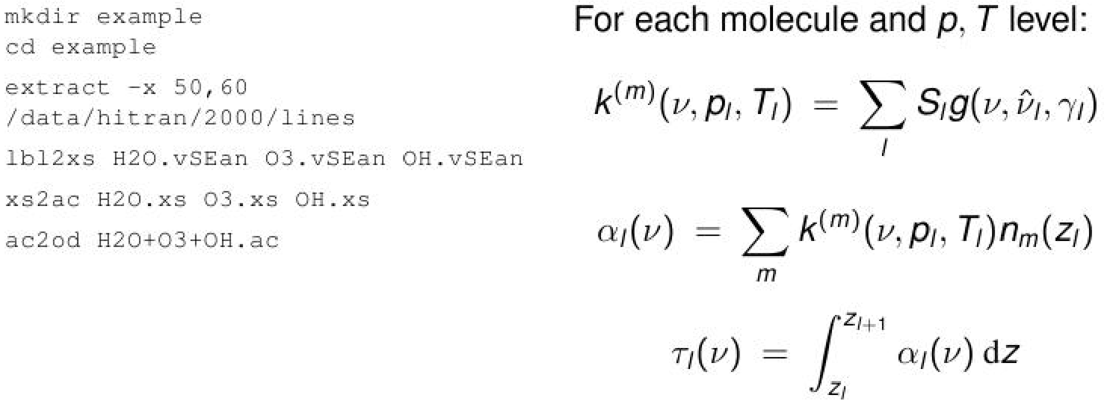


Figure 11: From Hitran/Geisa via cross sections (xs) and absorption coefficients (ac) to optical depths (od) and radiation intensity (ri). Note that cross sections are pressure and temperature dependent, absorption coefficients also depend on composition, and optical depth and radiation intensity depends on path geometry.



```
mkdir example
cd example
extract -x 50,60
/data/hitran/2000/lines
lbl2xs H2O.vSEan O3.vSEan OH.vSEan
xs2ac H2O.xs O3.xs OH.xs
ac2od H2O+O3+OH.ac
```

For each molecule and p, T level:

$$k^{(m)}(\nu, p_l, T_l) = \sum_l S_l g(\nu, \hat{\nu}_l, \gamma_l)$$

$$\alpha_l(\nu) = \sum_m k^{(m)}(\nu, p_l, T_l) n_m(z_l)$$

$$\tau_l(\nu) = \int_{z_l}^{z_{l+1}} \alpha_l(\nu)\, \mathrm{d}z$$

Figure 12: Typical workflow for line-by-line modelling with Py4CAtS: Hitran/Geisa ⟶ line parameter extracts ⟶ cross sections ⟶ absorption coefficients ⟶ optical depth
Note that on the left hand side the output files are not indicated (usually done with the `-o` option). Furthermore specification of atmospheric data (pressure and temperature for `lbl2xs`, and additionally gas concentrations for `xs2ac` is not shown.)

5 Py4CAtS — The Python Scripts

Py4CAtS is a Python (`www.python.org`) re-implementation of the Fortran infrared radiative transfer code MIRART/GARLIC. Clearly a pure Python implementations would be by far too slow for a computational challenging task such as line-by-line modeling, so Py4CAtS makes heavy use of the Numeric Python extensions (`www.numpy.org`, Langtangen [2004]). The motivation to rewrite the code in Python was to provide easy access to intermediate quantities such as cross sections, absorption coefficients, or optical depths that is sometimes quite useful to deepen the understanding of the "physics" involved in a particular remote sensing application. Furthermore this appeared to be a way towards "computational steering", i.e., combining the best of two worlds by letting Python do the control, book-keeping etc., and executing the compute-intensive code-sections in compiled Fortran. However, the original approach with PyFort Dubois and Yang [1999] turned out to be somewhat difficult to port from machine to machine, and the recent advances with Numeric Python (allowing highly optimized array-processing) made this need for Fortran–Python interfacing less critical.

In Py4CAtS the individual steps of an infrared radiative transfer computation are implemented in separate scripts, see Fig. 11:

- `higstract`: Hitran-Geisa-extract (select, grep, ...) lines of relevant molecules in the spectral range of interest[4]
- `lbl2xs`: compute line-by-line cross sections for given pressure(s) and temperature(s)
- `xs2ac`: multiply cross sections with number densities and sum over all molecules
- `ac2od`: integrate absorption coefficients along the line-of-sight through atmosphere
- `od2ri`: solve Schwarzschild equation (1b), i.e. integrate Planck function vs. optical depth along the line-of-sight through atmosphere (assuming a plane-parallel, non-scattering, LTE atmosphere, the Schwarzschild equation can be easily evaluated by standard numerical quadrature rules)
- ......

All these scripts read their input from external files, and save their results on files, too, see the workflow indicated in Fig. 12. As a consequence, I/O operations can become quite time consuming (as the number of spectral grid points can become quite big); Furthermore a large part of the scripts was devoted to check the consistency of the various input files (e.g., the `xs2ac` script had to test that the different cross section files cover the same spectral range (or at least a common subset) for the same altitude range etc.) On the other hand, circumventing some of the intermediate files is straightforward, especially if one is mainly interested in the final optical depth, cf. Fig. 13:

- `lbl2ac` compute line-by-line cross sections (for a series of p, T pairs) and combine to absorption coefficients;
- `lbl2od` compute line-by-line cross sections and absorption coefficients as in `lbl2ac`, then integrate through the atmosphere.
- ○ *NOTE:* See the next section 6 "Py4CAtS within the (i)python shell" on an alternative approach to bypass the I/O operations.

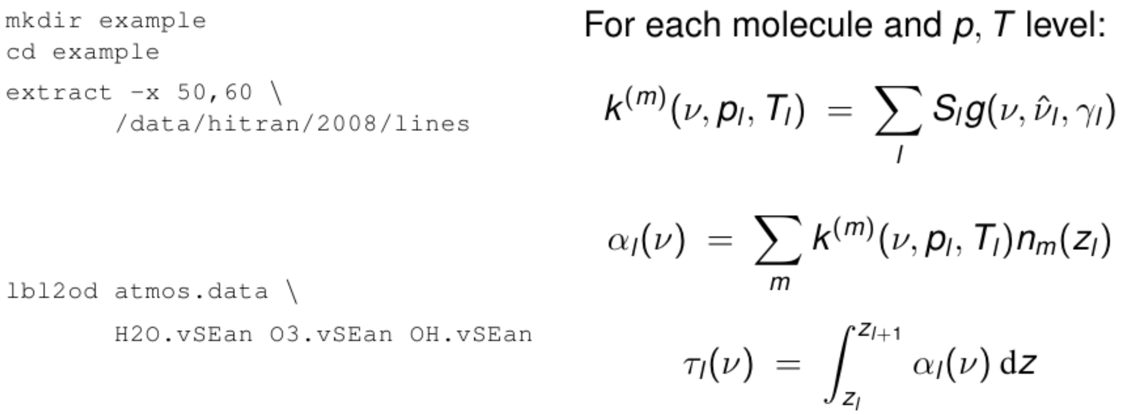
Finally, there are some scripts for quick plots, analysis and conversions:

- `atmos1D`: read atmospheric data (p, T, VMR's, ...) and convert, reformat, ...
- `molecules`: show properties (mass, ...) of IR relevant molecules
- `plot_atlas`: read spectral lines (from `higstract`) and plot
- `lines`: read spectral lines (from `higstract`) and convert to new p, T
- `xSection`: read cross sections, plot and save again ......
- `oDepth`: read optical depths and plot or ...

and some further scripts that are essentially subsidiaries (subroutines) required by the "main" scripts, but can be run independently, e.g.

---

[4] Note that this module has been called `extract.py` in the Python 2 version and has been renamed to `higstract.py` when porting to Python 3 in line with the function to be used inside the (I)Python interpreter (see subsection 6.2.2) to avoid a name clash with numpy's `extract` function!



```
mkdir example
cd example
extract -x 50,60 \
        /data/hitran/2008/lines
```

For each molecule and p, T level:

$$k^{(m)}(\nu, p_l, T_l) \;=\; \sum_l S_l g(\nu, \hat{\nu}_l, \gamma_l)$$

$$\alpha_l(\nu) \;=\; \sum_m k^{(m)}(\nu, p_l, T_l) n_m(z_l)$$

```
lbl2od atmos.data \
        H2O.vSEan O3.vSEan OH.vSEan
```

$$\tau_l(\nu) \;=\; \int_{z_l}^{z_{l+1}} \alpha_l(\nu)\,\mathrm{d}z$$

Figure 13: Typical workflow: From Hitran/Geisa line parameter extract(s) directly to optical depths.

- `hitran`: read the HITRAN database (subsidiary to `higstract`)
- `geisa`: read the GEISA database (subsidiary to `higstract`)
- `cgsUnit`: unit conversion
  NOTE: Internally py4cats uses cgs units **!!!**

5.1 Examples

In the following it is assumed that you have properly installed Py4CAtS, i.e., the executables in the `bin` subdirectory of Py4CAtS are in the search path (see the html documentation how to adjust the search path). Note that the executables in this `bin` directory should be symbolic links to the scripts in the `src` directory (with the `.py` extension omitted).

5.1.1 Near Infrared

Assume you are interested in the so-called “O2A band”. To get started lets check the HITRAN database where oxygen has spectral lines (note that the actual location of the Hitran (or Geisa) database on your computer might be different, here it is assumed that line data are stored in a `/data/` directory with `hitran/` and `geisa/` subdirectories, that in turn are further split into subsubdirectories for the different versions):

```
higstract -mO2 /data/hitran/2008/lines
```

This will write hundreds of lines to the screen, so tell the script to write its output to a file

```
higstract -mO2 -oO2.vSEan /data/hitran/2008/lines
```

The extension `.vSEan` tells the script to save the lines in a tabular form with five columns for the “core” parameters line position $\hat{\nu}$ (note the visual similarity with the latin “`v`”), strength S, energy E, air-broadening widths and exponent n (i.e. not in the original hitran format). The script informs you that it finds `1314 lines in 6256.380089 ... 15927.230093 cm-1`, and you can visualize these with

```
lines --plot S O2.vSEan
```

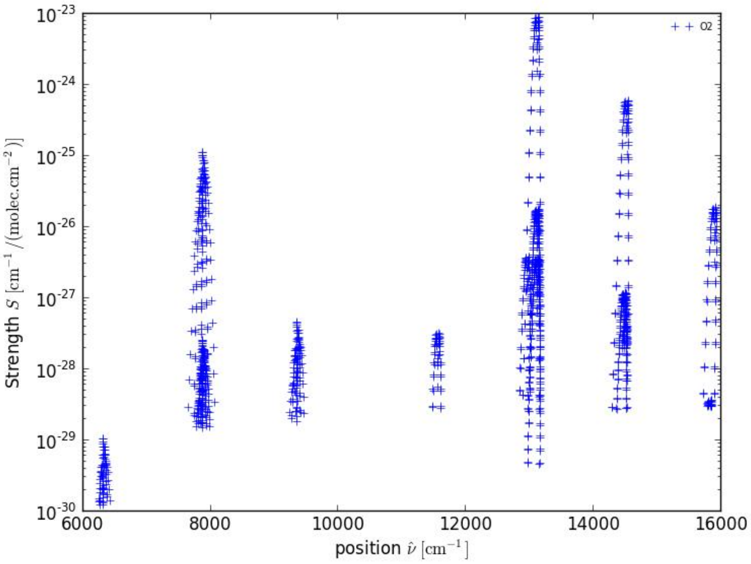


Figure 14: O_2 lines found in Hitran 2008 with the `extract` script and plotted with the `lines` script.

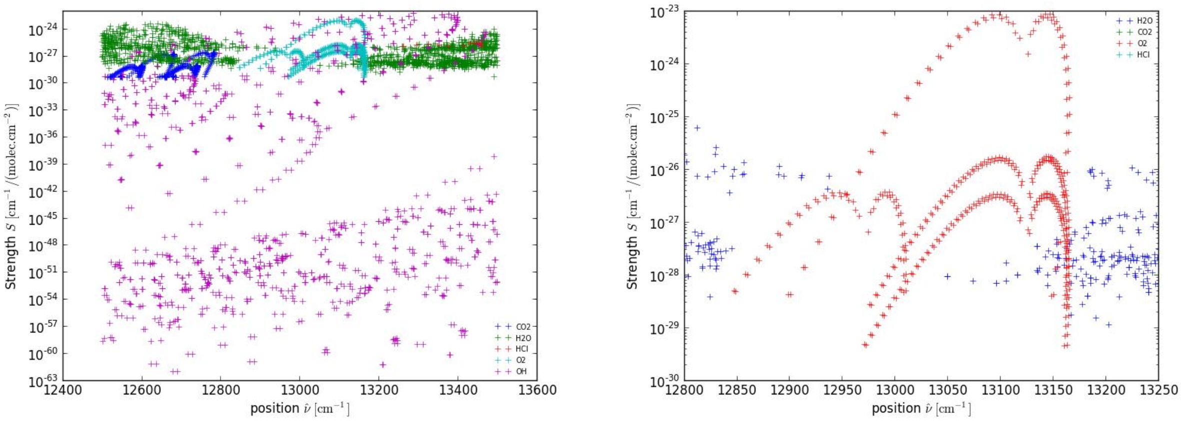


Figure 15: Lines found in Hitran 2008 in the region of the O2A band.

which produces the plot in Fig. 5.1.1. (Calling the script without any options simply gives a summary.)

Obviously the strongest lines are found in the region around 13 000 cm^{-1}, which is exactly the O2A band complex. In the following lets concentrate on this spectral region and first look into Hitran again for any other molecules absorbing in this spectral region:

```
higstract -x 12500,13500 -o O2.vSEan /data/hitran/2008/lines
```

which results in `22787 lines extracted from /data/hitran/2008/lines` with five more molecules in addition to oxygen: H_2O, CO_2, OH, HCl, and HF, see Fig. 15.

Next take a look at the cross sections defined in (6):

```
lbl2xs -oO2.xs O2.vSEan
```

Note that the `-o` option has been used again in order to save the data to a file, which then can be plotted easily (Fig. 16 left):

```
xSection --plot O2.xs
```

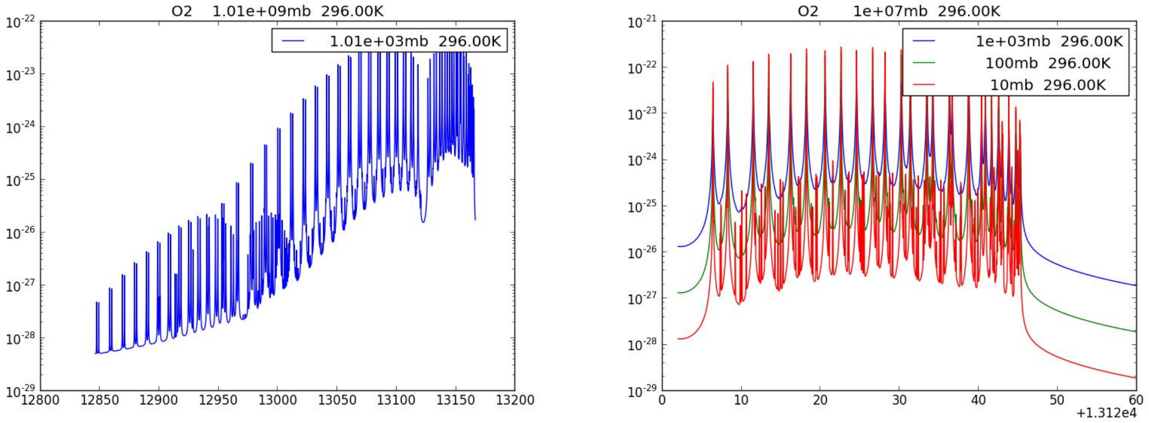


Figure 16: Cross sections of O_2 in the region of the O2A band.

Cross sections are pressure dependent due to air-broadening, see subsection 2.2.2, and this can readily be studied with

```
lbl2xs -p 1000,100,10 -x13122,13180 -oO2.xs O2.vSEan
```

where the wavenumber range has been restricted to see things clearer (Fig. 16 right). In a similar way the temperature dependence can be studied using the `-T` option.

Cross sections of all molecules absorbing in this spectral region can also be generated with this script, essentially `lbl2xs *.xs` or more specifically like this

```
lbl2xs -x 13122,13180 -fa -o xs O2.vSEan H2O.vSEan
```

Because several input files are used (all with the same extension!) the script automatically replaces the common extension “`vSEan`”with “`xs`”, i.e. the `-o xs` option is used to indicate the output files' extension. Furthermore the `-f a` option has been used to force an ascii tabular output format (default is to save cross sections in the Python specific “pickle” format), so the cross sections can be plotted with any plot package, e.g. ACE/Gr (a.k.a. xmgr, see `https://github.com/mlund/xmgr-resurrection`) or xmgrace (`http://plasma-gate.weizmann.ac.il/Grace`).

Finally look at optical depths. Assume you have atmospheric data also stored in the `/data/` directory (as for the line data the actual path/location of the atmospheric data file(s) might be different on your computer!), then use the command

```
lbl2od -o nir.dod /data/atmos/20/US.xy O2.vSEan H2O.vSEan
```

where the output file extension “`dod`” has been selected to indicate “delta optical depth”. Note that `lbl2od` assumes that the very first input file contains the atmospheric data (here a coarse, 20 level version of the US Standard atmosphere). If you only need the total optical depth (sum of all layer optical depths) you can either produce them directly using the mode option, i.e. `lbl2od -mt ...`, or you can combine the layer optical depths just produced with

```
oDepth -mt -o nir.tod nir.dod
```

(The mode “t” converts optical depth to transmission ($\tau \longrightarrow \mathcal{T} = e^{-\tau}$), and mode “T” produces total transmission.

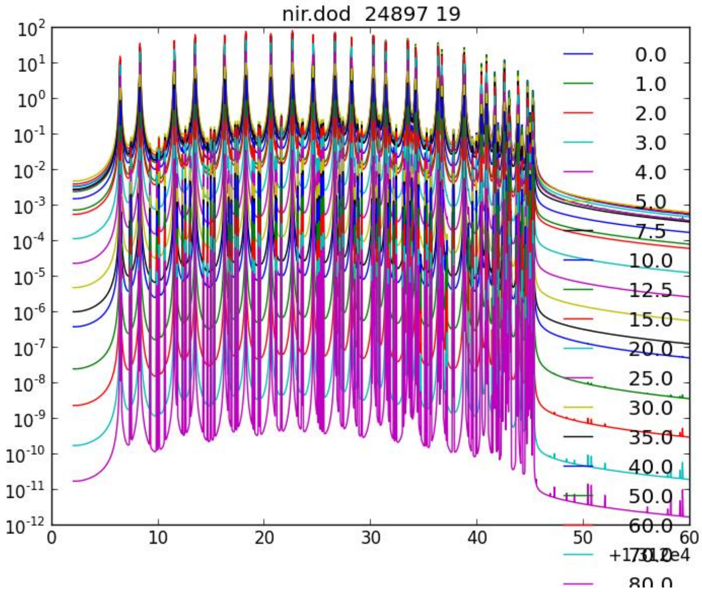


Figure 17: Delta optical depths in the region of the O2A band. The title indicates the number of spectral data points and the number of layers (nLevels-1). (Note that matplotlib's legends are not visually "perfect".)

5.1.2 Far Infrared

The OH radical has several groups of transitions in the far infrared, e.g. one around 83.869 cm^{-1}. To properly model the impact of line wings lets extend the 83 – 84 cm^{-1} spectral range and extract the molecular lines (remember that the path to Hitran used here might not work on your system):

```
higstract --main -x 73,94 /data/hitran/2008/lines
```

The boolean option tells `higstract` to select only the "main" molecules, i.e. the seven molecules collected in the very first edition of the Hitran database (selecting all molecules would have generated list for some dozen molecules). OH has been added in later editions, its molecule number is 13 in Hitran (or molecule # 14 in Geisa), so get its lines with another call of the script:

```
higstract -mOH -x 73,94 -oOH.vSEan /data/hitran/2008/lines
```

These two commands produce six files, that can be plotted with

```
plot_atlas -a -g9 *.vSEan
```

where the boolean `-a` option choses the old xmgr based plotting tool, and the `-g` option distributes the datasets to separate graphs as shown in Fig. 18 (left).

In a next step generate cross sections of the three "more important" molecules for two pressures

```
lbl2xs -p 100,1 -fxy -o xs -x83,84 H2O.vSEan O3.vSEan OH.vSEan
```

and plot these, again using xmgr

```
xmgr -log y -nxy -legend load *.xs
```

After some fine tuning this gives the plot shown on the right of Fig. 18.

In a third step compute optical depths

```
lbl2od -o fir.dod --BoA 10 /data/atmos/USS_20.xy {H2O,O3,OH}.vSEan
```

Because the troposphere is largely opaque in the far infrared (absorption by water), only optical depths above 10 km altitude are calculated.

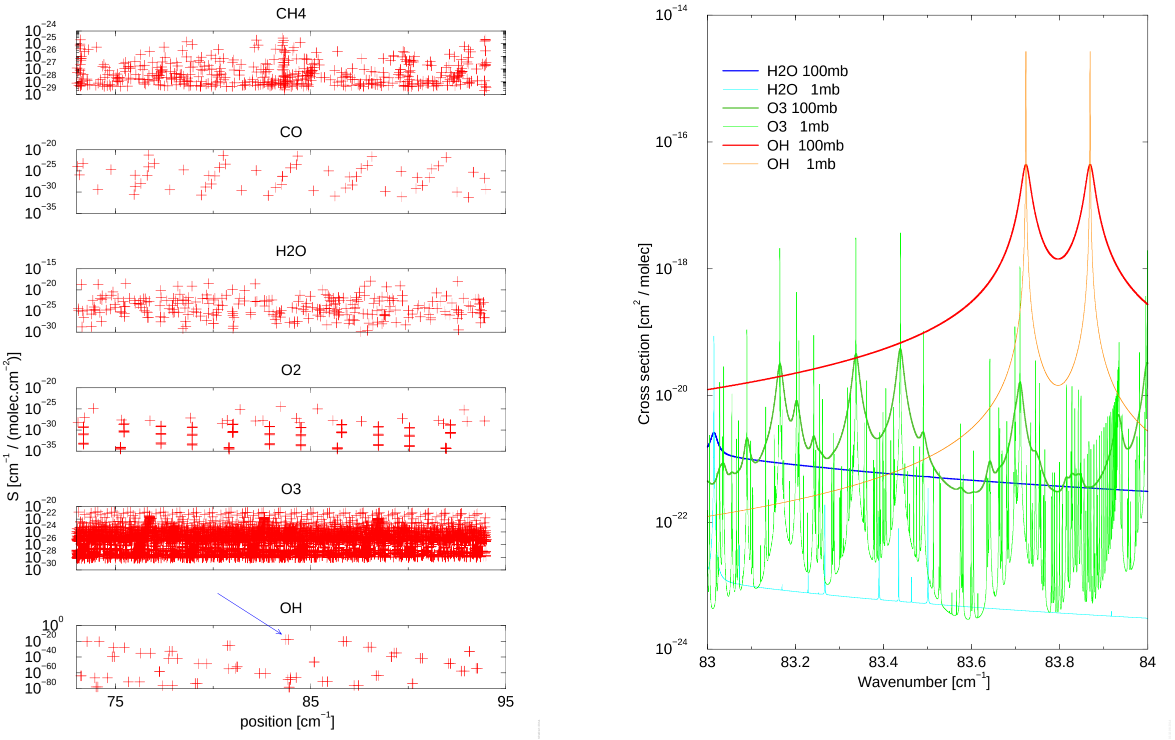


Figure 18: Left: lines found in Hitran 2008 in the FIR region next to the OH triplet around 83.869 cm^{-1}. (The blue arrow indicates the OH triplet of interest.) Right: some cross sections.

5.2 Some Notes on Options

- Some options are used by (almost) all scripts:

  `-h` to request help (usage description)

  `-c` to override the default character `#` used to indicate comment lines in data files

  `-o` to specify an output file (default: standard out)

  `-v` to request more informative output messages (verbose)

- Many options require one or several argument(s) (except for the boolean ones such as -h). The type of argument(s) is indicated in the help documentation:

  `char` a single character (usually a letter or digit)

  `string` a string (if there are blanks, enclose the string in quotes)

  `file` essentially a string specifying a valid file name

  `int` an integer

  `float` a real number (e.g. 3.14 or 1.23e45) (the "d" or "D" for the exponent of 10 is not allowed)

  `interval` two (usually real) numbers separated by comma (no blanks!)

- Some scripts allow to specify a list of integers, floats, strings etc. as an option (e.g. a sequence of pressures). Use commas as separator, but do not use blanks between the list elements, i.e. say `lbl2xs -p 1013,800,500` (or enclose everything in quotes: `lbl2xs -p '1013, 800, 500'`)
- Some scripts such as lbl2xs produce an output file for each input file. In these cases the -o option is used to specify the extension of the output file(s) (Typically the input files will have the same extension, and there will be a list of output files with the common extension replaced by the string given in the -o option)

5.3 The Atmospheric Datafile

The file containing an user's atmospheric profile data has to be in tabular / ascii / xy format:

- comment lines in this file have to begin with `#` in the first column
  (the `-c` option can be used to change this `commentChar`)
- a comment line starting with `#where:` can be used to include some info for the job
- at least 2 comment lines are mandatory:
  `#what:` followed by identifiers, e.g. altitude, molecular names, pressure, or temperature
  `#units:` followed by the physical units, e.g. km, mb, K, ppm
- a column for altitudes is mandatory (marked "ALTITUDE" or "HEIGHT" or "z" in the `#what:` record)
- columns with profiles not requested by the user (e.g. IR inactive gases) are ignored
- the number of columns in the data section and the number of identifiers and units given above have to be identical

Example:

```
#comment  this line will be ignored
#info:    this is ignored, too
#note:    the next line is optional
#where:   bavarian winter
#what:   z  pressure temperature        H2O      CO2        O3        CO
#units: km        mb           K        ppm      ppm       ppm       ppm
       0.0      1018     272.200   4.32e+03      360    0.0278      0.15
       1.0     897.3     268.700   3.45e+03      360     0.028     0.145
       2.0     789.7     265.200   2.79e+03      360    0.0285      0.14
       3.0     693.8     261.700   2.09e+03      360     0.032     0.135
       4.0     608.1     255.700   1.28e+03      360    0.0357     0.131
       5.0     531.3     249.700        824      360    0.0472      0.13
       7.5     373.4     234.700        170      360    0.0914     0.122
      10.0     256.8     219.700       29.6      360     0.237    0.0996
      20.0      53.7     215.200        4.5      360       2.9    0.0133
```

5.4 Misc Remarks

NOTE: See also the FAQ's in the html documentation.

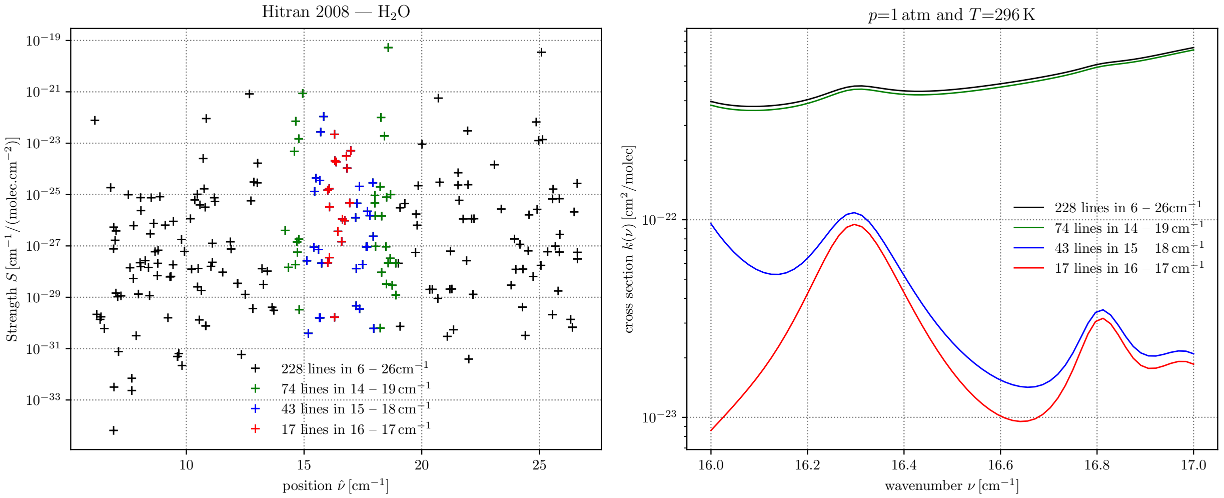


Figure 19: Impact of line wings on H_2O cross section in the ODIN [?] 501 GHz channel. A series of cross sections has been computed taking into account more and more lines to the left and right of the 16 to 17 cm^{-1} window.

Selection of spectral range, contributions from line wings

In order 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}) (the actual size of δ depends on the number, density, and strengths of lines outside 'your' interval and on further factors such as pressure). As the `higstract` and `lbl2od` (or `lbl2xs`) scripts are completely independent, this extension is not done automatically. The impact of line wing contributions on cross sections is demonstrated in Fig. 19.

Optical depths, transmission, and radiance for horizontal view

The functions `ac2od` or `lbl2od` do not have an angle as argument, so the optical depth returned is always the vertical optical depth thru the atmosphere. If you need transmission (or weighting functions) for a horizontal path, i.e. zenith angle 90°, hence a homogeneous atmosphere without any path dependence, then "extract" the absorption coefficient $\alpha(\nu, z)$ (that is defined for a set of altitude levels $z_1, z_2, \ldots, z_L$) for the corresponding altitude level z_l and evaluate transmission $\mathcal{T}(\nu) = \exp\left(-\alpha_l(\nu)s\right)$ as a function of path length s. To compute transmission as a function of (horizontal) distance, simply evaluate the exponent for an array (or list) of distances. Likewise, for the weighting functions compute the product of transmission times absorption coefficient for some distances (see Eq. (38)).

6 Py4CAtS used within the (i)python shell

Until recently the scripts provided in Py4CAtS could only be used as commands from the Unix/Linux shell. These scripts read their input data from external files, and save their results to files, too. As a consequence, I/O operations can become quite time–consuming esp. for "large" spectra.

However, in view of the impressive advancements and progress of interactive Python shells, in particular IPython (see `www.ipython.org`), the scripts have been substantially rewritten to provide an easy way to do lbl modeling within (i)python. Furthermore, a considerable part of the scripts presented in the previous chapter is devoted to consistency checks of the data read back from file(s).

6.1 Setting up IPython for Py4CAtS

NOTE: This setup is far from perfect and might change soon (e.g. when Py4CAtS is organized as a package and/or when setup tools such as distutils are used).
Furthermore, the details of the IPython configuration are — unfortunately — version dependent, so the description given here might be outdated if you use a more recent version.

There are several ways to start Py4CAtS in an interactive (I)Python console. Essentially the setup is done by executing the `py4cats.py` module, that is located in the `src` subdirectory of Py4CAtS.This script tries to "adjust" the Python search path (the list of directories in `sys.path`) and then import all "important" modules. Note that numpy (and scipy, matplotlib) are *not* imported here, it is assumed that you are doing this anyway.

Whatever your method to start Py4CAtS, you should see a list of modules/functions imported and finally the message `INFO --- py4cats:  setup done` along with the number of directories in Python's `sys.path` and the very first directory there, which should be the location of Py4CAtS's sources.

IPython without a new profile

Probably the easiest way is to start IPython, ideally with the `--pylab` option to automatically load numpy and matplotlib:

```
ipython --pylab &
```

or better (the qtconsole supports embedded matplotlib figures and makes command editing easier and more flexible; alternatively you might want to consider the `notebook`)

```
ipython qtconsole --pylab &
```

and then `%load` or `%run` Py4CAtS setup file by hand inside the IPython shell[5]

```
%load /a/very/long/path/to/py4cats/src/py4cats.py
```

or

```
%run /a/very/long/path/to/py4cats/src/py4cats.py
```

---

[5] Note that "sometimes" IPython apparently loads the file, but does not give a prompt afterwards. This happens even for "simple" test files to be loaded, and is probably related to the qtconsole and/or the IPython version.

IPython using the standard PYTHONSTARTUP

If you use Py4CAtS frequently, this becomes inconvenient and tedious. However, IPython is recognizing the “classical” Python startup file, so you can adjust the corresponding environment variable `PYTHONSTARTUP`[6]

```
ln -s /a/very/long/path/to/py4cats/src/py4cats.py ~/
setenv PYTHONSTARTUP ~/py4cats.py
```

(for the csh/tcsh, or an equivalent statement for the bash) and start IPython “as usual”.

Defining a IPython profile

As described in the IPython documentation, a new profile can be setup with the command (given on the (unix) shell)

```
ipython profile create py4cats
```

In a typical Unix/Linux system this will generate a new subdirectory in the `.config/ipython` directory in your home directory, e.g. `/users/schreier/.config/ipython/profile_py4cats/` with some files and subdirectories in it. Next copy (or symbolically link) the `py4cats.py` file,

```
cp ~/src/py4cats/src/py4cats.py ~/.config/ipython/profile_py4cats/startup/
```

where we assume that the Py4CAtS modules are in the `~/src/py4cats/src` directory. If Py4CAtS is somewhere else, change the `catsPath` variable in the `py4cats.py` file, i.e. correct `catsPath = os.path.join(os.path.expanduser('~'), 'src', 'py4cats')` to whatever else (e.g. `catsPath='/a/very/long/path/to/py4cats'`) and also adjust the copy command given above.)

Now you should be ready to start lbl modeling within the (i)python shell

```
ipython qtconsole --pylab --profile py4cats &
```

The `--pylab` option tells IPython to load numpy and matplotlib, too. Use `--pylab=inline` to embed the plots/figures in the qtconsole[7]. If you prefer the standard IPython shell or the IPython notebook, omit the `qtconsole` or replace it with `notebook`. If you have a “standard” Python startup file you want to ignore for the lbl modeling, it might be convenient to define an alias, e.g.

```
alias ipy4cats 'unsetenv PYTHONSTARTUP && ipython qtconsole --pylab --profile py4cats &'
```

---

[6] In the setup phase, Python automatically inserts the directory containing the startup file at the top of `sys.path` and removes this afterwards. So if `PYTHONSTARTUP` specifies the “true/actual” `py4cats.py` file, then the Py4CAtS directory (including this file) will be removed from `sys.path` after execution! Therefore link or copy `py4cats.py` to some convenient place, e.g. your home directory.

[7] According to `ipython qtconsole --help` graphics are inlined with the option `--matplotlib=inline`, but this (or `--matplotlib inline`) does not work if you also set `--pylab`! You can toggle between the modes with the IPython magics `%matplotlib qt` and `%matplotlib inline`

Alternatives without IPython

As an alternative (e.g. in case you do not use ipython) you can use the `py4cats.py` file as Python's startup file (see above). Then start Python as usual.

Another possibility is to simply start the "good/old/traditional" Python shell and tell Python to read the Py4CAtS startup file before going interactive, i.e. `python -i ~/src/py4cats/src/py4cats.py`.
Or, slightly more "advanced" with IDLE, Python's first (?) Integrated DeveLopment Environment:
`idle -r ~/src/py4cats/src/py4cats.py`. Note that in both cases Matplotlib's plotting function are not loaded automatically! Furthermore, note that IDLE does not show any error messages issued by `raise SystemExit` statements, so some operations might fail without any info/warning.

6.2 Examples

In the following it is assumed that you have properly installed Py4CAtS, e.g. using an IPython profile, i.e., when starting IPython it will automatically adjust the search path for modules and import (important) Py4CAtS modules. Or, at least, you should have `%load`ed Py4CAtS. Furthermore familiarity with Python and numpy is assumed.

The line data and atmospheric data required by Py4CAtS are stored internally in so-called "structured arrays", see appendix A.1. For example, reading one of the atmospheres in the `py4cats/data/atmos/` subdirectory will result in a structured array with several columns with field names ‘z‘, ‘p‘, ‘T‘, ‘H2O‘, ‘CO2‘, ... More precisely, if you have read one of the AFGL atmospheres given on the original altitude grid (available in the `py4cats/data/atmos/50/` subdirectory), then the structured array will have 50 rows.

6.2.1 Atmospheric Data

First read a dataset from a file (see subsection 5.3 for details) using the `atmRead` function[8] of the `atmos1D.py` module

```
mls = atmRead(’~/src/py4cats/data/atmos/20/mls.xy’)
```

and do some prints for information

```
print len(mls), gases(mls))
```

which should return 20 (the number of levels) and a list of gases such as `[’H2O’, ’CO2’, ’O3’, ’N2O’, ’CO’, ’CH4’, ’N2’]`. Note that `atmRead` expects a "complete" atmospheric data file, i.e. with altitude, pressure, temperature,*and* molecular concentrations. If you want to read a file with only concentrations vs. altitude, use the `vmrRead` function instead.

`mls` is an example of a structured array (see the appendix A.1), 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. Note that you can specify "rows" and "columns" in two ways; for example, the BoA (bottom-of-atmosphere) temperature is `mls[’T’][0] = mls[0][’T’]`.

Suppose you have read the midlatitude summer and winter atmospheres. Then you can easily compare the temperatures with

---

[8] Note that in the Python 2 version this function has been called `atmos1D`.

```
atmPlot ([mls,mlw])
```

which is essentially a shortcut for `plot (mls['T'],mls['z'], mlw['T'],mlw['z'])`. The summer ozone number density is depicted using `atmPot (mls,'O3')`. Note that `atmPlot` expects either a single structured array or a list thereof.

Molecular concentrations are stored internally as number densities (i.e. if the data file contains volume mixing ratios (VMR), these are converted to densities by multipication with the air number density $n = p/(kT)$). Volume mixing ratios (in units "pp1") can be obtained with the `vmr` function, e.g. `vmr(mls)`.

The function `vcd` integrates the (molecular and air) number densities along a vertical path through the atmosphere ($N = \int n(z)\,\mathrm{d}z$, default from bottom to top), e.g.
`vcd(mls)` or `vcd(mls, zMax=80.0)` or `vcd(mls, 'CO', zMin=3.0)`,
where the last example gives the CO column above the Zugspitze mountain. (Remember that Py4CAtS internally uses cgs units consistently, including *cm* for altitudes. As a convenience, PyCAtS interprets "very small" altitudes (smaller than 250) as *km*, assuming that for atmospheric radiative transfer altitudes smaller than 250 cm do not make sense. (Might not work perfect, but ...))

Finally, the "Column Mixing Ratio", i.e. the ratio of the molecular VCD's and the air VCD, can be computed by, e.g., `cmr(mls)`. And the `atmRegrid (mls, zNew, ...)` function allows to interpolate the atmospheric data to a new altitude grid (where `parseGridSpec` from the `grid` module can help to setup the new grid).

To combine atmospheric data from different files, use `atmMerge`, 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 will be 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".

6.2.2 Shortwave Infrared

Line Parameters: To model atmospheric absorption in SCIAMACHY's channel 8 mainly used for CO retrievals we need the spectral lines of CO and the interfering species: [9]

```
dictOfLineLists = higstract('/data/hitran/2012/lines', (4250,4330))
```

returns a dictionary of 20 line lists, more precisely structured arrays, one array for each molecule with transitions in this spectral range around 2.3 μm. Actually, these arrays are subclassed numpy arrays `lineArray` holding some extra information as attributes, e.g. `dictOfLineLists['CO'].p` and `dictOfLineLists['CO'].t` will give Hitran's reference pressure 1013.25 10^3 dyn/cm^2 and temperature 296.0 K, respectively. If you extract lines from a single molecule only (e.g., `higstract('/data/geisa/2003/lines', molecule='CO')`), a single `lineArray` is returned. (However, if you don't specify a molecule, but `higstract` finds lines from just one molecule in the given spectral range, a dictionary is returned with just one entry.)

You can easily plot the line parameters using the standard matplotlib functions, e.g. `semilogy (dictOfLineLists['CO']['v'], dictOfLineLists['CO']['S'],'+')` for line position vs. strength, or more simply with Py4CAtS' own `atlas` function, e.g.

---

[9] The module containing this function has been called `extract.py` in the Python 2 version and has been renamed to `higstract.py`. Unfortunately a function with the same name would conflict with numpy's `extract` function, so the new name `higstract`, short for hitran-geisa-extract.

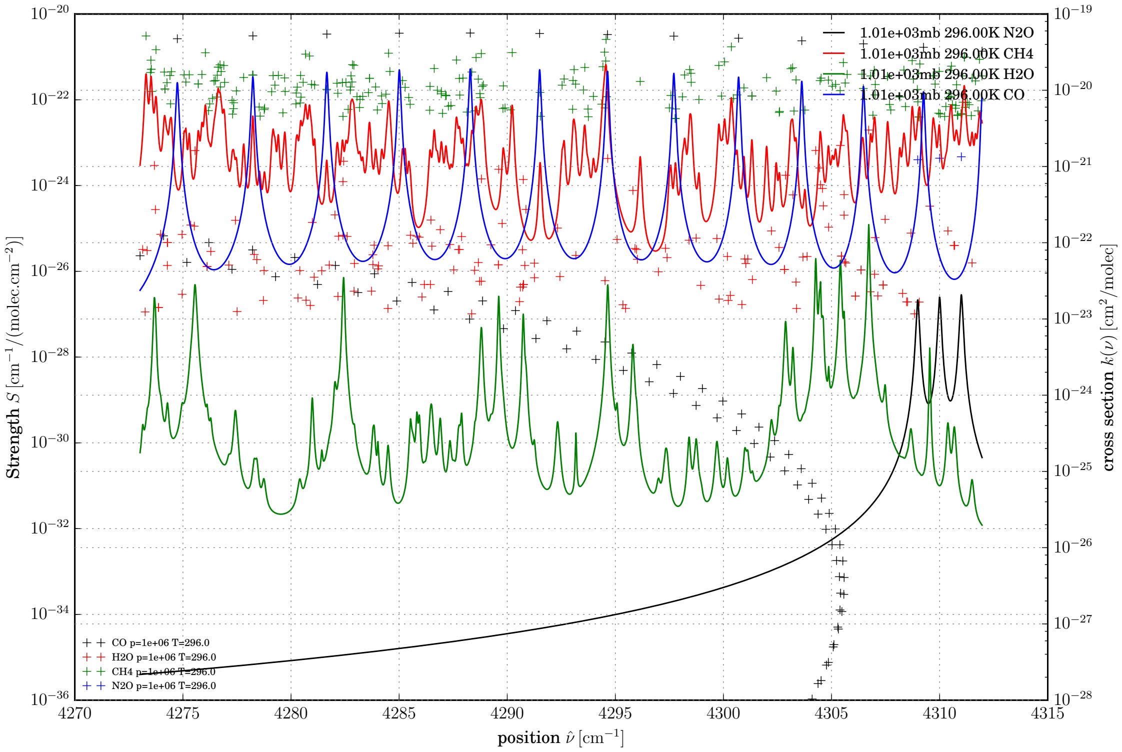


Figure 20: Combination of line atlas and xsPlot. This example has been generated with

```
dll = higstract('/data/hitran/2000/lines',(4273,4312), 'main')
xss = lbl2xs(dll)
atlas(dll); twinx(); xsPlot(xss)
```

```
atlas(dictOfLineLists['CO'], yType='E')
```

will show you the lower state energies of the CO transitions, or the linestrengths of all molecules

```
atlas(dictOfLineLists)
```

The `atlas` function automatically selects a logarithmic y-axis for line strengths (or energies) and a linear scale otherwise. (If you don't like it, you can easily toggle the 'log' ↔ 'linear' scaling of y-axes using matplotlib's event keys, see `http://matplotlib.org/users/navigation_toolbar.html#navigation-keyboard-shortcuts`.) If later on you want to see the molecular cross section (or absorption coefficients or optical depths etc.) overlayed on top of the line atlas, you can you use matplotlib's `twinx()` function, see Fig. 20.

The (core) line parameters of a certain molecule can be saved on file using the `write_lines_xy` function (the output corresponds to the file produced by the `extract.py` script, see subsection 5.1). Later on you can read a set of line parameter files with the function `get_dictOfLineLists`, and `read_line_file` reads a single file. (*Note December 2016:* `read_line_file` now works recursively, i.e. returns a `dictOfLineLists` when you give a list of files.)

Absorption cross sections: Having read atmospheric and spectroscopic data you are ready to compute absorption cross sections (in units cm^2/molecules) as defined in Eq. (6):

`xs = lbl2xs(dictOfLineLists['CO'])` will return the cross section of CO in the spectral range around 4300 cm^{-1} for the database (here HITRAN) reference pressure and temperature. Actually `lbl2xs` returns a subclassed numpy array (i.e. `type(xs) = xsArray`) with the "attributes" such as lower and upper wavenumber bound, pressure, temperature, and molecule stored in further items, e.g. `xs.x` or `xs.p`. Note that the cross sections are evaluated on a uniform (equidistant) wavenumber grid, so it is sufficient (and more memory efficient) to save the very first and very last grid point only.

To get cross sections for different p, T try [10]

`lbl2xs(dictOfLineLists['CO'], pressure=500e3)`

for $p = 500\,\mathrm{mb} = 500 \cdot 10^3\,\mathrm{g/cm/s^2}$ or

`xss = lbl2xs(dictOfLineLists['CO'], temperature=[200,300,400])`.

CO cross sections for all levels (i.e. p, T pairs) of the midlatitude summer atmosphere[11] can be obtained by

`xss = lbl2xs(dictOfLineLists['CO'], mls['p'], mls['T'] )`

and cross sections for all molecules and levels are produced by

`xssDict = lbl2xs(dictOfLineLists, mls['p'], mls['T'], (4280.,4305.)  )`

In the very last example we have also specified the wavenumber range as the fourth argument, because otherwise `lbl2xs` (or `lbl2ac` and `lbl2od`) would consider all lines extracted from Hitran/Geisa (see also the first remark in subsection 5.4).

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 number of molecules. In the very first example (CO and one p, T) above 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 will give a dictionary of lists of `xsArray`'s, each list for a single molecule and a dictionary entry for each molecule.

Because the cross-section-dictionary does not store the wavenumber grid, you cannot simply plot cross sections vs. wavenumber using matplotlib's function. The `xSection.py` module has a function `xsPlot` to visualize the cross sections, e.g. `xsPlot(xss)`. This function works recursively, i.e. it can be called with a single `xsArray`, a list thereof, or a dictionary of (lists of) `xsArray`'s.

To save cross section(s) to file(s) use the `xsSave (xss, ...)` function of the `xSection.py` module.

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 (5) for all levels are generated with `absCoList = 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(absCoList[0])` → `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,

---

[10] Important: remember that "internally" `lbl2xs` expects cgs units, esp. pressures in dyn/cm^2 = g/cm/s^2. As a convenience, you can give a list/tuple of pressures with the unit as very first or very last entry, e.g. `pressure = [1013, 300., 100., 'mb']`

[11] See the example in subsubsection 6.2.1.

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

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

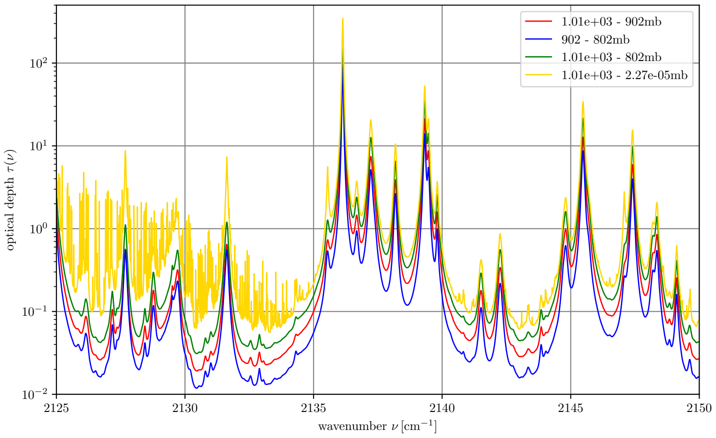


Figure 21: Computing and combining optical depths.

respectively. Note that the number of levels in the atmospheric data set (here `mls`) and the lengths of the cross section lists in the `xssDict` has to 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 (in units 1/cm) can be easily plotted with the standard matplotlib functions, but Py4CAtS also has a function to make this easier: `acPlot(absCoList)`. The function `acInfo(absCoList)` prints essential information about the absorption coefficients (Actually its just a loop calling the corresponding `info` method of `acArray`, i.e. `for ac in absCoList: ac.info()`).

The data can be saved to file (tabular-ascii) with the standard numpy `savetxt` or Py4CAtS’ `awrite` function. The `acSave` function will automatically save the absorption coefficients along with the atmospheric data, and the `acRead` function allows you to read 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.

Optical depth: The next step is to integrate the absorption coefficients along the (vertical) path through the atmosphere: `dodList = ac2dod (acList)`. Similar to cross sections and absorption coefficients this will return 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).

Note that these 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 `dodList[0]+dodList[1]` (Note that addition will fail if the two layers are not neighboring). If you are interested in the total optical depth only, simply use `tod = dod2tod(dodList)`, see Fig. 21. Likewise, if you want to have the accumulated optical depth do `codList = dod2cod(dodList)` to start accumulating with the very first (usually at BoA) layer (default) or `codList = dod2cod(dodList,True)` to start accumulating with the very last layer. In the first case, the very last element of the `cod` list should be the total optical depth, in the `back=True` case the very first `cod[0]` corresponds to the total opotical depth.

Save the data to a file with `savetxt('myOptDepth.xy', column_stack([vGrid, od]))`, or slightly more convenient using Py4CAtS’ `awrite` function, i.e. `awrite([vGrid,` `dod], 'myOpticalDepth.xy')`. However, in both cases all the "attributes" are lost, these functions only save the "raw" numpy arrays, so the recommended approach is `odSave (dodList, 'myOptDepth')`. Further options allow to select netcdf output and to convert from wavenumbers to wavelengths (nanometer). Later on, you can read the optical depth data back from file into a (new) ipython session with `oDepth = odRead (odFile)`.

Similar to the cross sections, optical depths can be plotted using the standard matplotlib functions, but you can also use the "special" `odPlot (optDepth)`.

The `oDepth` function in the `oDepth.py` module offers several possibilities to "transform" the optical depth, e.g. to sum-up the delta optical depth to the total optical depth.

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.

Weighting functions 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 Eq. (1a). They are defined by $\frac{\partial \mathcal{T}}{\partial z}$ for a vertical path, or more generally

$$\frac{\partial \mathcal{T}(\nu, s)}{\partial s} = -\mathcal{T}(\nu, s)\,\alpha(\nu, s) \tag{38}$$

for a slant path with $s = z/\cos\theta$ (where α is the absorption coefficient, see (5)) and can be computed with `wgtFct = ac2wf(acList,zObs,angle)` function. (The zenith angle θ is zero for an uplooking observer and 180° for a downlooking observer.) Optionally `ac2wf` also allows to treat 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'`). Like cross sections, absorption coefficients, and optical depth the weighting functions are stored in a sub-classed numpy array `wfArray` with special attributes for the limits of the wavenumber grid `vGrid`, path distance, viewing angle etc.

Alternatively, given the delta/layer optical depths the weighting functions can be approximated by finite differencing using `dod2wf(dodList,zObs,angle)` function, but starting from the absorption coefficient is much more reliable!

Note that the weighting functions returned by `ac2wf` and `dod2wf` are "matrices" with `nLevels` columns and `len(vGrid)` rows. Furthermore, `sGrid` contains the distances w.r.t. the observer, i.e. from ToA to BoA in case of a downlooking nadir view (in this case you can easily "translate" to altitudes in kilometers with `cgs('!km',mls['z'][-1]-sGrid)`).

The function `wfPlot(wgtFct, wavenumber, header)` provides a simple visualization tool (with all arguments except for the first being optional), and `wfSave(wgtFct,...` writes the data to a file.

For weighting functions of a horizontal path (zenith angle $\theta = 90°$) see the second remark in subsection 5.4.

Radiation intensity: The `dod2ri` function can be used to evaluate the Schwarzschild integral (1b), i.e. the Planck function $B(\nu, T)$ is integrated along a line-of-sight through the atmosphere. `radiance = dod2ri (dod)` will return the intensity (again a sub-classed numpy array with attributes for wavenumber interval, altitude, pressure, and temperature minimum/maximum, observer zenith angle, and background temperature) as seen by an uplooking observer at the surface ("bottom-of-atmosphere", BoA), whereas `dod2ri (dod, 180.0, mls['T'][0])` will give the radiance for a nadir-viewing observer looking down from top-of-atmosphere (ToA) with an angle of 180.0° (relative to the zenith angle) to Earth (or whatever ...); 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 Eq. (1) with $I_\mathrm{b}(\nu) = B(\nu, T_\mathrm{b})$.

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 "observer inside" in App. D.)

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 subsection 3.4.

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 with a spectral response function, a special method `convolve` has been implemented, e.g. `radBox1 = radiance.convolve()` will use the default "box" with a half width 1.0. Likewise, `radGauss2 = radiance.convolve(2.0,'G')` will use a Gaussian response function with HWHM 2.0.

Shortcuts: If you do not need the absorption coefficients, you can directly go from cross sections to optical depth with `deltaOptDepthList = xs2dod (mls, xssDict)`. Furthermore, you can bypass the cross sections with the `lbl2od` function, e.g.

```
deltaOptDepthList = lbl2od (mls, lineListsDict)
```

or similarly `acList = lbl2ac (mls, lineListsDict)`. Note that `lbl2od` "inherits" most options accepted by `lbl2xs` or `xs2ac`, in particular the `mode` option.

6.3 Further remarks

In the previous subsection a typical iPy4CAtS session has been worked out for demonstration. Here we give a brief (and probably incomplete) survey of some "common" things.

6.3.1 Input/Output:

Cross sections, absorption coefficients, and optical depths can be read from data files with the functions `xsRead`, `acRead`, and `odRead`. Likewise, these data are written to files using `xsSave`, `acSave`, and `odSave`. 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.

Atmospheric data are read from ascii tabular files with the `atmRead` function (see subsection 6.2.1) and can be written (back) to file with `atmSave`. And finally, the `riSave` function can be used to write a radiance spectrum to file.

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 (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 will return a dictionary with a "`lineArray`" for each molecule.

All routines saving data in ascii format use the `awrite` function from the `aeiou` module, see the appendix A.3 for details.

6.3.2 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.4). 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 (so n must be 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 in order 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 Fig. 21. 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 will be given on different wavenumber grids, so the coarser spectrum will be 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 Appendix A.1) with attributes holding information about molecule and reference pressure and temperature.

6.3.3 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 coefficient, optical depths, radiance/intensity, and weighting functions. The name `atlas` goes back to the *Atlas of absorption lines from 0 to 17900* cm^{-1} by Park et al. [1987] 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, you can exploit the source code of these functions as a starting point for more sophisticated plots. And you can change x or y axis labels and limits, legend (entries, positions, ...), title, and curve colors, markers, styles and widths interactively in IPython / matplotlib.

6.3.4 Recursive functions

Some functions exploit Python's recursive capabilities in order 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 list), their length has to 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. And `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.

6.3.5 Miscellaneous: Conversion of physical units

The function `cgs` from the cgsUnits module can be used to convert physical quantities (scalar or array) to or from the cgs base unit, e.g. `cgs('kg') ⟶ 1000.` or `cgs('!km') ⟶ 1e-5` or `cgs('mb ! atm',1013.25) ⟶ 1.0` (where the exclamation mark separates the original (input) and final (output) unit). Note that the optional second argument "`data`" can be a float, list/tuple of floats, or a numpy array (In case of several `data` a array is always returned).

Two further modules `radiance2radiance.py` and `radiance2Kelvin.py` help to convert radiances. With the function `radiance2radiance` you can change the power and/or area and/or spectral unit of radiances, e.g. nW ↔ erg/s or wavenumber ↔ frequency ↔ wavelength. Likewise, `radiance2Kelvin` allows you to convert radiances to equivalent brightness temperatures using the "inverse" of the Planck function (2).

6.3.6 Geometry

As mentioned above (section 3.4, see also appendix D) Py4CAtS assumes a plane-parallel geometry, i.e. simulations of transmission and radiance for a limb-sounding configurations are not possible. Nevertheless, a sketch of the geometry for uplooking and downlooking (nadir) viewing in a spherical atmosphere might be useful. In particular, the "viewing angle" α considered by GARLIC and Py4CAtS is always measured relative to the observer with respect to the zenith.

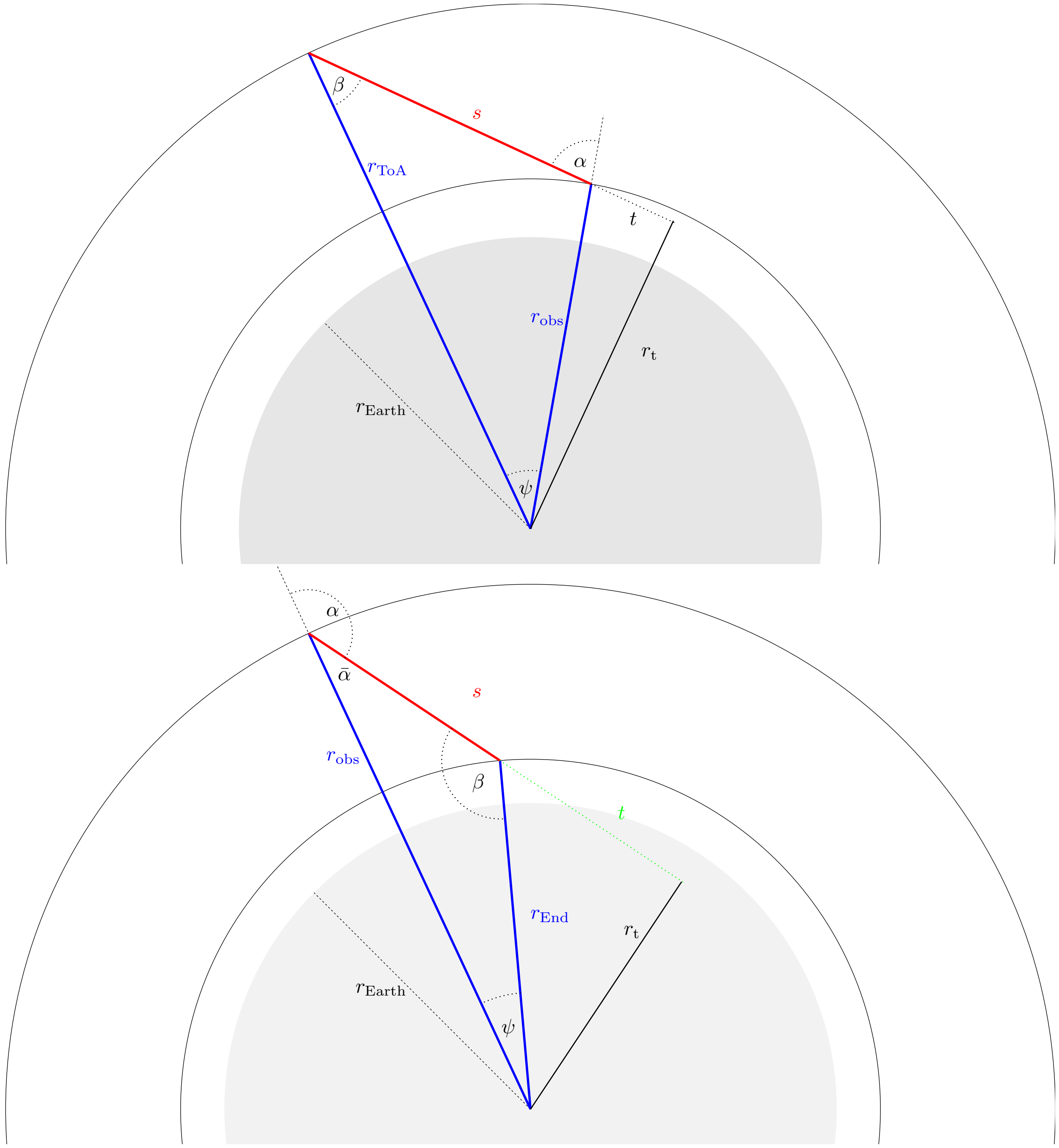


Figure 22: Geometry of uplooking (left) and downlooking (right) path: The observer is looking upwards (downwards) with an angle α from the zenith along the line-of-sight (red).

A Implementation Aspects

A.1 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 you can access the columns of these arrays by names instead of numbers, similar to dictionary entries.

Py4CAtS uses structured arrays for the atmospheric data (see subsection 6.2.1) and the line parameters (see subsection 6.2.2).

A.2 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 nicer features (currently) not available in our `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 possibilites, e.g. `diceNumber in [1,2,3,4,5,6]`, but a statement like `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.

A.3 Input/Output Utilities: the `aeiou.py` Module

A set of functions for some common tasks as reading and parsing the comment 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')`.

A.4 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.)`.

B ToDo's

- Packaging, disutils, ...
- Consolidate the various subclassed numpy arrays (`xsArray`, `acArray`, `odArray`, `riArray`, ..., see 6.3.2) using a super-subclass `specArray` defining the common things;
- Clean-up, esp. make names more consistent, e.g.
  'p' ↔ 'press' ↔ 'pressure' ↔ 'pressures';
- Interpolation: clean-up & more consistent, too;
- Exploit astropy (see `www.astropy.org`, esp. for unit conversion, maybe for data I/O);
- Finalize the new `atmos1D.py` (e.g. add regridding to the command line options),
  Merging "main gases" data with trace gases data files (also clean-up `data/atmos`);
- FoV for radiance, transmission
  Spectral response convolution

C Known Problems

- Module `atmos1D.py`: when this is called from `lbl2od` and the atmospheric data are converted to cgs units (e.g. pressure to dyn/cm^2), numpy raises a "FutureWarning" related to copying arrays**?¿?**
  (See also `https://github.com/numpy/numpy/issues/8383`.)
- Wavenumber interval limits, `xLimits`, impact of line wings, ...
- Python's search path `sys.path`, a list of directories to search for modules to be imported: After IPython has executed its config file(s), the very first directory in `sys.path` (pointing to this config) is removed. This might conflict with Py4CAtS' assumption, that `sys.path[0]` is the directory of Py4CAtS' source files.

D Limitations — What Py4CAtS cannot do

- Spherical atmospheres: modeling radiance and/or transmission for limb sounding is not possible (and probably will never be); Py4CAtS is assuming a plane parallel atmosphere.
- No continuum contribution to molecular absorption
- Scattering: so far only the Schwarzschild equation with the Planck function as source is supported. However, you can use the optical depths as input for any multiple scattering solver [e.g. DISORT Stamnes et al., 1988], see libRadtran [Mayer and Kylling, 2005, Emde et al., 2016].
- Observer "inside" a layer, i.e. observer altitude different from any atmospheric altitude grid point. If you definitely need an observer, say, at 3.14159 km you can interpolate the atmospheric profiles to a new grid including this point and proceed as usual.
- Line shapes beyond Voigt (well, "brute-force" line-mixing added recently)

References

M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions.* National Bureau of Standards, AMS55, New York, 1964. 3.2

B.H. Armstrong. Spectrum line profiles: The Voigt function. *JQSRT*, 7:61–88, 1967. doi: 10.1016/0022-4073(67)90057-X. 3.2

P. Bernath, C. T. McElroy, M. C. Abrams, C. D. Boone, M. Butler, C. Camy-Peyret, M. Carleer, C. Clerbaux, P.-F. Coheur, R. Colin, P. DeCola, M. DeMaziere, J. R. Drummond, D. Dufour, W. F. J. Evans, H. Fast, D. Fussen, K. Gilbert, D. E. Jennings, E. J. Llewellyn, R. P. Lowe, E. Mahieu, J. C. McConnell, M. McHugh, S. D. McLeod, R. Michaud, C. Midwinter, R. Nassar, F. Nichitiu, C. Nowlan, C. P. Rinsland, Y. J. Rochon, N. Rowlands, K. Semeniuk, P. Simon, R. Skelton, J. J. Sloan, M.-A. Soucy, K. Strong, P. Tremblay, D. Turnbull, K. A. Walker, I. Walkty, D. A. Wardle, V. Wehrle, R. Zander, and J. Zou. Atmospheric Chemistry Experiment (ACE): Mission overview. *Geophys. Res. Letters*, 32:L15S01, 2005. doi: 10.1029/2005GL022386. 4.4

P.F. Bernath. The atmospheric chemistry experiment (ACE). *JQSRT*, 186:3 – 16, 2017. doi: 10.1016/j.jqsrt.2016.04.006. 4.4

Ronald F. Boisvert, Ronald Cools, and Bo Einarsson. Assessment of accuracy and reliability. In Bo Einarsson, editor, *Accuracy and Reliability in Scientific Computing*, chapter 2. SIAM, Philadelphia, PA, 2005. 5

S.A. Buehler, P. Eriksson, T. Kuhn, A. von Engeln, and C. Verdes. ARTS, the atmospheric radiative transfer simulator. *JQSRT*, 91:65–93, 2005. doi: 10.1016/j.jqsrt.2004.05.051. 1, 7

A. Calder, J. Dursi, B. Fryxell, T. Plewa, G. Weirs, T. Dupont, H. Robey, J. Kane, B. Remington, F. Timmes, G. Dimonte, J. Hayes, M. Zingale, P. Drake, P. Ricker, J. Stone, and K. Olson. Validating astrophysical simulation codes. *Computing in Science & Eng.*, 6(5):10–20, 2004. doi: 10.1109/MCSE.2004.44. 4

S.A. Clough and F.X. Kneizys. Convolution algorithm for the Lorentz function. *Appl. Opt.*, 18:2329–2333, 1979. doi: 10.1364/AO.18.002329. 3.1

S.A. Clough, F.X. Kneizys, G.P. Anderson, E.P. Shettle, J.H. Chetwynd, L.W. Abreu, L.A. Hall, and R.D. Worsham. FASCOD3: spectral simulation. In J. Lenoble and J.F. Geleyn, editors, *IRS'88: Current Problems in Atmospheric Radiation*, pages 372–375. A. Deepak Publishing, 1988. 1, 2.2.2, 3.1, 6

S.A. Clough, M.W. Shephard, E.J. Mlawer, J.S. Delamere, M.J. Iacono, K. Cady-Pereira, S. Boukabara, and P.D. Brown. Atmospheric radiative transfer modeling: a summary of the AER codes. *JQSRT*, 91(2):233–244, 2005. doi: 10.1016/j.jqsrt.2004.05.058. 2.2.2

P.F. Dubois and T.-Y. Yang. Extending Python with Fortran. *Computing in Science & Eng.*, 1(5):66–73, 1999. doi: 10.1109/5992.790589. 5

D.P. Edwards. Atmospheric transmittance and radiance calculations using line–by–line computer models. In *Modelling of the Atmosphere*, volume 928, pages 94–116. Proc. SPIE, 1988. 1, 3.1

Bo Einarsson, Ronald Boisvert, Françoise Chaitin-Chatelin, Ronald Cools, Craig Douglas, Kenneth Dritz, Wayne Enright, William Gropp, Sven Hammarling, Hans Petter Langtangen, Roldan Pozo, Siegfried Rump, Van Snyder, Elisabeth Traviesas-Cassan, Mladen Vouk, William Walster, and Brian Wichmann. *Accuracy and Reliability in Scientific Computing.* SIAM, Philadelphia, PA, 2005. 4

C. Emde, R. Buras-Schnell, A. Kylling, B. Mayer, J. Gasteiger, U. Hamann, J. Kylling, B. Richter, C. Pause, T. Dowling, and L. Bugliaro. The libRadtran software package for radiative transfer calculations (version 2.0.1). *Geosci. Model Dev.*, 9(5):1647–1672, 2016. doi: 10.5194/gmd-9-1647-2016. D

B.A. Fomin. Effective interpolation technique for line–by–line calculation of radiation absorption in gases. *JQSRT*, 53:663–669, 1995. doi: 10.1016/0022-4073(95)00029-K. 3.1

L. Garand, D.S. Turner, M. Larocque, J. Bates, S. Boukabara, P. Brunel, F. Chevallier, G. Deblonde, R. Engelen, M. Hollingshead, D. Jackson, G. Jedlovec, J. Joiner, T. Kleespies, D.S. McKague, L.M. McMillen, J.-.L. Moncet, J.R. Pardo, P.J. Rayer, E. Salathé, R. Saunders, N.A. Scott, P. Van Delst, and H. Woolf. Radiance and Jacobian intercomparison of radiative transfer models applied to HIRS and AMSU channels. *J. Geophys. Res.*, 106(D20):24017–24031, 2001. doi: 10.1029/2000JD000184. 4.3

R.M. Goody and Y.L. Yung. *Atmospheric Radiation — Theoretical Basis.* Oxford University Press, second edition, 1989. 2.1

R. Hughes, P. Bernath, and C. Boone. ACE infrared spectral atlases of the Earth's atmosphere. *JQSRT*, 148:18 – 21, 2014. doi: 10.1016/j.jqsrt.2014.06.016. 4.4

A.K. Hui, B.H. Armstrong, and A.A. Wray. Rapid computation of the Voigt and complex error functions. *JQSRT*, 19:509–516, 1978. doi: 10.1016/0022-4073(78)90019-5. 3.2, 3.2

J. Humlíček. An efficient method for evaluation of the complex probability function: the Voigt function and its derivatives. *JQSRT*, 21:309–313, 1979. doi: 10.1016/0022-4073(79)90062-1. 3.2

J. Humlíček. Optimized computation of the Voigt and complex probability function. *JQSRT*, 27:437–444, 1982. doi: 10.1016/0022-4073(82)90078-4. 3.2, 3.2

K. Imai, M. Suzuki, and C. Takahashi. Evaluation of Voigt algorithms for the ISS/JEM/ SMILES L2 data processing system. *Adv. Space Res.*, 45:669–675, 2010. doi: 10.1016/j.asr.2009.11.005. 3.2

N. Jacquinet-Husson, N.A. Scott, A. Chedin, L. Crepeau, R. Armante, V. Capelle, J. Orphal, A. Coustenis, C. Boonne, N. Poulet-Crovisier, A. Barbe, M. Birk, L.R. Brown, C. Camy-Peyret, C. Claveau, K. Chance, N. Christidis, C. Clerbaux, P.F. Coheur, V. Dana, L. Daumont, M.R. De Backer-Barilly, G. Di Lonardo, J.M. Flaud, A. Goldman, A. Hamdouni, M. Hess, M.D. Hurley, D. Jacquemart, I. Kleiner, P. Köpke, J.Y. Mandin, S. Massie, S. Mikhailenko, V. Nemtchinov, A. Nikitin, D. Newnham, A. Perrin, V.I. Perevalov, S. Pinnock, L. Regalia-Jarlot, C.P. Rinsland, A. Rublev, F. Schreier, L. Schult, K.M. Smith, S.A. Tashkun, J.L. Teffo, R.A. Toth, Vl.G. Tyuterev, J. Vander Auwera, P. Varanasi, and G. Wagner. The GEISA spectroscopic database: Current and future archive for Earth and planetary atmosphere studies. *JQSRT*, 109:1043–1059, 2008. doi: 10.1016/j.jqsrt.2007.12.015. 3.1

M. Kuntz. A new implementation of the Humlicek algorithm for the calculation of the Voigt profile function. *JQSRT*, 57:819–824, 1997. doi: 10.1016/S0022-4073(96)00162-8. 3.2

Hans Petter Langtangen. *Python Scripting for Computational Science*, volume 3 of *Texts in Computational Science and Engineering*. Springer, 2004. 1, 5

Johnny Wei-Bing Lin. Why Python is the next wave in earth sciences computing. *Bull. Am. Met. Soc.*, 93 (12):1823–1824, 2012. doi: 10.1175/BAMS-D-12-00148.1. 1

Kuo-Nan Liou. *An Introduction to Atmospheric Radiation*. Academic Press, Orlando, 1980. 2.1

B. Mayer and A. Kylling. Technical note: The libradtran software package for radiative transfer calculations — description and examples of use. *Atm. Chem. Phys.*, 5(7):1855–1877, 2005. doi: 10.5194/acp-5-1855-2005. D

C. Melsheimer, C. Verdes, S.A. Buehler, C. Emde, P. Eriksson, D.G. Feist, S. Ichizawa, V.O. John, Y. Kasai, G. Kopp, N. Koulev, T. Kuhn, O. Lemke, S. Ochiai, F. Schreier, T.R. Sreerekha, M. Suzuki, C. Takahashi, S. Tsujimaru, and J. Urban. Intercomparison of general purpose clear sky atmospheric radiative transfer models for the millimeter/submillimeter spectral range. *Radio Sci.*, 40:RS1007, 2005. doi: 10.1029/2004RS003110. 4.2

R.H. Norton and C.P. Rinsland. ATMOS data processing and science analysis methods. *Appl. Opt.*, 30: 389–400, 1991. doi: 10.1364/AO.30.000389. 2.2.1

J.J. Olivero and R.L. Longbothum. Empirical fits to the Voigt line width: a brief review. *JQSRT*, 17: 233–236, 1977. doi: 10.1016/0022-4073(77)90161-3. 2.2.4

J.H. Park, L.S. Rothman, C.P. Rinsland, D.J. Richardson, and J.S. Namkung. Atlas of absorption lines from 0 to 17900 cm^{-1}. Reference Publication 1188, NASA, September 1987. 6.3.3

H.M. Pickett, R.L. Poynter, E.A. Cohen, M.L. Delitsky, J.C. Pearson, and H.S.P. Müller. Submillimeter, millimeter, and microwave spectral line catalog. *JQSRT*, 60:883–890, 1998. doi: 10.1016/S0022-4073(98)00091-0. 3.1

L.S. Rothman, R.R. Gamache, A. Goldman, L.R. Brown, R.A. Toth, H.M. Pickett, P.L. Poynter, J.-M. Flaud, C. Camy-Peyret, A. Barbe, N. Husson, C.P. Rinsland, and M.A.H. Smith. The HITRAN database: 1986 edition. *Appl. Opt.*, 26:4058, 1987. 2.2.2

L.S. Rothman, I.E. Gordon, Y. Babikov, A. Barbe, D. Chris Benner, P.F. Bernath, M. Birk, L. Bizzocchi, V. Boudon, L.R. Brown, A. Campargue, K. Chance, E.A. Cohen, L.H. Coudert, V.M. Devi, B.J. Drouin, A. Fayt, J.-M. Flaud, R.R. Gamache, J.J. Harrison, J.-M. Hartmann, C. Hill, J.T. Hodges, D. Jacquemart, A. Jolly, J. Lamouroux, R.J. Le Roy, G. Li, D.A. Long, O.M. Lyulin, C.J. Mackie, S.T. Massie, S. Mikhailenko, H.S.P. Müller, O.V. Naumenko, A.V. Nikitin, J. Orphal, V. Perevalov, A. Perrin, E.R. Polovtseva, C. Richard, M.A.H. Smith, E. Starikova, K. Sung, S. Tashkun, J. Tennyson, G.C. Toon, Vl.G. Tyuterev, and G. Wagner. The HITRAN2012 molecular spectroscopic database. *JQSRT*, 130:4–50, 2013. doi: 10.1016/j.jqsrt.2013.07.002. 3.1

F. Schreier. The Voigt and complex error function: A comparison of computational methods. *JQSRT*, 48: 743–762, 1992. doi: 10.1016/0022-4073(92)90139-U. 3.2

F. Schreier. Optimized evaluation of a large sum of functions using a three-grid approach. *Comp. Phys. Comm.*, 174:783–802, 2006. doi: 10.1016/j.cpc.2005.12.015. 3.3

F. Schreier. Optimized implementations of rational approximations for the Voigt and complex error function. *JQSRT*, 112(6):1010–1025, 2011. doi: 10.1016/j.jqsrt.2010.12.010. 3.2

F. Schreier, S. Gimeno García, P. Hedelt, M. Hess, J. Mendrok, M. Vasquez, and J. Xu. GARLIC – a general purpose atmospheric radiative transfer line-by-line infrared-microwave code: Implementation and evaluation. *JQSRT*, 137:29–50, 2014. doi: 10.1016/j.jqsrt.2013.11.018. 1, 3, 4

F. Schreier, M. Milz, S.A. Buehler, and T. von Clarmann. Intercomparison of three microwave/infrared high resolution line-by-line radiative transfer codes. *JQSRT*, 211:64–77, 2018a. doi: 10.1016/j.jqsrt.2018.02.032. 4.3

F. Schreier, S. Städt, P. Hedelt, and M. Godolt. Transmission spectroscopy with the ACE-FTS infrared spectral atlas of Earth: A model validation and feasibility study. *Molec. Astrophysics*, 11:1–22, 2018b. doi: 10.1016/j.molap.2018.02.001. 4.4

L. Sparks. Efficient line–by–line calculation of absorption coefficients to high numerical accuracy. *JQSRT*, 57:631–650, 1997. doi: 10.1016/S0022-4073(96)00154-9. 3.1

K. Stamnes, S-Chee Tsay, W. Wiscombe, and K. Jayaweera. Numerically stable algorithm for discrete–ordinate–method radiative transfer in multiple scattering and emitting layered media. *Appl. Opt.*, 27: 2502–2509, 1988. doi: 10.1364/AO.27.002502. D

G.P. Stiller, T. von Clarmann, B. Funke, N. Glatthor, F. Hase, M. Höpfner, and A. Linden. Sensitivity of trace gas abundances retrievals from infrared limb emission spectra to simplifying approximations in radiative transfer modelling. *JQSRT*, 72:249–280, 2002. doi: 10.1016/S0022-4073(01)00123-6. 1, 6

A. Uchiyama. Line–by–line computation of the atmospheric absorption spectrum using the decomposed Voigt line shape. *JQSRT*, 47:521–532, 1992. 3.1

T. von Clarmann, M. Höpfner, B. Funke, M. López-Puertas, A. Dudhia, V. Jay, F. Schreier, M. Ridolfi, S. Ceccherini, B.J. Kerridge, J. Reburn, and R. Siddans. Modeling of atmospheric mid–infrared radiative transfer: The AMIL2DA algorithm intercomparison experiment. *JQSRT*, 78:381–407, 2002. doi: 10.1016/S0022-4073(02)00262-5. 4.1

J.A.C. Weideman. Computation of the complex error function. *SIAM J. Num. Anal.*, 31:1497–1518, 1994. doi: 10.1137/0731077. 3.2, 3.2

W. Zdunkowski, T. Trautmann, and A. Bott. *Radiation in the Atmosphere — A Course in Theoretical Meteorology.* Cambridge University Press, 2007. 2.1