libRadtran

library for radiative transfer calculations
Edition 1.0 for libRadtran version 1.1-beta
December 2005

Arve Kylling and Bernhard Mayer

Preface

libRadtran is a library of radiative transfer routines and programs. It has evolved from the `uvspec` radiative transfer model. If you are not familiar with `uvspec`, please note the following: The central program of the libRadtran package is an executable called `uvspec` which can be found in the '`tools`' directory. If you are interested in a user-friendly program for radiative transfer calculations, this is the main information you need to know. A description of `uvspec` is provided in the first part of this manual, and examples including various input files for different atmospheric conditions are provided in the examples directory.

The '`tools`' directory also provides related utilities, like e.g. a mie program (`mie`), some utilities for the calculation of the position of the sun (`zenith`, `noon`, `sza2time`), a few tools for interpolation, convolution, and integration (`spline`, `conv`, `integrate`), and some other small tools.

The second part of the manual (which is only available in the developer manual) describes library routines which might be of interest if you want to write your own programs. The documentation is far from complete at present. Available are routines to read ASCII files, to do interpolations, integrations, convolutions, for Mie theory, to calculate the position of the sun, and some other stuff. More might be available somewhen in the future.

Please note that this document is by no means complete. It is under rapid development and major changes will take place.

Acknowledgements

Many people have already contributed to libRadtran's development. In addition to Arve Kylling (`arve.kylling@nilu.no`) and Bernhard Mayer (`bernhard.mayer@dlr.de`), the following people have contributed to libRadtran or helped out in various other ways.

- The `disort` solver was developed by Knut Stamnes, Warren Wiscombe, S.C. Tsay, and K. Jayaweera
- Warren Wiscombe provided the Mie code `MIEV0`, and the routines to calculate the refractive indices of water and ice, `REFWAT` and `ICEWAT`.
- Seiji Kato (`kato@aerosol.larc.nasa.gov`) provided the correlated-k tables described in Kato et al. (1999).
- Tom Charlock (`t.p.charlock@larc.nasa.gov`), Quiang Fu (`qfu@atm.dal.ca`), and Fred Rose (`f.g.rose@larc.nasa.gov`) provided the most recent version of the Fu and Liou code.
- David Kratz (`kratz@aquila.larc.nasa.gov`) provided the routines for the simulation of the AVHRR channels described in Kratz (1995).
- Frank Evans (`evans@nit.colorado.edu`) provided the `polradtran` solver.
- Ola Engelsen provided data and support for different ozone absorption cross sections.
- Albano Gonzales (`aglezf@ull.es`) included the Yang et al. (2000), Key et al. (2002) ice crystal parameterization.
- Tables for the radiative properties of ice clouds for different particle "habits" were obtained from Jeff Key and Ping Yang, Yang et al. (2000), Key et al. (2002). In

addition, Ping Yang and Heli Wei kindly provided a comprehensive database of particle single scattering properties which we used to derive a consistent set of ice cloud optical properties for the spectral range 0.2 - 100 micron following the detailed description in Key et al. (2002).

- Paul Ricchiazzi (paul@icess.ucsb.edu) and colleagues allowed us to include the complete gas absorption parameterization of their model SBDART into uvspec.
- Luca Bugliaro (luca.bugliaro@dlr.de) wrote the analytical TZS solver (thermal, zero scattering).
- Many unnamed users helped to improve the code by identifying or fixing bugs in the code.

— The Detailed Node Listing —

1 A Brief Overview of libRadtran

This manual documents how to install and use libRadtran and corresponds to libRadtran version 1.1-beta.

libRadtran is a collection of C and Fortran functions and subroutines useful for radiative transfer calculations in the Earth's atmosphere. In addition, programs in C, Fortran and Perl using the libRadtran functions and subroutines are included to allow the user to develop his/her own programs.

It is expected that the reader is familiar with radiative transfer terminology. In addition, a variety of techniques and parameterizations from various sources are used. For more information about the usefulness and applicability of these methods in certain contexts, the user is referred to the referenced literature.

1.1 Radiative transfer calculations

The central program of the libRadtran package is called `uvspec`. It was originally designed to calculate spectral irradiance in the ultraviolet and visible spectral ranges, hence the name. Over the years, `usvpec` has evolved to become a tool for many applications, including the simulation of instruments, the calculation of the radiation budget of the Earth, or the development of remote sensing techniques. `uvspec` is driven with a human-readable input file which allows the definition of the model input in a user-friendly way. Various commands are available to specify the properties of the atmosphere, including Rayleigh scattering, molecular absorbtion, aerosols, water and ice clouds, and the surface albedo. A selection of several radiative transfer solvers is available to simulate different aspects of the radiation field, including the `disort` solver and a pseudo-spherical version of this code, a fast twostream code, Frank Evans' `polRadtran`, and the three-dimensional `MYSTIC` code.

Radiative transfer calculations with `uvspec` are straightforward. The input to the radiative transfer solver is specified in the '`input_file`'. Output is written to `stdout` and can easily be re-directed into an '`output_file`':

```
uvspec < input_file > output_file
```

The syntax of input and output is described in Section 2.1 [uvspec], page 5. Examples of `uvspec` input and output files are found in the '`examples`' directory.

1.2 Ozone retrieval from global irradiance measurements

Stamnes et al. (1991) described a method to derive the total ozone column from global irradiance measurements. The method is based on the comparison of measured irradiance ratios at two wavelengths in the UV part of spectrum with a synthetic chart of this ratio computed for a variety of ozone values. One of the wavelengths should be appreciably absorbed by ozone compared with the other. Typically choices are 305 and 340 nm. The method is reliable under cloudfree conditions, but increasingly overestimates the ozone column for optically thicker clouds, Mayer et al. (1998).

Within libRadtran the method is implemented with two pieces of software

`Gen_o3_tab`
: Generates a look–up–table of ozone values as a function of the irradiance ratio for a given pair of wavelengths and solar zenith angle.

`read_o3_tab`
: Takes as input the measured ratio and solar zenith angle, reads the look–up–table, and returns an ozone value.

The use of these tools is described in Section 2.9 [Geno3tab], page 52.

1.3 Cloud optical thickness from global irradiance measurements

Stamnes et al. (1991) described a method to derive a representative cloud optical depth from global irradiance measurements. The method compares the measured irradiance at a wavelength were ozone absorption is minimal to irradiances generated by a radiative transfer model as a function of cloud optical thickness. The method is very sensitive to the ground albedo and independent measurement of the albedo are needed when there is snow on the surface.

Within libRadtran the method is implemented with two pieces of software

`Gen_wc_tab`
: Generates a look–up–table of water cloud optical depths for various solar zenith angles for a given wavelength.

`read_o3_tab`
: Takes as input the measured irradiance and solar zenith angle, reads the look–up–table, and returns a cloud optical depth.

The use of these tools is described in Section 2.10 [Genwctab], page 53.

1.4 ... and much more

Apart from the above described programs, the '`tools`' directory contains related utilities, like e.g. a mie program (`mie`), a tool to calculate wavelength-dependent cloud properties (`cldprp`), a simple shell script to add levels to existing profiles (`addlevel`), some utilities for the calculation of the position of the sun (`zenith`, `noon`, `sza2time`), a few tools for interpolation, convolution, and integration (`spline`, `conv`, `integrate`), and some other small utilities. Generally, these programs will give some information about their purpose and usage when called without arguments.

2 Some useful tools

2.1 uvspec

`uvspec` calculates the radiation field in the Earth's atmosphere for a given set of input parameters. It reads input from standard input, and outputs to standard output. It is normally invoked in the following way:

```
uvspec < input_file > output_file
```

The format of the input and output files are described below. Several realistic examples of input files are subsequently given.

uvspec may produce a wealth of diagnostic messages and warnings, depending on your use of `verbose` or `quiet`. Diagnostics, error messages, and warnings are written to `stderr` while the uvspec output is written to `stdout`. To make use of this extra information, you may want to write the standard uvspec output to one file and the diagnostic messages to another. To do so, try `(./uvspec < uvspec.inp > uvspec.out) >& verbose.txt`. The irradiances and radiances will be written to '`uvspec.out`' while all diagnostic messages go into '`verbose.txt`'. This method can also be used to collect uvspec error messages.

Warning: Please note the error checking on input variables is not complete at the moment. Hence, if you provide erroneous input, the outcome is unpredictable.

2.1.1 The uvspec input file

uvspec is controlled in a user-friendly way. The control options are named in a (hopefully) intuitive way. In the following, several examples are given, how to create an input file, how to define a cloudless sky atmosphere, how to add aerosols and clouds, etc. All following examples are taken from the libRadtran examples directory and are part of the uvspec self-check. For a complete listing and explanation of all input options, have a look at section Section 2.1.3 [Complete description of input parameters], page 21.

The uvspec input file consists of single line entries, each making up a complete input to the uvspec program. First on the line comes the parameter name, followed by one or more parameter values. The parameter name and the parameter values are separated by white space. Filenames are entered without any surrounding single or double quotes. Comments are introduced by a `#`. Blank lines are ignored.

2.1.1.1 Cloudless, aerosol-free atmosphere

The simplest possible input file contains only a few lines:

```
                          # Location of atmospheric profile file.
atmosphere_file ../data/atmmod/afglus.dat

                          # Location of the extraterrestrial spectrum
solar_file ../data/solar_flux/atlas_plus_modtran

wavelength 310.0 310.0    # Wavelength range [nm]
```

The first three statements define the location of some data files: the uvspec data path (`data_files_path`) where all internal files are expected, the atmospheric profile (`atmosphere_file`), and the extraterrestrial spectrum (`solar_file`). The last line defines the desired wavelength range which is a monochromatic data point in this example. All other data which are not explicitely mentioned assume a default value which is "0" in most cases. Here, the solar zenith angle is 0, the surface albedo is 0, and the atmosphere does not contain clouds nor aerosols. Pressure, temperature, ozone concentration, etc. are read from `atmosphere_file`.

An example of a more complete input file for a clear sky atmosphere is

```
                          # Location of atmospheric profile file.
atmosphere_file ../data/atmmod/afglus.dat
                          # Location of the extraterrestrial spectrum
solar_file ../data/solar_flux/atlas_plus_modtran
ozone_column  300.        # Scale ozone column to 300.0 DU
day_of_year 170           # Correct for Earth-Sun distance
albedo 0.2                # Surface albedo
sza 32.0                  # Solar zenith angle
rte_solver disort         # Radiative transfer equation solver
deltam  on                # delta-M scaling on
nstr  6                   # Number of streams
wavelength 299.0 341.0    # Wavelength range [nm]
slit_function_file ../examples/TRI_SLIT.DAT
                          # Location of slit function
spline 300 340 1          # Interpolate from first to last in step

quiet
```

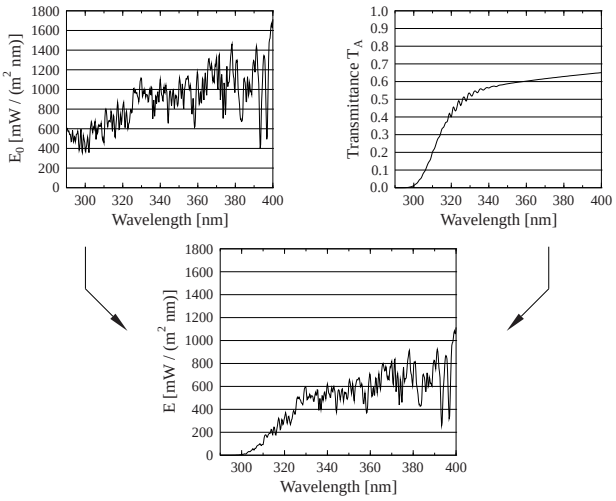
The atmosphere model, i.e. pressure, temperature, and ozone concentration profiles are read from `../data/atmmod/afglus.dat`. The extraterrestrial solar flux is read from the file `../data/solar_flux/atlas_plus_modtran`.

A wavelength dependent **surface albedo** may be specified using `albedo_file` instead of `albedo`. Non-Lambertian surface reflectance (BRDF) for vegetation and water may also be defined (please note that these require the use of `rte_solver disort2`. The BRDF of Vegetation is specified using `rpv_rho0`, `rpv_k`, and `rpv_theta`, following the definition of Rahman et al. (1993). Wavelength-dependent BRDF for vegetation can be defined with `rpv_file`. The BRDF of water surfaces is parameterized following Cox and Munk (1954a,

1954b) and Nakajima (1983). The respective parameters are the wind speed `cox_and_munk_u10`, the pigment concentration `cox_and_munk_pcl`, and the salinity `cox_and_munk_sal`. A complete description of these parameters is given in section Section 2.1.3 [Complete description of input parameters], page 21.

It is helpful to know some details about the **input/output wavelength resolution** in uvspec and how it can be influenced by the user. Basically there are three independent wavelength grids, the **input grid**, the **internal grid**, and the **output grid**. The essential thing to know is that the internal grid is chosen by uvspec itself in a reasonable way, if not explicitely defined in the input file with `transmittance_wl_file` or `molecular_tau_file`. The output grid is completely independent of the internal grid and is entirely defined by the `solar_file`. The wavelength grid of all other input data (e.g. albedo, optical properties of aerosols and clouds, etc) is also completely independent. These data are automatically interpolated to the resolution of the internal wavelength grid. Hence, only two constraints are set to the gridding of the input data: (1), the wavelength range has to cover all internal grid points; and (2), it should be chosen in a reasonable manner to allow reasonable interpolation (which essentially means, dense enough).

In the ultraviolet/visible, uvspec uses an internal grid with a step with of 0.5nm below 350nm and 1nm above 350nm. This is a conservative choice which fully resolves the broad ozone absorption bands and the slowly varying Rayleigh, aerosol, and cloud extinctions. The idea is outlined in the following figure which is taken from Mayer et al. (1997):



The transmittance (or reflectance) is calculated on a moderate resolution grid which reduces the number of calls to the `rte_solver` and hence the computational time. Then, the transmittance is interpolated to the wavelengths in the `solar_file` (which is usually defined with higher spectral resolution), multiplied with the extraterrestrial irradiance, and possibly post-processed. Hence, the wavelength in the output spectrum are those contained in the `solar_file` which has two important implications: (1) Only those wavelengths are output that are contained in the `solar_file`. If e.g. a monochromatic calculation is defined by setting '`wavelength 327.14 327.14`', there will only be output if the wavelength 327.14 is explicitely listed in `solar_file`; (2) this is also true at thermal wavelengths where the extraterrestrial irradiance is zero; hence, even for a calculation in the thermal range a `solar_`

`file` can be specified which defines the output grid in the first column and arbitrary values in the second column. Keeping these points in mind, `solar_file` is a convenient way to define an arbitrary output grid. `solar_file` may be omitted for thermal radiation calculations (`source thermal`) as well as for `transmittance` and `reflectivity` calculations. If omitted, the output grid equals the internal wavelength grid.

If required, a **user-defined internal grid** can be specified with `transmittance_wl_file` or `molecular_tau_file`. Note that this is a way to speed up the calculation considerably. E.g., for some applications the internal grid in the UV-A and visible can be set to 10nm which would reduce computational times by up to a factor of 10.

Things are completely different if one of the `correlated_k` parameterizations is selected (see below). In this case all flexibility is taken away from the user which is an inherent feature of the k distribution method. Internal grid as well as the extraterrestrial file are in this case defined by the choice of the parameterization itself.

2.1.1.2 Spectral resolution

`uvspec` offers four different ways of spectral calculations:

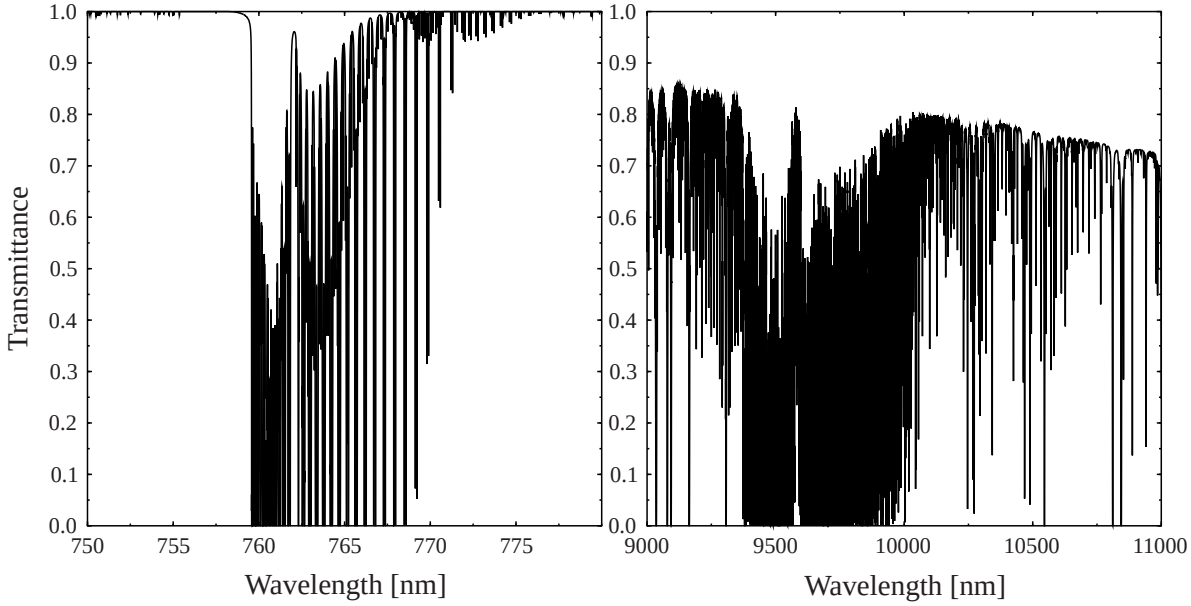
1. **Spectrally resolved calculation** in the UV and visible spectral ranges;
2. **Line-by-line calculation** with user-defined molecular absorption data;
3. **The correlated-k method**.
4. **Pseudo-spectral calculation** with exponential-sum-fit, from LOWTRAN; code adopted from SBDART (Ricchiazzi et al., 1998);

The choice of the method is determined by the problem and the decision is therefore entirely up to the user. The spectrally resolved calculation and the line-by-line calculation are more or less exact methods while the correlated-k distribution and the pseudo-spectral calculation are approximations that provide a compromise between speed and accuracy. In the following it is briefly described which method fits which purpose:

A **spectrally resolved calculation** is the most straightforward way, and will be the choice for all users interested in the ultraviolet and visible spectral ranges. In the UV/vis gas absorption generally occurs in broad bands with only slow spectral variation, the most important of these being the Hartley, Huggins, and Chappuis bands of ozone. Hence, a radiative transfer calculation every 1nm usually is sufficient to fully resolve any spectral variation using the method described in the last section. Absorption cross sections for various species are included, among them the most important O3 and NO2.

In the infrared, however, molecular absorption spectra are characterized by thousands of narrow absorption lines. There are two ways to treat these, either by highly resolved spectral calculations, so-called **line-by-line** calculations, or by a band parameterization. Concerning line-by-line, uvspec offers the possibility to define a spectrally resolved absorption cross section profile using `molecular_tau_file`. There is no option in libRadtran to generate such a `molecular_tau_file`, because (1) the HITRAN database which forms the basis for such calculations amounts to about 100 MByte which are updated continuously; and (2), there are sophisticated line-by-line programs available, like e.g. `genln2` (Edwards, 1992). Using `genln2` it is straightforward to create the input for uvspec line-by-line calculations. It is also planned to make line-by-line cross sections available for the six standard profiles that

come with libRadtran. The following figure shows an example of a line-by-line calculation of the atmospheric transmittance in two selected solar and thermal spectral ranges, the O2A-absorption band around 760nm and a region within the infrared window around 10 micron.



All spectral lines in the left figure are due to absorption by oxygen, while the ones in the left figure are due to ozone, water vapour, and CO2. Line-by-line is obviously the exact way for radiation calculations. For most applications, however, line-by-line is far too slow. Here one needs a band parameterization, and the most accurate of these is the so-called **correlated-k approximation**. uvspec contains several correlated-k parameterizations which are invoked with `correlated_k`, in particular Kato et al. (1999), Fu and Liou (1992/93), Kratz (1995), as well as the possibility to specify a user-defined one. Kato et al. (1999) is a accurate parameterization for the solar spectral range. `uvspec` contains three different versions:

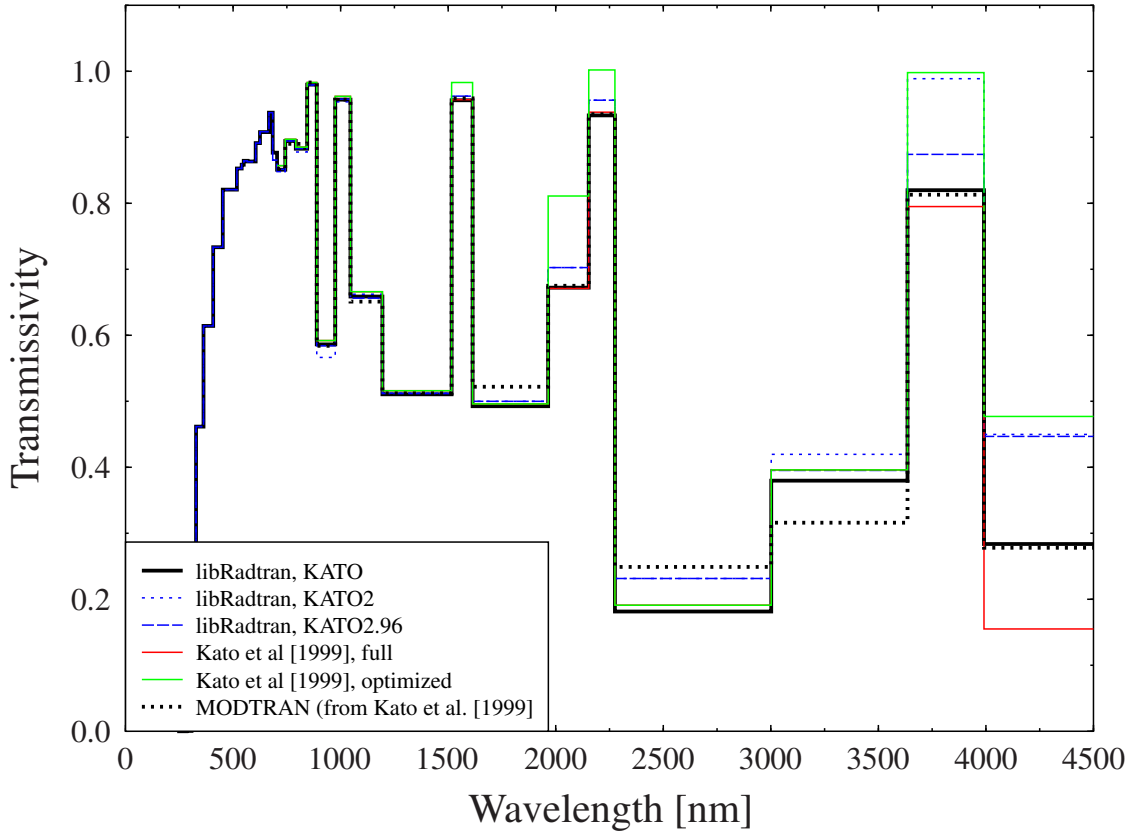
`Kato` The original tables provide by Seiji Kato which should correspond to the full version described in Kato et al. (1999); 575 subbands total, that is, 575 calls to the `rte_solver`

`Kato2` A new, optimized version of the tables, provided by Seiji Kato, 2003, with only 148 subbands (that is, calls to the `rte_solver`); the uncertainty is only slightly higher than `Kato`; the absorption coefficients are based on HITRAN 2000.

`Kato2.96` Similar to `Kato2` but based on HITRAN96.

The following Figure shows a comparison between the three parameterization which are part of libRadtran and the data from Figure 3 by Kato et al. [1999]. It is immediately obvious that the uncertainty is high for all bands above 2.5 micron which is probably due to the treatment of band overlap. For this reasons, the results for the indiviudal bands should not be trusted while the integrated shortwave radiation (the sum of all 32 bands) is calculated with high accuracy because (1) the bands above 2.5 micron contribute only

little to the integrated irradiance; and (2) errors are random and cancel each other to some degree.



For more information on these parameterizations please refer to the mentioned publications. Correlated-k is a powerful way to calculate spectrally integrated quantities, however, it takes away some flexibility. In particular, this means that the wavelength grid is no longer chosen by the user but by the parameterization, that is, by `uvspec`. The `uvspec` output is then no longer spectral quantities, e.g. W / (m2 nm), but integrated over the spectral bands, e.g. W / m2.

A simple but complete example for a correlated-k approximation of the solar spectrum:

```
# Conditions for the calculation of Figure 3 in
# Kato et al., JQSRT 62, 109-121, 1999.
# To compare the data, the direct irradiance calculated
# by uvspec has to be divided by cos(30 deg) because
# Kato et al. plot direct normal irradiance.

                          # Location of atmospheric profile file.
atmosphere_file ../examples/AFGLMS50.DAT
                          # Location of the extraterrestrial spectrum

albedo 0.2                # Surface albedo
sza 30.0                  # Solar zenith angle
rte_solver twostr         # Radiative transfer equation solver

correlated_k KATO         # Correlated-k by Kato et al. [1999]

output sum                # Calculate integrated solar irradiance

quiet
```

Here, the solar spectrum is split up into 32 bands according to Kato et al. (1999). In order to calculate integrated shortwave irradiance, simply sum the outputs, or even simpler, add `output sum` to the input file.

For **pseudo-spectral calculations** in the whole spectral range, we have adopted the molecular absorption parameterization from LOWTRAN/SBDART by Ricchiazzi et al. (1998). The respective section of this paper says:

> SBDART relies on low-resolution band models developed for the LOWTRAN 7 atmospheric trans-mission code (Pierluissi and Peng, 1985). These models provide clear-sky atmospheric transmission from 0 to 50000 cm-1 and include the effects of all radiatively active molecular species found in the earth s atmosphere. The models are derived from detailed line-by-line calculations that are degraded to 20 cm-1 resolution for use in LOWTRAN. This translates to a wavelength resolution of about 5 nm in the visible and about 200 nm in the thermal infrared. These band models represent rather large wavelength bands, and the transmission functions do not necessarily follow Beers Law. This means that the fractional transmission through a slab of material depends not only on the slab thickness, but also on the amount of material penetrated before entering the slab. Since the radiative transfer equation solved by SBDART assumes Beers Law behavior, it is necessary to express the transmission as the sum of several exponential functions (Wiscombe and Evans, 1977). SBDART uses a three-term exponential fit, which was also obtained from LOWTRAN 7. Each term in the exponential fit implies a separate solution of the radiation transfer equation. Hence, the RT equation solver only needs to be invoked three times for each spectral increment. This is a great computational economy compared to a higher order fitting polynomial, but it may also be a source of significant error.

The LOWTRAN/SBDART gas parameterization is invoked with `correlated_k LOWTRAN`. The spectral resolution may be arbitrarily chosen by the user. If not explicitely

defined with `transmittance_wl_file`, an internal grid with a step width of 0.5nm below 350nm and 1nm above 350nm is chosen which is practically overkill for most applications in the infrared. An extraterrestrial spectrum covering the complete solar range is provided at two different resolutions, `data/solar_flux/kurudz_1.0nm.dat` and `data/solar_flux/kurudz_0.1nm.dat`. An example for the solar range is shown in `examples/UVSPEC_SBDART_SOLAR.INP`:

```
atmosphere_file ../data/atmmod/afglus.dat
solar_file ../data/solar_flux/kurudz_1.0nm.dat

albedo 0.2               # Surface albedo
sza 30.0                 # Solar zenith angle

rte_solver twostr        # Radiative transfer equation solver
wavelength 250.0 2500.0  # Wavelength range

correlated_k SBDART      # select SBDART molecular absorption

aerosol_default
aerosol_visibility 20

quiet
```

while `examples/UVSPEC_SBDART_THERMAL.INP` shows how to do a thermal calculation:

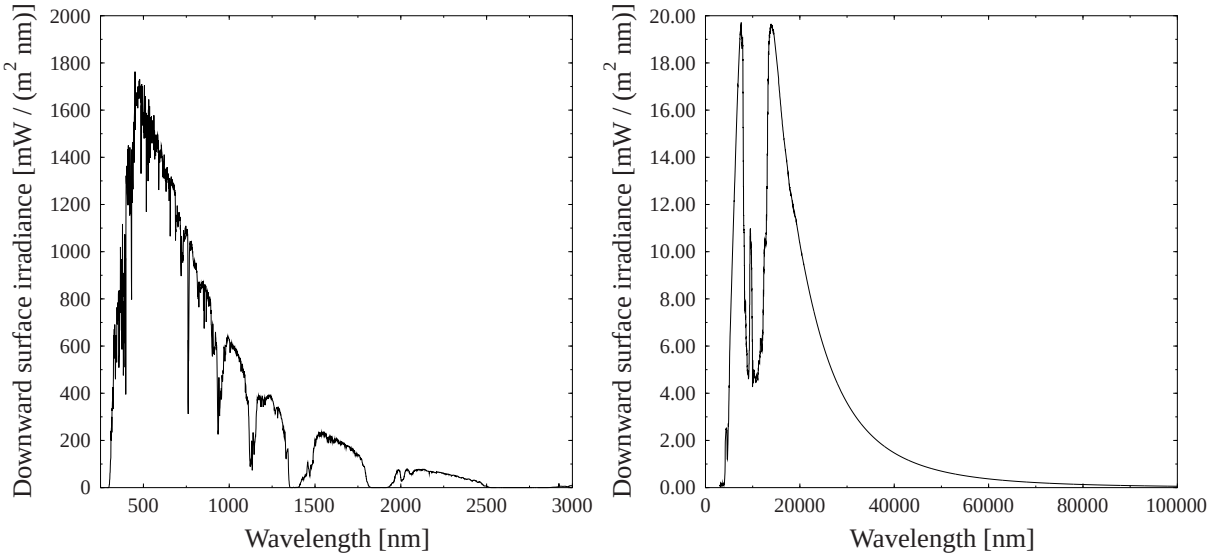
```
# uvspec data files
data_files_path ../data/
atmosphere_file ../examples/AFGLUS.70KM
solar_file ../examples/UVSPEC_SBDART_THERMAL.TRANS

source thermal

rte_solver twostr     # Radiative transfer equation solver
transmittance_wl_file ../examples/UVSPEC_SBDART_THERMAL.TRANS

correlated_k SBDART   # select SBDART molecular absorption

quiet
```



The figure shows the results of the solar and thermal calculations. The water vapour absorption bands in the solar range are clearly visible, as is the absorption window around 10 micron in the thermal. Please note the following points:

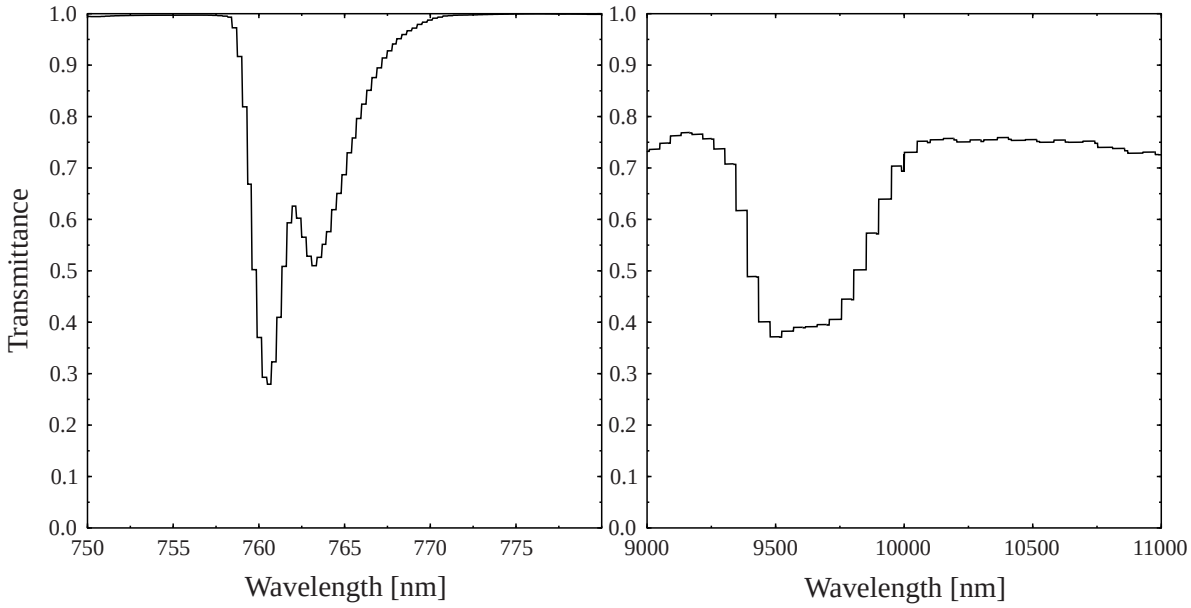
- Thermal radiation is per default output in W/(m2 cm-1), see documentation of `thermal_bandwith` and `thermal_bands_file`. To convert e.g. to W/(m2 nm) multiply with k/lambda where k is the wavenumber [cm-1] and lambda is the wavelength [nm]. To calculate band-integrated thermal quantities please consider `thermal_bands_file`.

- Even though no extraterrestrial irradiance is required, a `solar_file` may be specified for the thermal case. The reason is that, as explained initially, the `solar_file` defines the output grid. The second column in `solar_file` can be chosen arbitrarily in this case because it is ignored.

- For the choice of the wavelength grid for the calculation (`transmittance_wl_grid`) please consider that the resolution of the absorption parameterization is 5cm-1 which converts to 0.3nm at 750nm and to 50nm at 10 micron. Choosing higher resolutions for the internal wavelength grid (`transmittance_wl_file`) is usually a waste of computational time.

- Please also make sure to choose a fine enough spectral resolution in order to capture all absorption features.

The following figure shows two selected wavelength intervals of the solar and thermal range, to demonstrate the spectral resolution of the LOWTRAN/SBDART absorption parameterization.



The resolution is about 5cm-1 which translates to about 0.3nm in the left figure (oxygen A-band) and 50nm in the right figure (ozone absorption band in the atmospheric window). Compare this figure to the above line-by-line example to get an impression about the differences between both methods.

2.1.1.3 Aerosol

All options to set up and modify aerosol properties start with `aerosol_`. Aerosols may be specified in a hierarchical way. The most simple way to define an aerosol is by the command `aerosol_default` which will set up the aerosol model by E.P. Shettle, "Models of aerosols, clouds and precipitation for atmospheric propagation studies", in "Atmospheric propagation in the uv, visible, ir and mm-region and related system aspects", AGARD Conference Proceedings (454), 1989. The default properties are a rural type aerosol in the boundary layer, background aerosol above 2km, spring-summer conditions and a visibility of 50km. These settings may be modified with `aerosol_haze`, `aerosol_vulcan`, `aerosol_season`, and `aerosol_visibility`. More information can be introduced step by step, overwriting the default parameters. `aerosol_tau_file`, `aerosol_ssa_file`, and `aerosol_gg_file`, can be used to define the profiles of optical thickness, single scattering albedo, and asymmetry parameter. The integrated optical thickness can be set to a constant value using `aerosol_set_tau` or scaled with `aerosol_scale_tau`. The single scattering albedo may be scaled by `aerosol_scale_ssa` or set to a constant value by `aerosol_set_ssa`. The aerosol asymmetry factor may be set by `aerosol_set_gg`. The wavelength dependence of the aerosol optical depth is specified using the `aerosol_angstrom` parameter. `aerosol_moments_file` allows specification of the scattering phase function. If microphysical properties are available these may be introduced by defining the complex index of refraction `aerosol_refrac_index` or `aerosol_refrac_file` and the size distribution `aerosol_sizedist_file`. Finally, one may define the aerosol optical properties of each layer explicitely using `aerosol_files`.

The following list is an overview of the aerosol description parameters. The entries are arranged in a way that a parameter 'overwrites' all values higher up in the list.

`aerosol_default`
: Generate default aerosol according to Shettle (1989)

`aerosol_vulcan, aerosol_haze, aerosol_season, aerosol_visibility`
: Set Shettle (1989) aerosol properties (aerosol type, visibility)

`aerosol_files`
: Specify optical properties of each layer explicitly, that is, extinction coefficient, single scattering albedo, and the moments of the phase function (everything as a function of wavelength).

`aerosol_tau_file, aerosol_ssa_file, aerosol_gg_file`
: Overwrite profiles of optical thickness, single scattering albedo, and asymmetry parameter

`aerosol_moments_file`
: Specify a phase function to be used instead of the Henyey-Greenstein phase function

`aerosol_refrac_index, aerosol_refrac file, aerosol_sizedist_file`
: Calculate optical properties from size distribution and index of refraction using Mie theory. Here is an exception from the rule that ALL values defined above are overwritten because the optical thickness profile is re-scaled so that the optical thickness at the first internal wavelength is unchanged. It is done that way to give the user an easy means of scaling the specifying the optical thickness at a given wavelength.

`aerosol_set_gg, aerosol_set_ssa, aerosol_scale_ssa, aerosol_set_tau, aerosol_scale_tau`
: Overwrite profiles of asymmetry parameter and single scattering albedo

`aerosol_angstrom`
: Overwrite the integrated optical thickness (profiles are not changed).

An example for a uvspec aerosol description is

```
include ../examples/UVSPEC_CLEAR.INP

aerosol_vulcan 1          # Aerosol type above 2km
aerosol_haze 6            # Aerosol type below 2km
aerosol_season 1          # Summer season
aerosol_visibility 20.0   # Visibility
aerosol_angstrom 1.1 0.2  # Scale aerosol optical depth
                          # using Angstrom alpha and beta
                          # coefficients
aerosol_scale_ssa 0.85    # Scale the single scattering albedo
                          # for all wavelengths
aerosol_set_gg 0.70       # Set the asymmetry factor
aerosol_tau_file ../examples/AERO_TAU.DAT
                          # File with aerosol optical depth profile
```

By combining this with the clear sky example given above a complete uvspec input file including aerosol is constructed.

2.1.1.4 Water clouds

All options to set up and modify water cloud properties start with `wc_`.

The easiest way to define a water cloud is to specify a `wc_file` which defines the liquid water content and effective droplet radius at each model level. By combining the following lines with the clear sky example given above a complete uvspec input file including water clouds is constructed.

```
include ../examples/UVSPEC_CLEAR.INP

wc_file ../examples/WC.DAT  # Location of water cloud file
wc_set_tau 15.              # Set total water cloud optical depth
```

A typical example for a wc_file looks like:

```
#      z     LWC    R_eff
#     (km)  (g/m^3) (um)
      5.000   0      0
      4.000   0.2    12.0
      3.000   0.1    10.0
      2.000   0.1     8.0
```

The three columns are the level altitude [km], the liquid water content [g/m3], and the effective droplet radius [micron]. Per default, these quantities are defined at the given model levels, that is, the value 0.2 g/m3 refers to altitude 4.0km, as e.g. in a radiosonde profile. The properties of each layer are calculated as average over the adjacent levels. E.g. the single scattering properties for the model layer between 3 and 4km are obtained by averaging over the two levels 3km and 4km. To allow easy definition of sharp cloud boundaries, clouds are only formed if both liquid water contents above and below the respective layer are larger than 0. Hence, in the above example, the layers between 2 and 3 as well as between 3 and 4km are cloudy while those between 1 and 2km and between 4 and 5km are not.

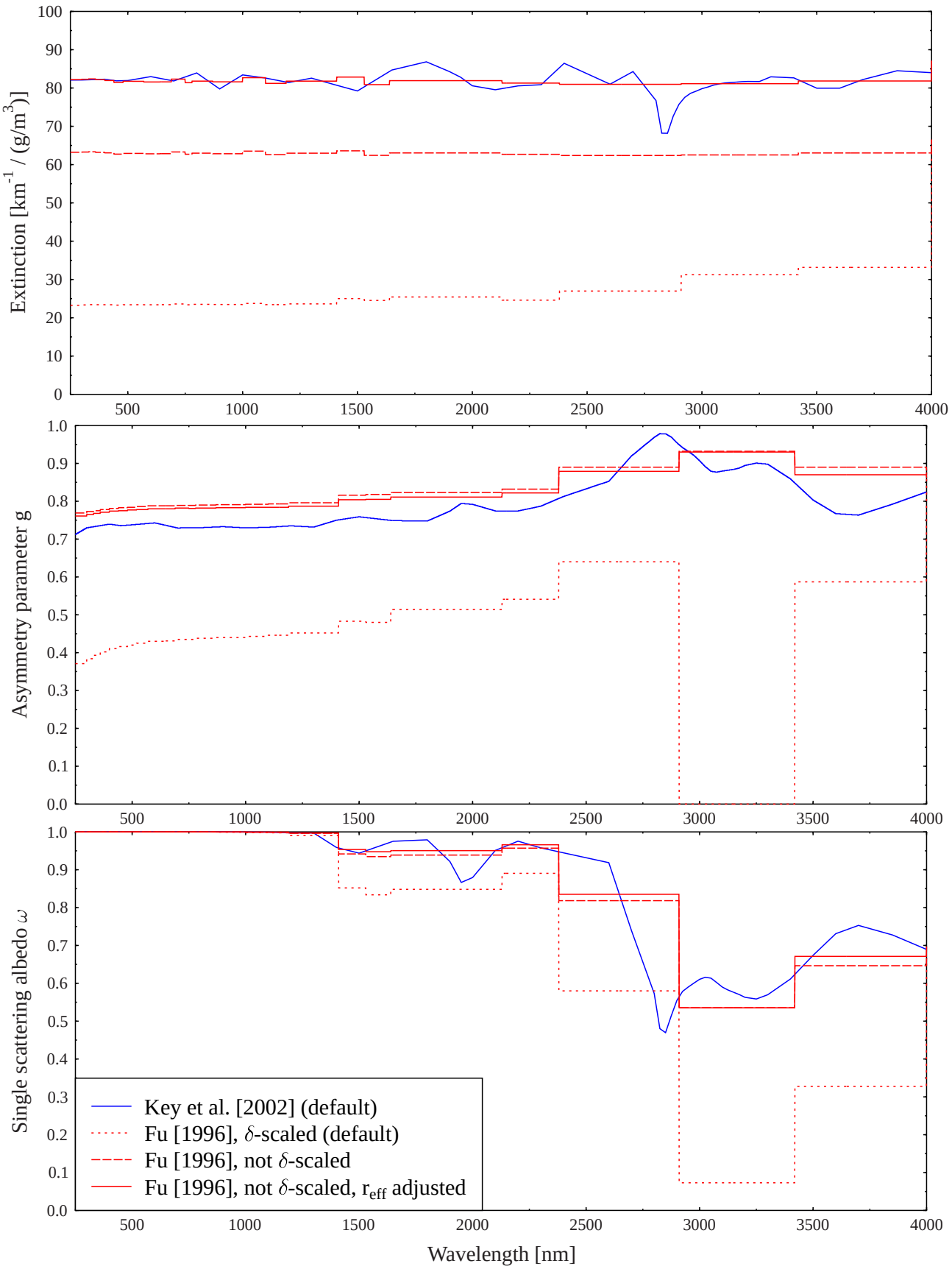
`wc_layer` provides a useful alternative if one prefers to define layer quantities. If specified, the properties are interpreted as layer properties, and in the above example, the cloud would extend from 2 to 5km, with e.g. a LWC of 0.2 g/m3 for the layer between 4 and 5km. To make sure that the clouds really look as you want them to look, it is recommended to use the `verbose` option. This option shows not only where the cloud is actually placed, it rather tells the user exactly how LWC and effective radius are translated into optical properties, depending on the choice of parameterisation. Please also note that the definition of the empty top level at 5km is important to tell uvspec where the cloud ends. If omitted, the cloud would extend all the way to the top of the atmosphere.

There are different ways to convert the microphysical properties to optical properties. Either a parameterization is used, like the one by Hu and Stamnes (1993) (which is the default), or by Mie calculations. The latter are very time-consuming, hence we decided not to include these online into uvspec but rather have an option to read in pre-calculated Mie tables. The option `wc_properties` controls the method: `hu` selects the Hu and Stamnes (1993) parameterization, `mie` selects pre-calculated Mie tables which are available at http://www.libradtran.org. If `wc_properties mie` is selected, the model expects one Mie cloud property file for each internal wavelength which is useful for the fixed wavelength grids used by the correlated-k parameterisations `correlated_k kato`, `correlated_k fu`, etc. For a spectral calculation with free wavelength grid, there is also the possibility to use a pre-defined set of Mie tables (available at the web site) and to defined `wc_properties_interpolate` to automatically interpolate the Mie properties to the internal wavelength grid. Although this is an extremely useful option, please use it careful because it might consume enourmous amounts of memory. Finally, there is the option to define an arbitrary file which is in the format as generated by Frank Evans' `cloudprp` which comes with SHDOM, see http://nit.colorado.edu/~evans/shdom.html.

As for the aerosol, there are several options to modify the optical properties of the clouds. And of course there is also the option of defining all cloud properties explicitely using `wc_files`.

2.1.1.5 Ice clouds

Ice clouds are generated in a similar way to water clouds. All options to set up and modify ice cloud properties start with `ic_`. The main difference between water and ice clouds is that the latter usually consist of non-spherical particles. Hence, the conversion from microphysical to optical properties is much less defined, and several parameterizations are available. Please note in addition that there are different definitions of the effective radius and e.g. the parameterizations by Key et al. (2002) and by Fu (1996) actually use different definitions (see explanation of `ic_properties`). Finally, the sharp forward peak which is typical for ice particles is also treated differently: E.g. Fu (1996) provides delta-scaled optical properties while Key et al. (2002) uses unscaled parameters (see explanation of `ic_fu_tau`). The following figure illustrates the implications. Plotted are extinction coefficient, asymmetry parameter, and single scattering albedo for ice clouds as a function of wavelength. If treated consistently, both Key et al. (2002) and Fu (1996) provide nearly identical results (blue and red solid line). However, Fu (1996) uses a different definition of effective radius and recommends to use the delta-scaled properties (red dotted line). The larger part of the difference compared to the solid line is due to delta-scaling which can be seen by comparing the red dotted (delta-scaled) and red-dashed (not delta-scaled) lines. Lately, a new set of optical properties was derived, combining the Key et al. (2002) data below 3.4 micron with new calculations above 3.4 micron, to be invoked with `ic_properties yang`.


Extinction [km^{-1} / (g/m^3)]
Asymmetry parameter g
Single scattering albedo ω
Wavelength [nm]
Key et al. [2002] (default)
Fu [1996], δ-scaled (default)
Fu [1996], not δ-scaled
Fu [1996], not δ-scaled, r_{eff} adjusted

2.1.1.6 Calculation of radiances

To calculate radiances the following lines will do the job when combined with the clear sky example above

```
include ../examples/UVSPEC_AEROSOL.INP # Include's may be nested.

rte_solver disort2 # This override what is specified in above file
                   # and files included in that file etc.

phi0 40.0                  # Solar azimuth angle
umu  -1.0 -0.5  -0.2 -0.1 # Output cosine of polar angle
phi 0.0 45. 90. 135. 180.0 225. 270.0 # Output azimuth angles
```

More examples with output are found in the 'examples' directory.

2.1.2 The uvspec output

The uvspec output differs for the different solvers. It may be controlled to some degree using output_user.

2.1.2.1 DISORT, SDISORT and SPSDISORT

For the disort, sdisort and spsdisort solvers uvspec outputs one block of data to standard output (stdout) for each wavelength. The format of the block is

```
lambda edir edn eup uavgdir uavgdn uavgup
```

if umu is not specified. If umu is specified the format of the block is

```
lambda edir edn eup uavgdir uavgdn uavgup
umu(0) u0u(umu(0))
umu(1) u0u(umu(1))
  .      .
  .      .
```

If both umu and phi is specified the output format of each block is

```
lambda edir edn eup uavgdir uavgdn uavgup
                            phi(0)      ...         phi(m)
umu(0) u0u(umu(0))   uu(umu(0),phi(0)) ... uu(umu(0),phi(m))
umu(1) u0u(umu(1))   uu(umu(1),phi(0)) ... uu(umu(1),phi(m))
  .      .             .                      .
  .      .             .                      .
umu(n) u0u(umu(n))   uu(umu(n),phi(0)) ... uu(umu(n),phi(m))
```

and so on for each wavelength.

2.1.2.2 TWOSTR

The format of the output line for the twostr solver is

```
lambda edir edn eup uavg
```

for each wavelength.

2.1.2.3 POLRADTRAN

The output from the `polradtran` solver depends on the number of Stokes parameters, `polradtran_nstokes`. For each wavelength the output block is

```
lambda down_flux(1) up_flux(1) ...  down_flux(is) up_flux(is)
```

if `phi` is not specified. Here `is` is the number of Stokes parameters specified by `polradtran_nstokes`. If `phi` and `umu` are specified the block is

```
lambda down_flux(1) up_flux(1) ...  down_flux(is) up_flux(is)
                            phi(0)                  ...                 phi(m)
Stokes vector I
umu(0) u0u(umu(0))   uu(umu(0),phi(0)) ... uu(umu(0),phi(m))
umu(1) u0u(umu(1))   uu(umu(1),phi(0)) ... uu(umu(1),phi(m))
  .     .             .                      .
  .     .             .                      .
umu(n) u0u(umu(n))   uu(umu(n),phi(0)) ... uu(umu(n),phi(m))
Stokes vector Q
  .       .                                    .
  .       .                                    .
```

Note that `polradtran` outputs the total (=direct+diffuse) downward flux. Also note that u0u is always zero for `polradtran`.

2.1.2.4 Description of symbols

In the above output blocks the symbols used have the following meaning.

`cmu` Computational polar angles from `polradtran`.

`down_flux, up_flux`
The total (direct+diffuse) downward (down_flux) and upward (up_flux) irradiances. Same units as extraterrestrial irradiance.

`lambda` Wavelength (nm)

`edir` Direct beam irradiance (same units as extraterrestrial irradiance, e.g mW/(m2 nm) if using the '`atlas3`' spectrum in the '`data/solar_flux`' directory.)

`edn` Diffuse down irradiance, i.e. total minus direct beam (same units as `edir`).

`eup` Diffuse up irradiance (same units as `edir`).

`uavg` The mean intensity. Proportional to the actinic flux: To obtain the actinic flux, multiply the mean intensity by 4 pi (same units as `edir`).

`uavgdir` Direct beam contribution to the mean intensity. (same units as `edir`).

`uavgdn` Diffuse downward radiation contribution to the mean intensity. (same units as `edir`).

`uavgup` Diffuse upward radiation contribution to the mean intensity. (same units as `edir`).

`u0u` The azimuthally averaged intensity at `numu` user specified angles `umu`. (units of e.g. mW/(m2 nm sr) if using the '`atlas3`' spectrum in the '`data/solar_flux`' directory.)

`uu` The radiance (intensity) at `umu` and `phi` user specified angles. (units of e.g. mW/(m2 nm sr) if using the '`atlas3`' spectrum in the '`data/solar_flux`' directory.)

`uu_down, uu_up`
The downwelling and upwelling radiances (intensity) at `cmu` and `phi` angles. (units of e.g. mW/(m2 nm sr) if using the '`atlas3`' spectrum in the '`data/solar_flux`' directory.)

The total downward irradiance is given by

```
irr_down = edir + edn
```

The total mean intensity is given by

```
uavg = uavgdir + uavgdn + uavgup
```

If `deltam` is on it does not make sense to look at the individual contributions to `uavg` since they are delta-M scaled.

2.1.3 Complete description of input parameters

The uvspec input file consists of single line entries, each making up a complete input to the uvspec program. First on the line comes the parameter name, followed by one or more parameter values. The parameter name and the parameter values are separated by white space. Filenames are entered without any surrounding single or double quotes. Comments are introduced by a `#`. Blank lines are ignored. The order of the lines is not important, with one exception: if the same input option is used more than once, the second one will usually over-write the first one.

The various input parameters are described in detail below.

`aerosol_angstrom`
Scale the aerosol optical depth using the Ångström formula. Specify the Ångström alpha and beta coefficients. The optical thickness defined here is the integral from the user-definded `altitude` to TOA (top of atmosphere).

`aerosol_default`
Set up a default aerosol according to Shettle (1989). The default properties are a rural type aerosol in the boundary layer, background aerosol above 2km,

spring-summer conditions and a visibility of 50km. These settings may be modified with `aerosol_haze`, `aerosol_vulcan`, `aerosol_season`, and `aerosol_visibility`.

`aerosol_files`
: A way to specify aerosol optical depth, single scattering albedo, and phase function moments for each layer. The file specified by `aerosol_files` has two columns where column 1 is the altitude in km. The altitudes must be the same as those specified in the `atmosphere_file`. The second column is a the name of a file which defines the optical properties of the layer starting at the given altitude. The files specified in the second column must have the following format:

  `Column 1:` The wavelength in nm. These wavelengths may be different from those in `solar_file`. Optical properties are interpolated to the requested wavelengths.

  `Column 2:` The extinction coefficient of the layer in units km-1.

  `Column 3:` The aerosol single scattering albedo of the layer.

  `Column 4-(nmom+4):`
  The moments of the aerosol phase function.

  For some simple examples see the files '`examples/AERO_*.LAYER`'. Note that if using the `rte_solver disort2` it makes good sense to make the number of moments larger than `nstr`. For `rte_solver disort` and `rte_solver polradtran` the number of moments included in the calculations will be `nstr+1`. Higher order moments will be ignored for these solvers. Please note that the uppermost line of the `aerosol_files` denotes simply the top altitude of the uppermost layer. The optical properties of this line are consequently ignored. There are two options for this line: either an optical property file with zero optical thickness is specified or "NULL" instead.

`aerosol_gg_file`
: Location of aerosol asymmetry parameter file. The file must have two columns. Column 1 is the altitude in km. The altitude grid must be exactly equal to the altitude grid specified in the file `atmosphere_file`. Column 2 is the asymmetry parameter of each layer. At present, the asymmetry parameter defined here is constant with wavelength but it is planned to introduce the wavelength dependence in the near future. Comments start with `#`. Empty lines are ignored.

`aerosol_haze`
: Aerosol type in the lower 2 km of the atmosphere. Integer. See E.P. Shettle, "Models of aerosols, clouds and precipitation for atmospheric propagation studies", in "Atmospheric propagation in the uv, visible, ir and mm-region and related system aspects", AGARD Conference Proceedings (454), 1989.

  `1` Rural type aerosols.

  `4` Maritime type aerosols.

  `5` Urban type aerosols.

`6` Tropospheric type aerosols.

`aerosol_moments_file`
Location of aerosol moments file, a one-column file containing an arbitrary number of Legendre terms of the phase function. The phase function $p(\mu)$ is

$$p(\mu) = \sum_{m=0}^{\infty} (2m+1) \cdot k_m \cdot P_m(\mu)$$

where k_m is the m'th moment and $P_m(\mu)$ is the m'th Legendre polynomial. If not specified, a Henyey-Greenstein phase function is assumed where the asymmetry parameter g is either a default value depending on the aerosol type or may be specified using `aerosol_set_gg`.

`aerosol_refrac_file`
File containing the wavelength-dependent refractive index of the aerosol. Three columns are expected: wavelength [nm] and the real and imaginary parts of the refractive index. Together with `aerosol_sizedist_file` this forms the input to Mie calculations of the aerosol optical properties. Attention: If the aerosol properties are defined using the refractive index and the size distribution, the wavelength dependence of the optical properties is determined by Mie theory. At present there are three ways to define the absolute value of the optical thickness: (1) `visibility` defines the profile at the first *internal* wavelength; for a monochromatic calculation and in correlated-k mode, the first *internal* wavelength equals the first wavelength output by `uvspec`; for spectral calculations, the first wavelength might be a little bit smaller than the first wavelength output by `uvspec`; (2) `aerosol_tau_file` defines the optical thickness profile at the first *internal* wavelength; or (3) absolute optical thickness and wavelength-dependence are defined by `aerosol_angstrom`. In future it is planned to also allow the specification of absolute particle densities, etc.

`aerosol_refrac_index`
Wavelength-independent refractive index of the aerosol; if wavelength-dependence is required, use `aerosol_refrac_file` instead. Together with `aerosol_sizedist_file` this forms the input to Mie calculations of the aerosol optical properties. See comment at `aerosol_refrac_file`.

`aerosol_scale_ssa`
Scale the aerosol single scattering albedo for all wavelengths and altitudes with a positive number. If the resulting scaled single scattering albedo is larger than 1 it is set to 1.

`aerosol_scale_tau`
Scale the aerosol extinction for all wavelengths and altitudes with a positive number.

`aerosol_set_gg`
Set the aerosol asymmetry parameter for all wavelengths and altitudes to a constant value between -1.0 and 1.0.

`aerosol_set_ssa`
: Set the aerosol single scattering albedo for all wavelengths and altitudes to a constant value between 0.0 and 1.0.

`aerosol_set_tau`
: Set the aerosol optical thickness for all wavelengths and altitudes to a constant value. The optical thickness defined here is the integral from the user-definded `altitude` to TOA (top of atmosphere).

`aerosol_set_tau550`
: Set the aerosol optical thickness at 550nm. Other wavelengths are scaled accordingly. Note that this option requires for technical reasons that the wavelength interval defined by `wavelength` does contain 550nm. The optical thickness defined here is the integral from the user-definded `altitude` to TOA (top of atmosphere).

`aerosol_season`
: Specify season to get appropriate aerosol profile.

  `1` Spring-summer profile.

  `2` Fall-winter profile.

`aerosol_sizedist_file`
: Aerosol size distribution. Two columns are expected: The radius [micron] and the particle number. Together with `aerosol_refrac_index` or `aerosol_refrac_file` this forms the input to Mie calculations of the aerosol optical properties. See comment at `aerosol_refrac_file`.

`aerosol_ssa_file`
: Location of aerosol single scattering albedo file. The file must have two columns. Column 1 is the altitude in km. The altitude grid must be exactly equal to the altitude grid specified in the file `atmosphere_file`. Column 2 is the single scattering albedo of each layer. At present, the single scattering albedo defined here is constant with wavelength but it is planned to introduce the wavelength dependence in the near future. Comments start with `#`. Empty lines are ignored.

`aerosol_tau_file`
: Location of aerosol optical depth file. The file must have two columns. Column 1 is the altitude in km. The altitude grid must be exactly equal to the altitude grid specified in the file `atmosphere_file`. Column 2 is the aerosol optical depth of each layer. To get wavelength dependence use the parameter `aerosol_angstrom`. Comments start with `#`. Empty lines are ignored.

`aerosol_visibility`
: Visibility in km.

`aerosol_vulcan`
: Aerosol situation above 2 km. Integer.

  `1` Background aerosols.

`2` Moderate vulcanic aerosols.

`3` High vulcanic aerosols.

`4` Extreme vulcanic aerosols.

`albedo` The Lambertian surface albedo, a number between 0.0 and 1.0, constant for all wavelengths. For wavelength dependent surface albedo use `albedo_file`. The default albedo is 0.0.

`albedo_file`
Location of surface albedo file for wavelength dependent surface albedo. The file must have two columns. Column 1 is the wavelength, in nm, and column 2 the corresponding Lambertian surface albedo. The wavelength grid may be freely set. The albedo will be interpolated to the wavelength grid used for the radiation calculation. Comments start with `#`. Empty lines are ignored.

`angstrom` Still supported but obsolete. Replaced by `aerosol_angstrom`.

`altitude` Set the bottom level in the model atmosphere provided in `atmosphere_file` to be at altitude (km).

```
altitude 0.73   # Altitude of IFU, Garmisch-Partenkirchen
                # Be aware, for this to work the atmosphere
                # file must start at 0 km.
```

The profiles of pressure, temperature, molecular absorbers, ice and water clouds are cut at the specified altitude. The aerosol profile is not affected by `altitude` but starts right from the model surface. This is a convenient way for the user to calculate the radiation at other altitudes than sealevel. Note that `altitude` is very different from `zout` where the radiation is calculated at an altitude of zout over the surface. E.g. to calculate the radiation field 1 km above the surface at 0.73 above sealevel, one would specify '`altitude` 0.73' and '`zout` 1.0'.

A second optional argument may be given to `altitude` as e.g.

```
altitude 0.73 0.5
```

Here the bottom level will be at 0.73 km and the vertical resolution of the model atmosphere will be redistributed to have a spacing between levels specified by the second number, here 0.5 km. Be aware that specifying a fine vertical spacing will produce many layers thus increasing the computing time. Also the radiative transfer equation solvers implemented in Fortran 77 might need to have some array sizes increased (see '`src_f/DISORT.MXD`').

`atmosphere_file`
Location of the atmospheric data file. Must have at least three columns containing the altitude, pressure, and temperature. Missing profiles are filled with 0 (e.g., if you did not specify the ozone profile, there will be no ozone absorption!), with exception of the air density which is calculated from pressure and temperature. Other traces gases may be set by `dens_file`. The columns are interpreted as follows:

`1. z(km)` Altitude above sea level in km

2. `p(mb)` Pressure in mbar

3. `T(K)` Temperature in K

4. `air_density(cm-3)`
   Air density in cm-3

5. `o3_density(cm-3)`
   Ozone density in cm-3

6. `o2(cm-3)`
   Oxygen density in cm-3

7. `h2o(cm-3)`
   Water vapour density in cm-3

8. `co2(cm-3)`
   CO2 density in cm-3

9. `no2(cm-3)`
   NO2 density in cm-3

The atmosphere is specified top-down, that is, the top level is the first line in the file, the bottom (surface) level the last line. All properties refer to model *level* z, not to model *layer*. It is important that the correct units are used, otherwise unpredictable results are guaranteed. Comments start with `#`. Empty lines are ignored. Please note that there is some redundancy: Assuming that air is an ideal gas, the air density can be calulated from pressure and temperature, dens = p / kT. `uvspec` will check if this relation is fulfilled and will issue a warning if it is not. libRadtran provides the six standard atmospheres by Anderson et al. (1986):

`afglt` Tropical

`afglms` Midlatitude Summer

`afglmw` Midlatitude Winter

`afglss` Subarctic Summer

`afglsw` Subarctic Winter

`afglus` U.S. Standard

`brightness`
Convert radiances / irradiances to brightness temperatures.

`ch4_mixing_ratio`
The mixing ratio of CH4 in ppm (default: 1.6 ppm).

`co2_mixing_ratio`
The mixing ratio of CO2 in ppm; scale the profile so that the mixing ratio at the user-defined `altitude` assumes the specified value.

`correlated_k`
To calculate integrated shortwave or longwave irradiance, or to simulate satellite instrument channels, choose between the following correlated-k schemes:

`Kato` Kato et al. (1999), shortwave; based on HITRAN 96. Please note that the bands above 2.5 micron are not very reliable which, however, has only little impact on integrated shortwave radiation.

`Kato2` Kato et al. (1999), shortwave; optimized version (Seiji Kato, personal communication, 2003); please note that `Kato2` only has 148 subbands (that is, calls to the `rte_solver`) compared to 575 for `Kato` which translates to an increase in computational speed by up to a factor of 4 with only little increase in uncertainty. The absorption data are based on HITRAN 2000. Please note that the bands above 2.5 micron are not very reliable which, however, has only little impact on integrated shortwave radiation.

`Kato2.96` Kato et al. (1999), shortwave; optimized version (Seiji Kato, personal communication, 2003); similar to `Kato2` but based on HITRAN96. Please note that the bands above 2.5 micron are not very reliable which, however, has only little impact on integrated shortwave radiation.

`Fu` Fu and Liou (1992/93), shortwave and longwave; fast parameterization, developed for climate models.

`AVHRR_KRATZ`
Kratz (1995), AVHRR instrument channels

`LOWTRAN` Gas absorption parameterization from LOWTRAN; code adopted from SBDART (Ricchiazzi et al., 1998); please see the section on "Spectral resolution".

`SBDART` Identical to LOWTRAN.

If correlated_k KATO/KATO2/FU/AVHRR_KRATZ is specified, the extraterrestrial flux is taken from internally defined files specific for each parameterization, not from `solar_file`. The output is the integrated irradiance for each band. To get e.g. integrated shortwave irradiance, simply add all bands of the Kato et al. (1999) or the Fu and Liou (1992/93) parameterization. The five AVHRR channels are weighted sums of the libRadtran output. Examples how to integrate the output in the `AVHRR_KRATZ` case are included in the `uvspec` self-test which is initiated with `make check`.

`cox_and_munk_pcl`
Pigment concentration for Cox and Munk ocean surface BRDF (in mg/m-3); at present only available with `rte_solver DISORT2`. The number of streams (`nstr`) is automatically increased to 16 if cox_and_munk BRDF is switched on, to avoid numerical problems.

`cox_and_munk_sal`
Salinity for Cox and Munk ocean surface BRDF (in ppt); at present only available with `rte_solver DISORT2`. The number of streams (`nstr`) is automatically increased to 16 if cox_and_munk BRDF is switched on, to avoid numerical problems.

`cox_and_munk_u10`
: Wind speed for Cox and Munk ocean surface BRDF (in m/s); at present only available with `rte_solver DISORT2`. The minimum allowed wind speed is 1 m/s because otherwise the strong specular reflection causes numerical problems. If a lower value is specified, the wind speed is automatically set to 1m/s. Also, the number of streams (`nstr`) is automatically increased to 16 if cox_and_munk BRDF is switched on, to avoid numerical problems.

`crs_file` May be used to specify cross sections of ozone (O3), nitrogendioxide (NO2), bromine oxide (BRO), OCLO, or HCHO to be used instead of one of those coming with libRadtran. No temperature dependence may be specified. Use as follows:

```
crs_file NO2 ../examples/no2_crs.dat
```

The `NO2` or `O3`,`BRO` or `OCLO` or `HCHO` must be specified to identify the specie for which the cross section applies. The cross section file has two columns:

1. `wavelength (nm)`
2. `cross section (cm+2)`

`data_files_path`
: The path to the directory where all uvspec internal data files live, e.g. the files that are in the subdirectories of the '`data`' directory that comes with the uvspec distribution. The default is `../data/`.

`day_of_year`
: Integer, to correct the calculated radiation quantities for the Sun-Earth distance for the specified Julian day (1-365). If not given, the Earth-Sun distance is 1 AU (i.e. equinox distances), that is, no correction is applied to the extraterrestrial irradiance `solar_file`.

`deltam` Turn delta-M scaling on/off. Set to either `on` or `off`. Note that for the `rte_solver disort2` delta-M scaling is hardcoded to be always on.

`dens_column`
: Set the total column of a density profile. The column is integrated between the user-defined altitude and TOA (top of atmosphere). The syntax is

```
dens_column specie column [unit]
```

where specie is one of O3, NO2, BRO, OCLO, or HCHO, see also `dens_file`. Column is the total column value of the trace gas and the column is in unit which is optional. The default units are O3 (DU), NO2 (CM_2), BRO (CM_2), OCLO (CM_2), and HCHO (CM_2). Here DU is Dobson units and CM_2 =cm-2. In addition DU may be used for NO2 as well, e.g.

```
dens_column NO2 1.2 DU
```

`dens_file`
: Specify density profiles (or matrix, see below) of various traces gases to be included in the radiative transfer calculation. At the moment ozone (O3), nitrogen dioxide (NO2), water vapor (H2O), bromine oxide (BRO), OCLO, HCHO, and carbon dioxide (CO2) are included. The various density profiles are idenfied by their abbrevations given in the parenthesis above.

```
dens_file O3 ../examples/afglus_o3.dat
```

The density file has two columns:

1. `z(km)` Altitude above sea level in km.
2. `density of trace gas (cm-3)`
   The density of the trace gase

To scale the profile to a total column value use `dens_column`.

For airmass factor calculations it is for some species necessary to account for the variation of the profile with sza. This may be accomplished by specifying a `dens_file` in the following format:

1. `row 1, column 1: a zero, that is 0.0`
2. `column 1 below row 1: Altitude above sea level in km.`
3. `row one after column 1: The solar zenith angle in degrees`
4. `The rest of the matrix:`
   the density of trace gases as a function of solar zenith angle and altitude.

The matrix may only be specified for one specie. It may however be combined with profiles of other species. For examples see the `examples` directory. A density matrix may only be used in connection with `rte_solver sdisort`.

`f11_mixing_ratio`
: The mixing ratio of F11 in ppm (default: 0.000268 ppm).

`f12_mixing_ratio`
: The mixing ratio of F12 in ppm (default: 0.000503 ppm).

`f22_mixing_ratio`
: The mixing ratio of F22 in ppm (default: 0.000105 ppm).

`filter_function_file`
: If specified, the calculated spectrum is multiplied with a filter function defined in '`filter_function_file`'. The file must contain two columns. Column 1 is the wavelength, in nm. Column 2 is the corresponding filter function value. Comments start with `#`. Empty lines are ignored. In combination with `output sum` or `output integrate` this option is useful e.g. to calculate weighted irradiances or actinic fluxes or to simulate broadband or satellite observations.

`fisot` Specifies that isotropic illumination is used at top-boundary instead of beam source. Useful for those who want to calculate the reflectance for a homogeneous or inhomogeneous atmosphere. The intensity is still set by `solar_file`.

`flexstor` Output is in flexstor format. May not be combined with `header`. Also, does not currently work when `umu` and/or `phi` is specified.

`h2o_mixing_ratio`
: The mixing ratio of H2O in ppm. Scale the profile so that the mixing ratio at the user-define `altitude` assumes the specified value.

`h2o_precip`
: Precipitable water in kg / m2 (which is approximately 1mm). The water vapor profile is scaled accordingly. The precipitable water is integrated from the user-defined `altitude` to TOA (top of atmosphere).

`header`
: Output information on some of the input parameters. May not be combined with `flexstor`.

`n2o_mixing_ratio`
: The mixing ratio of N2O in ppm (default: 0.28 ppm).

`ic_cloudcover`
: Set the fraction of the horizontal sky area which is covered by clouds. When a cloud cover is specified, the result will be calculated by the independent pixel approximation (IPA), that is, as weighted average of cloudless sky and overcast sky, where the cloud properties are taken from `ic_file`, etc. Please note that, if both `wc_cloudcover` and `ic_cloudcover` are set, both must be equal.

`ic_file`
: Location of file defining ice cloud properties. The file must contain three columns. Column 1 is the altitude in km, column 2 the ice water content in grams per qubic meter, and column 3 the effective particle radius in micron. The ice water content and effectice particle radius must be specified at the same altitude grid as in `atmosphere_file`. Note that the definition of cloud altitudes in `ic_file` refers to sea level, not to altitude above ground. E.g., when altitude is set to 1.63km, and the first cloud level is defined at 3km, the cloud would start at 1.37km above ground. The optical depth of a layer is calculated using information from the upper and lower levels defining the layer. Comments start with `#`. Empty lines are ignored. An example of an ice cloud is given in '`examples/IC.DAT`'. Per default, the cloud properties are interpreted as properties at a given altitude *level*. If `ic_layer` is defined, they are interpreted as *layer* properties (please see the section about water clouds for a realistic example how the contents of the `ic_file` are converted to optical properties).

`ic_files`
: A way to specify ice cloud optical depth, single scattering albedo, and phase function moments for each layer. The file specified by `ic_files` has two columns where column 1 is the altitude in km. The altitudes must be the same as those specified in the file `atmosphere_file`. The second column is the name of a file which defines the optical properties of the level starting at the given altitude. The files specified in the second column must have the following format:

  `Column 1:` The wavelength in nm. These wavelengths may be different from those in `solar_file`. Optical properties are interpolated to the requested wavelengths.

  `Column 2:` The extinction coefficient of the layer in units km-1.

  `Column 3:` The single scattering albedo of the layer.

  `Column 4-(nmom+4):`
  The moments of the scattering phase function.

Note that if using the `rte_solver disort2` it makes good sense to make the number of moments larger than `nstr`. For `rte_solver disort` and `rte_solver polradtran` the number of moments included in the calculations will be `nstr+1`. Higher order moments will be ignored for these solvers. Please note that the uppermost line of the `ic_files` denotes simply the top altitude of the uppermost layer. The optical properties of this line are consequently ignored. There are two options for this line: either an optical property file with zero optical thickness is specified or "NULL" instead.

`ic_fu_tau`

Specify if the Fu (1996) optical properties are delta-scaled or not. With `ic_fu_tau scaled` delta-scaling is switched on, with `ic_fu_tau unscaled` it is switched off. The default is with delta-scaling, as suggested in the paper (equation 3.8 and description; equations A.2a - A.2c and description). If you define a cloud only by its microphysical properties (ice water content, effective radius), delta-scaling should certainly be switched on and you do not need to read further. If, however, you want to use the Fu (1996) parameterization in combination with one of `ic_set_tau/tau550/gg/ssa` or `ic_scale_gg/ssa` it might be reasonable to switch delta-scaling off and you should make sure that you understand the following. Citing from Fu (1996): "For nonspherical particles in cirrus clouds, it is found that a simple representation of the scattering phase function through the asymmetry factor is inadequate (Fu and Takano 1994). As demonstrated in appendix A, the fraction of scattered energy residing in the forward peak, f, needs to be removed from the scattering parameters to incorporate the strong forward peak contribution in multiple scattering." Or in other words, the sharp forward peak is truncated and added to the unscattered direct radiation. The remaining phase function (excluding the sharp forward peak) can be safely approximated by a Henyey-Greenstein function. The scaling implies a reduction of the optical thickness, the asymmetry parameter, and the single scattering albedo. This reduction can be rather severe, e.g. a factor of about 3 for the optical thickness in the visible spectral range. This implies seemingly inconsistent optical properties: For idential IWC content and effective radius, `ic_properties key/yang` would give an (unscaled) optical thickness about three times higher than `ic_properties fu`. The effect on the radiation field, however, will be comparable, due the consistent scaling of optical thickness, asymmetry parameter, and single scattering albedo. If you, however, adjust the optical thickness using e.g. `ic_set_tau`, the effect on the radiation field will be completely different because the (unscaled) optical thickness by Key (2002) has a completely different meaning as the (scaled) optical thickness by Fu (1996). In such cases it might be reasonable to switch scaling off. This is a complicated and confusing topic and it is suggested that you play around a bit with the options, read the Fu (1996) paper, and make heavy use of the `verbose` feature.

`ic_fu_reff`

If `ic_fu_reff yang` is specified, the Fu (1996/98) parameterization uses the same definition of the effective radius as the Key et al. (2002) and Yang et al. (2000) parameterization; see discussion of `ic_properties`.

`ic_habit` Ice crystal habit for the Yang et al. (2000) and Key et al. (2002) parameterizations, see also `ic_properties key/yang`. May be one of solid-column, hollow-column, rough-aggregate, rosette-4, rosette-6, plate, droxtal, and spheroid. Please note that this parameterization is only valid for a restricted size range, depending on the habit (see table 1 in Key et al. (2002)). Also, some of the habits are only available for wavelengths below 5 micron (rosette-4) while others are only available for wavelengths larger than 3 micron (droxtal, spheroid).

`ic_ipa_files`
A two-column file, defining ice cloud property files (see `ic_file`) in the first column and the correspoding weights in the second column. The radiative transfer calculation is performed independently for each cloud column and the result is the weighted average of all independent columns. If `ic_ipa_files` and `wc_ipa_files` are both defined, both must have the same columns in the same order, otherwise `uvspec` will complain.

`ic_layer` Specify ice cloud properties for model *layers* instead of *levels* (see also `ic_file`).

`ic_properties`
Defines how ice water content and effective particle radius are translated to optical properties. Possible choices are

`fu` Parameterization by Fu (1996), Fu et al. (1998), see ic_file; this is the default setting. Note that this is a parameterization which has been created to calculate fluxes but not radiances. Note also that the optical properties in the solar range provided by Fu (1996) are delta-scaled properties (that is, the forward peak of the phase function is truncated and optical thickness, asymmetry parameter, and single scattering albedo are reduced accordingly). Please see the detailed discussion in the description of `ic_fu_tau`! For wavelengths up to 4 micron Fu (1996) is used while for wavelengths larger than 4 micron Fu et al. (1998) is chosen. Please note that Fu (1996) is based on ray-tracing calculations while Fu et al. (1998) is a mixture of ray-tracing and Mie calculations (which is required for the infrared wavelengths where the geometrical assumption does not hold). Hence, both parameterizations are not fully consistent. Rather, differences of some % are to be expected in the wavelength region where both parameterizations overlap. Also, the wavelength dependence in the solar and infrared parts is treated differently: In the solar part (Fu, 1996) the optical properties are defined for wavelength bands - hence they are assumed constant within each band. In the infrared (Fu et al. 1998) they are defined at certain wavelengths and linearely interpolated in between. If you use this option, please see also the discussion of `ic_fu_tau` and `ic_fu_reff`.

`echam4` Use the very simple two-band parameterization of the ECHAM4 climate model, described in Roeckner et al. (1996); this is probably only meaningful if you want to compare your results with ECHAM4,

the two bands are 0.2 - 0.68 micron and 0.68 - 4.0 micron; within these bands, the optical properties are assumed constant.

`key` Parameterization by Key et al. (2002). This parameterization can also be used to calculate radiances because it uses a double-Henyey-Greenstein phase function which better represents both forward and backward peaks. This parameterization covers the region from 0.2 to 5.0 micron and is available for the following habits: solid-column, hollow-column, aggregate, rosette-4, rosette-6, and plate.

`mie` Use pre-calculated Mie tables; useful for `correlated_k`; the tables are expected in `data_files_path`/correlated_k/../
For spectral or pseudo-spectral (`correlated_k` `sbdart`) calculations, a set of pre-calculated tables is also available; the wavelength grid points of these data has been carefully selected such that the extinction cross section, single scattering albedo, and the asymmetry parameter are accurate to 1% (compared to the fully-resolved Mie calculation) for all wavelengths between 250nm and 100 micron. For spectral or pseudo-spectral calculations `wc_properties_interpolate` has to be defined explicitely to initiate the interpolation of the optical properties to the internal wavelength grid. Please note that this option may be extremely memory-comsuming because for each internal wavelength a full set of Legendre moments of the phase function is stored (up to several thousands). The Mie tables are not part of the standard distribution (because of their large size) but they are freely available from http://www.libradtran.org. Note that a Mie calculation assumes spherical ice particles, the scattering function of which differs systematically from non-spherical particles. Hence, `ic_properties` `mie` is usually not representative of natural ice clouds.

`filename` Read optical properties from specified filename; file format is as produced by Frank Evans' `cloudprp`; for each of the internal (computational) wavelengths, a separate file is expected (this option is subject to change).

Please note also that, in contrast to spherical particles, there is no unique definition of effective size for non-spherical particles. In particular, the above parameterizations use different definitions which, however, differ only by a constant factor. Yang et al. (2000), Key et al. (2002) use the general definition

$$r_{\text{eff}} = \frac{3}{4}\frac{\int V(h)n(h)dh}{\int A(h)n(h)dh}$$

where h is the maximum dimension of an ice crystal, n(h) is the number of particles with maximum dimension h in the size distribution, and V and A are the volume and mean projected area of the particles, respectively. The volume and area are based on the spherical diameter with equivalent volume and the spherical diameter with equivalent projected area as defined by Yang et al.

(2000). On the other hand, Fu et al. (1996,1998) use hexagonal columns and use the following definition

$$r_{\text{eff}} = \frac{\int D^2 L n(L) dL}{2 \int (DL + \frac{\sqrt{3}}{4} D^2) n(L) dL}$$

where D is the width of the ice crystal (that is, the maximum diameter of the hexagonal area) and L is the length. The integrand in the numerator is proportional to the volume while that in the denominator is proportional to the projected area. Evaluating these formulas one finds that, for the same hexagonal particle, the effective radius would be $3\sqrt{3}/4 = 1.299$ times larger following the Yang et al. (2000), Key et al. (2002) definition than the Fu (1996,1998) definition. As an example, an effective radius of $20\mu m$ with "ic_properties fu" and $1.299 \cdot 20\mu m = 26\mu m$ with "ic_properties yang" would give comparable results for hexagonal columns. To use consistent definitions of the effective radius in both parameterizations, use `ic_fu_reff yang`.

`ic_properties_interpolate`
: Interpolate ice cloud optical properties over wavelength; useful for precalculated optical property files defined with `ic_properties`. Please note that this option may be extremely memory-comsuming because for each internal wavelength a full set of Legendre moments of the phase function is stored (up to several thousands).

`ic_scale_gg`
: Scale the ice cloud asymmetry factor for all wavelengths and altitudes with a float between 0.0 and 1.0. If you use this option in combination with the ice cloud properties by Fu (1996), please make sure that you understand the explanation of `ic_fu_tau`.

`ic_scale_ssa`
: Scale the ice cloud single scattering albedo for all wavelengths and altitudes with a float between 0.0 and 1.0. If you use this option in combination with the ice cloud properties by Fu (1996), please make sure that you understand the explanation of `ic_fu_tau`.

`ic_set_gg`
: Set the ice cloud asymmetry factor for all wavelengths and altitudes to a float between -1.0 and 1.0. If you use this option in combination with the ice cloud properties by Fu (1996), please make sure that you understand the explanation of `ic_fu_tau`.

`ic_set_ssa`
: Set the ice cloud single scattering albedo for all wavelengths and altitudes to a value between 0.0 and 1.0. If you use this option in combination with the ice cloud properties by Fu (1996), please make sure that you understand the explanation of `ic_fu_tau`.

`ic_set_tau`
: Set the total ice cloud optical depth to a constant value for all wavelengths. The optical thickness defined here is the integral from the surface at the user-defined `altitude` to TOA (top of atmosphere). This option is useful only for monochromatic calculations or in wavelength regions where the optical properties of ice clouds can be considered constant, e.g. the ultraviolet region. If you use this option in combination with the ice cloud properties by Fu (1996), please make sure that you understand the explanation of `ic_fu_tau`.

`ic_set_tau550`
: Set the ice cloud optical thickness at 550nm. Other wavelengths are scaled accordingly. The optical thickness defined here is the integral from the surface at the user-defined `altitude` to TOA (top of atmosphere). Note that this option requires for technical reasons that the wavelength interval defined by `wavelength` does contain 550nm. If you use this option in combination with the ice cloud properties by Fu (1996), please make sure that you understand the explanation of `ic_fu_tau`.

`include`
: Include a file into the uvspec input. Works exactly like the C `#include` or the Fortran `INCLUDE` statements.

`molecular_tau_file`
: Location of molecular absorption optical depth file. Usually, molecular absorption is calculated from trace gas concentrations provided in `atmosphere_file` (scaled with `ozone_column`, etc. Use this option only if you want to specify the optical depth directly (e.g. for a model intercomparison) or for a line-by-line calculation. If a spectral `molecular_tau_file` is specified, the wavelength grid defined there is used as internal wavelength grid for the radiative transfer calculation, if not defined otherwise with `transmittance_wl_file`. `molecular_tau_file` can be either of the following three formats:

  Monochromatic:
  : Column 1 is the altitude in km where the altitude grid must be exactly equal to the altitude grid specified in `atmosphere_file`. Column 2 is the absorption optical depth of each layer.

  Spectral, ASCII:
  : The first line contains the level altitudes in decreasing order; the following lines contain the wavelength [nm] in the first column and then the absorption optical depths of each layer.

  Spectral, netcdf:
  : netcdf is a common platform independent format; the description, a library to read and write netcdf including some tools to generate netcdf is available at http://www.unidata.ucar.edu/packages/netcdf/. A `molecular_tau_file` must obey certain rules; an example is available at the libRadtran homepage, '`UVSPEC.O2A.afglms.cdf`', a line-by-line spectrum of the oxygen A-Band around 760nm, calculated for the mid-latitude summer atmosphere by Anderson et al. (1986). The

advantage of netcdf compared to ASCII is that it is much faster to read, and that the file is a self-contained, including data and a description of the variables and arrays. It is therefore particularly useful for line-by-line calculations where usually many spectral data points are involved.

Comments start with `#`. Empty lines are ignored.

`no_absorption`
Switch absorption off.

`no_molecular_absorption`
Switch all (molecular, aerosol, cloud, and ice cloud) absorption off. Note that this option does not affect the single scattering albedo; e.g. with an aerosol optical depth of 1 and a single scattering albedo of 0.7, the scattering optical depth will still be 0.7, even with `no_absorption`.

`no_rayleigh`
Switch Rayleigh scattering off.

`no_scattering`
Switch scattering off.

`no2_column_du`
Obsolete, use `dens_column` instead. Set the NO2 column to a given value. The column is integrated between the user-defined `altitude` and TOA (top of atmosphere). The value must be in Dobson units. If value is negative or `no2_column_du` is not specified, the NO2 column is not scaled.

`no2_column_moleccm-2`
Obsolete, use `dens_column` instead. Set the NO2 column to a given value. The column is integrated between the user-defined `altitude` and TOA (top of atmosphere). The value must be in molecules / cm2. If value is negative or `no2_column_moleccm-2` is not specified, the NO2 column is not scaled.

`nscat` The order of scattering for the `sos` radiative transfer equation solver. Default is 20. May also be used with the `sdisort` solver. If set to 1 `sdisort` will run in single scattering mode while if set in to 2, `sdisort` runs in full multiple scattering mode.

`nrefrac` For the `rte_solver sdisort` refraction may be included by specifying `nrefrac`. If refraction is included also set parameter `refraction_file`.

- `0` No refraction, default.
- `1` Refraction included using fast, but harsh method.
- `2` Refraction included using slow, but accurate method.

`nstr` Number of streams used to solve the radiative transfer equation. Default is 6 for fluxes and 16 for radiances.

`o2_mixing_ratio`
: The mixing ratio of O2 in ppm; scale the profile so that the mixing ratio at the user-defined `altitude` assumes the specified value.

`o3_crs` Choose between the following ozone cross sections.

`Bass_and_Paur`
: Bass and Paur ozone cross section.

`Molina` Molina and Molina (1986) ozone cross section.

`Daumont` Ozone cross section by Daumont et al. (1992), Malicet et al. (1995).

Molina and Molina is default.

`ozone_column`
: Obsolete, use `dens_column` instead. Set the ozone column to a given value. The column is integrated between the user-defined `altitude` and TOA (top of atmosphere). The value must be in Dobson units. If value is negative or `ozone_column` is not specified, the ozone column is not scaled.

`output` Output processing. Choose between the following options:

`sum` Sum output over wavelength. Useful in combination with the `correlated_k` option.

`integrate`
: Integrate output over wavelength. Useful for spectral calculations.

`heating` Calculation of heating rates. Note that heating rates are only well-behaved up to altitudes for which the respective correlated-k options are valid. E.g. about 60 km for `fu` and about 80 km for `Kato`, `Kato2` and `sbdart`. Also note that output is only provided at altitudes specified by `zout`. To get heating rate profiles a number of altitudes must thus be specified. Output is in two columns with column 1 being the altitude and column 2 the heating rate. Heating rates are output in units of K/day.

`none` No processing - output spectral information (default).

`output_user`
: User defined output. Here the user may specify the columns desired for output. Default output is the wavelength lambda, edir, edn, eup, uavgdir, uavgdn, uavgup for disort, sdisort, and spsdisort, whereas the default for twostr is lambda, edir, edn, eup, uavg. The lines containing radiances are not affected.

`lambda` Wavelength in nm.

`zout` Output altitude in km.

`edir, eglo, edn, eup, enet, esum`
: Direct, global, diffuse downward, and diffuse upward irradiance. Netto is global - upward, sum is global + upward.

`fdir, fglo, fdn, fup, f`
: Direct, global, diffuse downward, diffuse upward, and total actinic flux.

`uavgdir, uavgglo, uavgdn, uavgup, uavg`
: Direct, global, diffuse downward, diffuse upward, and total diffuse mean intensity (= actinic flux / 4 pi).

`albedo` Albedo.

`phi` Azimuth output angles (in degrees) in increasing order. The radiance is output at `phi` and `umu`.

`phi0` Azimuth angles of the sun (0 to 360 degrees). If `phi0` varies as a function of wavelength use `sza_file`.

`polradtran_aziorder`
: Order of Fourier azimuth series: 0 is azimuthially symmetric case. Default 0.

`polradtran_max_delta_tau`
: Initial layer thickness for doubling; governs accuracy, 10E-5 should be adequate. Do not go beyond half the real precision, i.e. 10e-8 for REAL*8. Default 1.e-05.

`polradtran_nstokes`
: Number of Stokes parameters

  `1` for I (no polarization)

  `2` for I,Q

  `3` for I,Q,U

  `4` for I,Q,U,V

  Default is 1.

`polradtran_quad_type`
: Type of quadrature used:

  `G` gaussian

  `D` double gaussian,

  `L` Lobatto

  `E` extra-angle(s), this must be used of `polradtran` is used in combination with `umu`. Will internally use Gaussian scheme (`G`). See also radtran documentation.

  Default G.

`polradtran_src_code`
: Radiation sources included:

  `0` none

  `1` solar

`2` thermal

`3` both

Default 1.

`pressure` The surface pressure (at the user-defined `altitude` in hPa. The pressure profile and density profiles are scaled accordingly.

`prndis` Specify one or more integers between 1 and 7. Print various disort input and output in disorts own format. See '`disort.doc`' for more information. **Warning:** Produces a lot of output.

`quiet` If specfied, a number of informative messages about what options are used and progress during computations are turned off. Otherwise these messages are output to stderr. See also `verbose`.

`rayleigh_crs`
Choose between the following Rayleigh scattering cross sections.

`Bodhaine` Bodhaine et al (1999) Rayleigh scattering cross section.

`Nicolet` Nicolet (1984) Rayleigh scattering cross section.

Bodhaine et al. is default.

`rayleigh_depol`
Rayleigh depolarization factor; the Rayleigh scattering phase function is p(mu) = a + b * mu**2 where a = 1.5*(1+depol)/(2+depol) and b = 1.5*(1-depol)/(2+depol). By default the depolarization is calculated using the expressions from Bodhaine et al. (1999).

`rayleigh_tau_file`
Location of Rayleigh scattering optical depth file. Usually, the Rayleigh scattering cross section is calculated from the air pressure provided in `atmosphere_file` (scaled with `pressure`). Use this option only if you really want to specify the optical depth directly (e.g. for a model intercomparison). The optical thickness profile may be either monochromatic or spectral. The format is exactly the same as for `molecular_tau_file`.

`reflectivity`
Calculate transmission / reflectivity instead of absolute quantities. For irradiances / actinic fluxes the transmission T is defined as

$$T = \frac{E}{E_0 \cos\theta}$$

where E is the irradiance / actinic flux, E_0 is the extraterrestrial flux, and θ is the solar zenith angle. The reflectivity R is defined as

$$R = \frac{\pi \cdot L}{E_0 \cos\theta}$$

where L is the radiance, E_0 is the extraterrestrial flux, and θ is the solar zenith angle. Obviously, reflectivities do not depend on Sun-Earth distance. Please note the difference to `transmittance`.

`reverse` Option for the strong and bold. Reverses the atmospheric input to the radiative transfer solvers. That is, the atmosphere is turned on the head. Yes, that is actually useful for some purposes. If you think you need this contact the author. Otherwise, do not use.

`rh_file` File that defines a profile of relative humidity. If specified, the water vapour profile in atmosphere_file is over-written. If -1 is specified at a level, the value from atmosphere_file is used.

`rpv_file` 4 column file, containing the Rahman, Pinty, and Verstraete (RPV) BDRF parameterization, Rahman et al. (1993). Bidirectional reflectance distribution functions for a variety of surfaces are given in the paper. This option in only supported with DISORT 2.0 and MYSTIC. The columns of the input file are wavelength [nm], rho0, k, and theta. The parameters are interpolated linearely to the internal wavelength grid. To make sure that the results are reasonable, specify the RPV data on a wavelength grid similar or equal to that used internally for the radiative transfer calculation! Optionally, a fifth column with a constant scaling factor may be defined which, however, is only used by `rte_solver disort2`.

`rpv_k` Constant RPV rho0, see `rpv_file`. `rpv_k` overwrites the wavelength-dependent value defined in `rpv_file`.

`rpv_rho0` Constant RPV rho0, see `rpv_file`. `rpv_rho0` overwrites the wavelength-dependent value defined in `rpv_file`.

`rpv_theta`
Constant RPV theta, see `rpv_file`. `rpv_theta` overwrites the wavelength-dependent value defined in `rpv_file`.

`rpv_sigma`
Constant RPV sigma, to be used for snow (Deguenther and Meerkoetter, 2000). A wavelength dependent sigma is not yet available.

`rpv_t1` Constant RPV t1, to be used for snow (Deguenther and Meerkoetter, 2000). A wavelength dependent sigma is not yet available.

`rpv_t2` Constant RPV t2, to be used for snow (Deguenther and Meerkoetter, 2000). A wavelength dependent sigma is not yet available.

`rpv_scale`
Apply a constant scaling factor for the RPV BRDF. Required e.g. if the the albedo should be set to a certain value. This factor is only used by `rte_solver disort2`.

`rte_solver`
Set the radiative transfer equation solver to be used. Options are

`disort` The standard plane–parallel disort algorithm by Stamnes et al. (1988), version 1.3. For documentation see '`src_f/DISORT.doc`' as well as the papers and the DISORT report at

ftp://climate.gsfc.nasa.gov/pub/wiscombe/Multiple_Scatt/. To optimize for computational time and memory, please adjust the parameters in src_f/DISORT.MXD for your application and re-compile.

`disort2` Version 2 of disort. For documentation see '`src_f/DISORT2.doc`' as well as the papers and the DISORT report at ftp://climate.gsfc.nasa.gov/pub/wiscombe/Multiple_Scatt/. disort2 has several improvements compared to its 'ancestor' disort 1.3. Hence we recommend to use disort2 rather than the older version. To optimize for computational time and memory, please adjust the parameters in src_f/DISORT.MXD for your application and re-compile.

`sdisort` Pseudospherical disort as described by Dahlback and Stamnes (1991). Double precision version. To optimize for computational time and memory, please adjust the parameters in src_f/DISORT.MXD for your application and re-compile.

`spsdisort`
Pseudospherical disort as described by Dahlback and Stamnes (1991), single precision version. **Warning:** it is not recommended to use spsdisort for really large solar zenith angles nor for cloudy conditions. For large optical thickness it is numerically unstable and may produce wrong results. To optimize for computational time and memory, please adjust the parameters in src_f/DISORT.MXD for your application and re-compile.

`polradtran`
The plane-parallel radiative transfer solver of Evans and Stephens (1991). Includes polarization. Note that polarization effects of aerosols and clouds are currently not included.

`twostr` The two–stream radiative transfer solver described by Kylling et al. (1995).

`sos` A scalar pseudospherical succesive orders of scattering code. Works for solar zenith angles smaller than 90 degrees. Can calculate azimuthally averaged radiances. Set `nscat` to specify the order of scattering.

`montecarlo`
The MYSTIC code, see http://www.bmayer.de/mystic.html. Note that MYSTIC is not part of the libRadtran distribution at present. However, it has been given to some users on a collaborative basis.

`tzs` TZS stands for "thermal, zero scattering" and is a very fast analytical solution for the special case of thermal emission in a non-scattering atmosphere. Please note that TZS does only radiance calculations at top of the atmosphere.

`sss` SSS stands for "solar, single scattering" and is an analytical single scattering approximation which might be reasonable for a optically thin atmosphere. Please note that SSS does only radiance calculations at top of the atmosphere.

`null` The NULL solver does not solve the radiative transfer equation. However, it sets up the optical properties, and does the post-processing; useful if you are either interested in the overhead time required by a particular model input or if you are simply interested in the optical properties, as output by `verbose`.

`slit_function_file`
If specified, the calculated spectrum is convolved with the function found in the '`slit_function_file`'. The file must contain two columns. Column 1 is the wavelength, in nm, and relative to the center wavelength. Column 2 is the corresponding slit function value. It must be unity at the maximum. The wavelength steps in the slit function file must be equidistant. Comments start with `#`. Empty lines are ignored.

`solar_file`
Location of file holding the extraterrestrial spectrum. The file must contain two columns. Column 1 is the wavelength, in nm, and column 2 the corresponding extraterrestrial flux. The user may freely use any units he/she wants on the extraterrestrial flux. The wavelength grid specified defines the wavelength resolution at which results are returned. However, the wavelength range is determined by `wavelength`. `solar_file` may be omitted for thermal radiation calculations (`source thermal`) as well as `transmittance` and `reflectivity` calculations. If omitted, the output resolution equals the internal wavelength grid which the model choses for the radiative transfer calculation. Comments start with `#`. Empty lines are ignored. Note that `solar_file` is ignored if `correlated_k` is specified.

`source` Solar or thermal source. Set to either `solar` or `thermal`.

`spline` Spline interpolate to wavelengths lambda_0 to lambda_1 in steps of lambda_step, in nm. Specified as e.g.

```
spline 290. 365. 0.5
```

Here, the calculated spectrum is interpolated to wavelengths 290., 290.5, 291., ... , 364.5, 365. For interpolation to arbitrary wavelengths use `spline_file`. The specified wavelength interval must be within the one specified by `wavelength`.

`spline_file`
Spline interpolate to arbitrary wavelengths, in nm, given as a single column in file '`spline_file`'. The specified wavelengths must be within the range specified by `wavelength`. Comments start with `#`. Empty lines are ignored.

`surface_temperature`
Surface temperature, used for thermal infrared calculations. If not specified, the temperature of the lowest atmospheric level is used as surface temperature.

`sza` The solar zenith angle. If the solar zenith angle varies with wavelength, use `sza_file`. The default solar zenith angle is 0.0.

`sza_file` Location of solar zenith angle file for wavelength dependent solar zenith angle. The file must have two or three columns. Column 1 is the wavelength, in nm, and column 2 the corresponding solar zenith angle. Optionally the third column may contain the corresponding solar azimuth angle. The solar azimuth angle is only needed when calculating radiances. The wavelength grid may be freely set. The solar zenith and azimuth angle will be interpolated to the wavelength grid used for the radiation calculation. Comments start with `#`. Empty lines are ignored.

`thermal_bands_file`
File with the center wavelengths and the wavelength band intervals to be used for calculations in the thermal range. The following three columns are expected: center (or reference) wavelength, lower wavelength limit, upper wavelength limit [nm]. `thermal_bands_file` defines the wavelength grid for the radiative transfer calculation. The RTE solver is called for each of the wavelengths in the first column. The atmospheric (scattering, absorption, etc) properties are also evaluated at these wavelengths. For thermal radiation calculations, the Planck function is integrated over the wavelength bands defined in the second and third columns. The result will therefore be a band-integrated irradiance which does only make sense when the `solar_file` grid equals the `thermal_bands_file` grid.

`thermal_bandwidth`
Specify a constant bandwidth in cm-1 for thermal calculations. The default is 1 cm-1. This option is ignored if the bands are defined explicitely, like with `thermal_bands_file` or `correlated_k KATO/FU/AVHRR_KRATZ`.

`transmittance`
Calculate transmittance / reflectance instead of absolute quantities. That is, set the extraterrestrial irradiance to 1 and do not correct for Sun-Earth distance:

$$T = \frac{E}{E_0}$$

where E is the irradiance / actinic flux / radiance and E_0 is the extraterrestrial flux. Please note the difference to `reflectivity`.

`transmittance_wl_file`
Location of single column file that sets the wavelength grid used for the internal transmittance calculations. The wavelengths must be in nm. Do not use this option unless you know what you are doing. Comments start with `#`. Empty lines are ignored.

`umu` Cosine of output polar angles in increasing order, starting with negative (downward) values (if any) and on through positive (upward) values. Must not have any zero values. The azimuthally averaged intensity `u0u` is output at `umu`.

`verbose` If specified abundances of informative messages are output to stderr. To make use of this information, you may want to write the standard uvspec output to one file and the diagnostic messages to another. To do so, try `(./uvspec < uvspec.inp > uvspec.out) >& verbose.txt`. The irradiances and radiances will be written to '`uvspec.out`' while all diagnostic messages go into '`verbose.txt`'. See also `quiet`.

`wc_cloudcover`
Set the fraction of the horizontal sky area which is covered by clouds. When a cloud cover is specified, the result will be calculated by the independent pixel approximation (IPA), that is, as weighted average of cloudless sky and overcast sky, where the cloud properties are taken from `wc_file`, etc. Please note that, if both `wc_cloudcover` and `ic_cloudcover` are set, both must be equal.

`wc_file` Location of file defining water cloud properties. The file must contain three columns. Column 1 is the altitude in km, column 2 the liquid water content in grams per qubic meter, and column 3 the effective droplet radius in micron. Note that the definition of cloud altitudes in `wc_file` refers to sea level, not to altitude above ground. E.g., when altitude is set to 1.63km, and the first cloud level is defined at 3km, the cloud would start at 1.37km above ground. The optical depth of a layer is calculated using information from the upper and lower levels defining the layer. Comments start with `#`. Empty lines are ignored. An example of a cloud is given in '`examples/WC.DAT`'. Per default, the cloud properties are interpreted as properties at a given altitude *level*. If `wc_layer` is defined, they are interpreted as *layer* properties (please see the section about water clouds for a realistic example how the contents of the `wc_file` are converted to optical properties).

`wc_files` A way to specify cloud extinction coefficient, single scattering albedo, and scattering phase function for each layer. The file specified by `wc_files` has two columns where column 1 is the altitude in km. The altitudes must be the same as those specified in the file `atmosphere_file`. The second column is the name of a file which defines the optical properties of the layer starting at the given altitude. The files specified in the second column must have the following format:

`Column 1:` The wavelength in nm. These wavelengths may be different from those in `solar_file`. Optical properties are interpolated to the requested wavelengths.

`Column 2:` The extinction coefficient of the layer in units km-1.

`Column 3:` The single scattering albedo of the layer.

`Column 4-(nmom+4):`
The moments of the scattering phase function.

Note that if using the `rte_solver disort2` it makes good sense to make the number of moments larger than `nstr`. For `rte_solver disort` and `rte_solver polradtran` the number of moments included in the calculations will be `nstr+1`. Higher order moments will be ignored for these solvers. Please note that the

uppermost line of `wc_files` denotes simply the top altitude of the uppermost layer. The optical properties of this line are consequently ignored. There are two options for this line: either an optical property file with zero optical thickness is specified or "NULL" instead.

`wc_ipa_files`
A two-column file, defining water cloud property files (see `wc_file`) in the first column and the corresponding weights in the second column. The radiative transfer calculation is performed independently for each cloud column and the result is the weighted average of all independent columns. If `ic_ipa_files` and `wc_ipa_files` are both defined, both must have the same columns in the same order, otherwise `uvspec` will complain.

`wc_layer` Specify cloud properties for layers instead of levels (see also wc_file). If wc_layer is specified, cloud properties are assumed to be constant over the layer. The layer reaches from the level, where the properties are defined in the `wc_file` to the level above that one. For example, the following lines

```
#       z      LWC    R_eff
#     (km)  (g/m^3) (um)
      4.000   0.0   0.0
      3.000   1.0  10.0
```

define a cloud in the layer between 3 and 4 km with sharp boundaries.

`wc_properties`
Define how liquid water content and effective droplet radius are translated to optical properties. Possible choices are

`hu` Parameterization by Hu and Stamnes (1993); this is the default setting. Note that the parameterization is somewhat different for 'correlated_k FU' than for all other cases because in the latter case the parameterization from the newer (March 2000) Fu and Liou code is used while otherwise the data are taken from the original Hu and Stamnes paper. Note that this parameterization has been developed to calculate irradiances, hence it is less suitable for radiances. This is due to the use of the Henyey-Greenstein phase function as an approximation of the real Mie phase function.

`echam4` Use the very simple two-band parameterization of the ECHAM4 climate model, described in Roeckner et al. (1996); this is probably only meaningful if you want to compare your results with ECHAM4, the two bands are 0.2 - 0.68 micron and 0.68 - 4.0 micron; within these bands, the optical properties are assumed constant.

`mie` Use pre-calculated Mie tables; useful for `correlated_k`; the tables are expected in `data_files_path`/correlated_k/../
For spectral or pseudo-spectral (`correlated_k sbdart`) calculations, a set of pre-calculated tables is also available; the wavelength grid points of these data has been carefully selected such that the extinction cross section, single scattering albedo, and the asymmetry parameter are accurate to 1% (compared to the fully-resolved

Mie calculation) for all wavelengths between 250nm and 100 micron. For spectral or pseudo-spectral calculations `wc_properties_interpolate` has to be defined explicitely to initiate the interpolation of the optical properties to the internal wavelength grid. Please note that this option may be extremely memory-comsuming because for each internal wavelength a full set of Legendre moments of the phase function is stored (up to several thousands). The Mie tables are not part of the standard distribution (because of their large size) but they are freely available from http://www.libradtran.org. This is the correct option to calculate radiances, to be preferred over the Henyey-Greenstein approach of Hu and Stamnes (1993).

`filename` Read optical properties from specified filename; file format is as produced by Frank Evans' `cloudprp`; for each of the internal (computational) wavelengths, a separate file is expected. Use only if you really know what you are doing (this option is subject to change).

`wc_properties_interpolate`
Interpolate water cloud optical properties over wavelength; useful for precalculated optical property files defined with `wc_properties`. Please note that this option may be extremely memory-comsuming because for each internal wavelength a full set of Legendre moments of the phase function is stored (up to several thousands).

`wc_scale_gg`
Scale the water cloud asymmetry factor for all wavelengths and altitudes with a float between 0.0 and 1.0.

`wc_scale_ssa`
Scale the water cloud single scattering albedo for all wavelengths and altitudes with a float between 0.0 and 1.0.

`wc_set_gg`
Set the water cloud asymmetry factor for all wavelengths and altitudes to a float between -1.0 and 1.0. This option is useful only for monochromatic calculations or in wavelength regions where the optical properties of water clouds can be considered constant, e.g. the ultraviolet range.

`wc_set_ssa`
Set the water cloud single scattering albedo for all wavelengths and altitudes to a float between 0.0 and 1.0. This option is useful only for monochromatic calculations or in wavelength regions where the optical properties of water clouds can be considered constant, e.g. the ultraviolet range.

`wc_set_tau`
Set the total water cloud optical thickness to a constant value for all wavelengths. The optical thickness defined here is the integral from the surface at the user-defined `altitude` to TOA (top of atmosphere). This option is useful

only for monochromatic calculations or in wavelength regions where the optical properties of water clouds can be considered constant, e.g. the ultraviolet range.

`wc_set_tau550`
: Set the water cloud optical thickness at 550nm. The optical thickness defined here is the integral from the surface at the user-defined `altitude` to TOA (top of atmosphere). Other wavelengths are scaled accordingly. Note that this option requires for technical reasons that the wavelength interval defined by `wavelength` does contain 550nm.

`wavelength`
: Set the wavelength range by specifying first and last wavelength in nm. The default output wavelength grid is that defined in `solar_file`, unless `spline` is specified. Note that the radiative transfer calculations are done on an internal grid which can be influenced with `transmittance_wl_file` or `molecular_tau_file`

`wavelength_index`
: Set the wavelengths to be selected. To be used together with predefined wavelength grids, such as `transmittance_wl_file` `molecular_tau_file` and particularly useful in combination with the `correlated_k` option where often only a specified number of wavelength bands is required. E.g., in combination with `correlated_k AVHRR_KRATZ`, `wavelength_index 15 15` will select wavelength index 15 which corresponds to channel 4, or `wavelength_index 10 14` will select those bands required for channel 3. Indices start from 1.

`zout`
: Output altitudes in km. One or more altitudes may be specified in increasing magnitude. Output altitudes must be within the range defined in the `atmosphere_file`. Note that `zout` does not restructure the atmosphere model. Hence, if you specify `zout 0.730` and have your atmosphere model in `atmosphere_file` go all the way down to sea level, i.e. 0.0km., output is presented at 0.730km and calculations performed with an atmosphere between 0.0 and 0.730 km (and above of course). If you want calculations done for e.g. an elevated site you have to restructure the atmosphere model and make sure it stops at the appropriate altitude. This you may either due by editing the atmosphere file or by using `altitude`. Note that for `rte_solver polradtran` the atmosphere file must contain the altitudes specified by `zout`.

`z_interpolate`
: The profile in the `atmosphere_file` provides the constituents of the atmosphere at the given levels. Where additional levels are introduced and in order to calculate layer properties, an assumption about the variation of the property within the layer is required. These interpolation methods can be changed by the `z_interpolate` option. Two arguments are required, the property, and the interpolation method, for example:

```
z_interpolate O3 linear
```

Possible interpolation methods are:

`linear` The specified property varies linearly with height.

`log` The specified property vaires logarithmiccally with hight. This is a reasonable option for all well mixed trace gases.

`linmix` This option is only possible for gas profiles. The mixing ratio of the gas (assuming a logarithmiccally varying air density) varies linearly with hight.

Properties which may be specified are:

`O3, O2, H2O, CO2, NO2, BRO, OCLO, HCHO`

`T` temperature (here linmix is not suitable)

`rh` relative humidity (here linmix is not suitable)

`zout_interpolate`

The z-grid of optical properties is determined by the `atmosphere_file`, and, if specified, by other profile files like `dens_file`, `rh_file`, or `refractive_index_file`. Additional levels might be introduced by the `zout` option and the second argument of the `altitude` option. By default (if `zout_interpolate` is not specified) levels introduced by the `zout` and `altitude` options will not affect the optical property profiles, that is, the optical properties are constant within the layers specified by the `atmosphere_file` and profile files. If `zout_interpolate` is specified, the atmospheric profiles (tracegases, temperature ...) are interpolated to the levels introduced by `zout` and `altitude`, and optical properties are determined from the interpolated atmospheric properties. If `output heating` is specified, `zout_interpolate` will also be automatically activated. `zout_interpolate` generally causes smoother variation of the optical properties.

2.2 mie

`mie` performes Mie scattering calculations for a specified wavelength interval. It reads input from standard input, and outputs to standard output. It is normally invoked in the following way:

```
mie < input_file > output_file
```

The format of the input and output files are described below. Several realistic examples of input files are subsequently given.

Warning: Please note the error checking on input variables is very scarce at the moment. Hence, if you provide erroneous input, the outcome is unpredictable.

2.2.1 The mie input file

The mie input file consists of single line entries, each making up a complete input to the mie program. First on the line comes the parameter name, followed by one or more parameter values. The parameter name and the parameter values are seperated by white space.

Filenames are entered without any surrounding single or double quotes.

Comments are introduced by a `#`. Blank lines are ignored.

The various input parameters are described in detail below.

`mie_program`
: Specify which Mie program to use:

  `BH` The Mie scattering program by Bohren and Hoffmann, ftp://astro.princeton.edu/draine/scat/bhmie/

  `MIEV0` The Mie scattering program by W. Wiscombe. For documentation see src_f/MIEV.doc and the NCAR Mie report at ftp://climate.gsfc.nasa.gov/pub/wiscombe/Single_Scatt/.

`mimcut` (positive) value below which imaginary refractive index is regarded as zero (computation proceeds faster for zero imaginary index). Only used by `mie_program MIEV0`.

`nmom` Number of moments of the phase function to be calculated (default: 0). Only possible with `mie_program MIEV0`.

`r_mean` The radius [micron] of the particle to calculate single scattering properties of. Used together with the wavelength information to calculate the Mie size parameter.

`refrac` Specify which refractive index to use. The following options are implemented:

  `ice` The complex refractive index is taken from the REFICE function of W. Wiscombe.

  `water` The complex refractive index is taken from the REFWAT function of W. Wiscombe.

  `user <re> <im>`
  : A user defined refractive index. re and im are the real and imaginary parts (both positive numbers).

  `file <filename>`
  : Read refractive index from a three-column file containing wavelength [nm], and the real and imaginary parts of the refractive index (both positive numbers). The Mie calculation is done for each wavelength defined here.

`size_distribution_file`
: Specify a two column file, r [micron], n(r), which describes a size distribution of droplets. The Mie calculation is repeated for each value of r found in the size distribution file, and the final result is a weighted average of these values. The user himself therefore has to choose a set of r's suited for his specific purpose.

`temperature`
: Ambient temperature, used to calculate the refractive indices of water and ice. Temperature dependence is only considered above 10 micron (water) and 167 micron (ice), respectively. Default: 300K.

`wavelength`
: Sets the wavelength range, in nm. Specify first wavelength and last wavelength. The wavelength step is specified by `wavelength_step`. Ignored if `refrac file` is specified.

`wavelength_step`
: The wavelength step, in nm. Ignored if `refrac file` is specified.

2.2.2 The mie output

The mie output is currently fixed. We hope to make it user controllable in the future.

mie outputs one line to standard output (stdout) for each wavelength. The format of the output line is

```
lambda refrac_real refrac_img qext omega gg spike pmom(0:nmom)
```

Here

`lambda` Wavelength, in nm.

`refrac_real`
: The real part of the refractive index.

`refrac_img`
: The imaginary part of the refractive index.

`qext` The extinction efficiency factor if `r_mean` was specified or the extinction coefficient [km-1] per unit concentration [cm3/m3] if a `size_distribution_file` was specified. If the medium is liquid water, 1 cm3/m3 equals a liquid water contend of 1g/m3 because the density of water is close to 1 g/cm3. For ice and other substances, the density has to be considered (0.917 g/cm3 for ice at 273K).

`omega` The single scattering albedo.

`gg` The asymmetry parameter.

`spike` To quote from Wiscombe's '`MIEV0.doc`':

> (REAL) magnitude of the smallest denominator of either Mie coefficient (a-sub-n or b-sub-n), taken over all terms in the Mie series past N = size parameter XX. Values of SPIKE below about 0.3 signify a ripple spike, since these spikes are produced by abnormally small denominators in the Mie coefficients (normal denominators are of order unity or higher). Defaults to 1.0 when not on a spike. Does not identify all resonances (we are still working on that).

Meaningless if a `size_distribution_file` was specified.

`pmom(0:nmom)`

The moments of the phase function. The phase function $p(\mu)$ is

$$p(\mu) = \sum_{m=0}^{\infty} (2m+1) \cdot k_m \cdot P_m(\mu)$$

where k_m is the m'th moment and $P_m(\mu)$ is the m'th Legendre polynomial.

2.2.3 Examples of mie input files

An example of a complete input file is

```
mie_program MIEV0
refrac ice
mimcut 0.0000000001
r_mean 200.
wavelength    280. 5000.
wavelength_step 5.
```

2.3 integrate

`integrate` calculates the integral between limits x_min and x_max by interpolating the data points (x[i], y[i]) with natural cubic splines or linear interpolation. x_min and x_max are the minimum and maximum values of the first column in the input_file. The x-values in the first column must be in ascending order.

The different options to `integrate` are displayed when executing:

```
integrate -h
```

2.4 spline

`spline` interpolates discrete data points using natural cubic splines or linear interpolation. The x-values in the first column must be in ascending order.

The different options to `spline` are displayed when executing:

```
spline -h
```

2.5 conv

`conv` convolutes a spectrum with a given filter function.

The different options to `conv` are displayed when executing:

```
conv -h
```

2.6 addlevel

`addlevel` is a simple shell script to add a level to one of the existing standard profiles.

The different options to `addlevel` are displayed when executing:

```
addlevel -h
```

2.7 snowalbedo

`snowalbedo` calculate the diffuse and direct albedo of snow as formulated by Wiscombe and Warren (1980).

The different options to `snowalbedo` are displayed when executing:

```
snowalbedo -h
```

2.8 cldprp

`cldprp` calculates wavelength-dependent cloud properties using one of several parameterizations.

The different options to `cldprp` are displayed when executing:

```
cldprp -h
```

2.9 Geno3tab

The Perl script `Gen_o3_tab.pl` is used to generate a matrix of ozone values for solar zenith angle versus a chosen ratio of global irradiance at different wavelengths. The table is read by the C program `read_o3_tab` which, for a solar zenith angle and a measured ratio, returns the overhead ozone column. All available options are displayed when executing

```
perl Gen_o3_tab.pl --help
```

and

```
read_o3_tab -h
```

The following different types of tables may be generated.

2.9.1 Simple wavelength ratios with Gen_o3_tab

The simplest type of table is made of ratios of the global irradiance at two single wavelengths. This is the type of table described by Stamnes et al. (1991). This type of table is typically used to analyse measurements of the global irradiance from spectroradiometers. It is generated by the following command (\ is line continuation character)

```
perl Gen_o3_tab.pl --slitfunction slitfncfile --lower_lambda 305. \
                   --upper_lambda 340. --file table.dat
```

Here '`slitfncfile`' is the name of the slit function file. It is a two column file where the first column is the wavelength (nm, in relative units) and the second column holds the slit function. The slit function must be normalized to unity at the center wavelength.

The generated table '`table.dat`' is read by `read_o3_tab` for a measured ratio, `-r 10.0`, and solar zenith angle, `-s 30.0`, corresponding to the modelled ratio in the table

```
read_o3_tab -r 10.0 -s 30.0 table.dat
```

2.9.2 Bandpassed wavelength ratios with Gen_o3_tab

Instead of using single wavelengths it may be of advantage to use ratios of irradiances covering a certain wavelength range and weighted with a bandpass function. This approach may reduce problems due to changes in cloud cover and experimental uncertainties. This approach is also suitable to calculate ozone columns from multichannel, moderate bandwidth filter instruments, Dahlback (1996). Such tables are generated by

```
perl Gen_o3_tab.pl --slitfunction slitfncfile  --lower_lambda 305.0 \
                   --upper_lambda 320.0 --file table.dat \
                   --bandpasslower bplow.dat --bandpassupper bpupp.dat
```

Here '`bplow.dat`' and '`bpupp.dat`' are the bandpass function of the lower and upper wavelength region respectively. The bandpass files have two columns. The first column is the wavelength in nm and relative units to –lower_lambda and –upper_lambda. If absolute units are specified as for filter instruments, use the –absolute option. The second column is the bandpass function.

The tables are read in the same way as the simple wavelength ratio tables.

2.10 Genwctab

The Perl script `Gen_wc_tab.pl` is used to generate a matrix of cloud optical depth for solar zenith angle versus a chosen global irradiance at a selected wavelength. The wavelength should be chosen such that it is not affected by ozone, e.g. 380 nm. The table is read by the C program `read_o3_tab` which, for a solar zenith angle and a measured irradiance, returns the overhead cloud optical depth. All available options are displayed when executing

```
perl Gen_wc_tab.pl --help
```

and

```
read_o3_tab -h
```

The following different types of tables may be generated.

2.10.1 Simple wavelength ratios with Gen_wc_tab

The simplest type of table is made of the global irradiance at a single wavelength. This is the type of table described by Stamnes et al. (1991). This type of table is typically used to analyse measurements of the global irradiance from spectroradiometers. It is generated by the following command (\ is line continuation character)

```
perl Gen_wc_tab.pl --slitfunction slitfncfile --lambda 380. \
                   --file table.dat
```

Here '`slitfncfile`' is the name of the slit function file. It is a two column file where the first column is the wavelength (nm, in relative units) and the second column holds the slit function. The slit function must be normalized to unity at the center wavelength.

The generated table '`table.dat`' is read by `read_o3_tab` for a measured global irradiance, `-r 10.0`, and solar zenith angle, `-s 30.0`, corresponding to the modelled ratio in the

table. The table must be corrected for the Earth–Sun distance for the day of the measurement. This is achieved by specifying `-d 170`, where `170` is the day number. The table is generated for day 1.

```
read_o3_tab -r 10.0 -s 30.0 -d 170 table.dat
```

2.10.2 Bandpassed wavelength ratios with Gen_wc_tab

Instead of using a single wavelength it may be of advantage to use irradiances covering a certain wavelength range and weighted with a bandpass function. This approach may reduce problems due to changes in cloud cover and experimental uncertainties. This approach is also suitable to calculate cloud optical depth from multichannel, moderate bandwidth filter instruments, Dahlback (1996). Such tables are generated by

```
perl Gen_wc_tab.pl --slitfunction slitfncfile  --lambda 380.0 \
                   --file table.dat --bandpass bp.dat
```

Here '`bp.dat`' is the bandpass function of the wavelength region. The bandpass file have two columns. The first column is the wavelength in nm and relative units to –lambda. If absolute units are specified as for filter instruments, use the –absolute option. The second column is the bandpass function.

The tables are read in the same way as the simple wavelength irradiance tables.

3 C functions in libRadtran

3.1 ASCII file access

3.1.1 Usage of the ASCII library

In order to use the functions provided by the ascii library, #include <ascii.h> in your source code and link with libRadtran_c.a.

Example: Example for a source file:

```
...
#include "../src_c/ascii.h"
...
```

Linking of the executable, using the GNU compiler gcc:

```
gcc -o test test.c -lRadtran_c -L../lib
```

3.1.2 General comments to the ASCII library

The ASCII library provides functions for parsing ASCII files containing arrays of data. An ASCII file is read line by line. Each line is split into fields; a field is an arbitrary combination of characters which does neither contain the line separator ('CARRIAGE RETURN') nor the field separator ('SPACE').

In detail:

Lines are separated by 'CARRIAGE RETURN'.

Tokens (or columns) are separated by one or more 'SPACE's.

Empty lines are simply ignored.

\% and # are comment symbols; text between a comment symbol and the next line separator is ignored

A comment symbol which is not at the beginning of a line is only recognized after a field separator, but not within a field

The number of fields may differ from line to line.

A simple example for an ASCII file, which would be recognized as a valid one-column or two-column ASCII file:

```
% This is an example for the input ASCII file for sdose,
% the time integration program
11.0     13.0   % the two hours around noon
```

```
10.0     14.0
 9.0     15.0

# total dose
-1.0     24.0    % integrate over maximum available time interval

# the following line shows, that an extra column does not matter
2  3.4  17
```

For most purposes, ASCII_file2double and ASCII_free_double provide a convenient way for parsing files. **Example:**

```
#include <stdio.h>
#include <ascii.h>

int main(int argc, char ** argv)
{
  int rows=0, max_columns=0, min_columns=0;
  int i=0, status=0;
  double **value=NULL;

  status = ASCII_file2double ("test.dat",
                              &rows,
           &max_columns,
           &min_columns,
           &value);

  for (i=0; i<rows; i++) {
    ... do something for each row of the matrix
 }

  ASCII_free_double (value, rows);

  return 0;
}
```

For special purposes (ASCII files with 1,2,3, or 5 columns) there are additionally functions read_1c_file, ... which facilitate the access even more.

3.1.3 ASCII Library functions

3.1.3.1 Function ASCII_checkfile

`int` **ASCII_checkfile** (`char *filename, int *rows, int *min_columns, int *max_columns, int *max_length`) Function

Description:

Check an ASCII file: count rows, minimum number of columns, maximum number of columns and the maximum length of a string; empty rows and characters after one of the comment symbols (either ASCII_COMMENT_1 or ASCII_COMMENT_2) are ignored.

Parameters:

`char *filename`
name of the file which should be checked

`int *rows` number of rows found

`int *min_columns`
minimum number of columns, set by function

`int *max_columns`
maximum number of columns, set by function

`int *max_length`
maximum length of a field, set by function

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

3.1.3.2 Function ASCII_calloc_string

`int ASCII_calloc_string (char ****string, int rows, int columns, int length)` Function

Description:

Allocate memory for a two-dimensional array of strings.

Parameters:

`char ****string`
Pointer to a two-dimensional array of strings; memory for string is allocated automatically

`int rows` Number of rows, specified by the caller

`int columns`
Number of columns, specified by the caller

`int length`
Maximum length of a string, specified by the caller

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.3 Function ASCII_free_string

int ASCII_free_string (`char ***string, int rows, int columns`) Function

Description:
Free memory, which has been allocated with ASCII_calloc_string.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.4 Function ASCII_calloc_int

int ASCII_calloc_int (`int ***value, int rows, int columns`) Function

Description:
Allocate memory for a two-dimensional array of int.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.5 Function ASCII_calloc_double

`int` **ASCII_calloc_double** (`double ***value, int rows, int columns`) Function

Description:
Allocate memory for a two-dimensional array of double.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.6 Function ASCII_calloc_double_3D

`int` **ASCII_calloc_double_3D** (`double ****value, int rows, int columns, int length`) Function

Description:
Allocate memory for a three-dimensional array of double.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.1.3.7 Function ASCII_calloc_double_3D_arylen

`int` **ASCII_calloc_double_3D_arylen** (`double ****value, int rows, int columns, int *length`) Function

Description:
Allocate memory for a three-dimensional array of double.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.1.3.8 Function ASCII_calloc_double_3D_arylen_restricted

`int` **ASCII_calloc_double_3D_arylen_restricted** (`double ****value, int rows, int columns, int columns_lower, int columns_upper, int *length`) Function

Description:
Allocate memory for a three-dimensional array of double. Only a restricted range (marked by columns_lower and columns_upper) is actually allocated; the function is very special - maybe not of much general use.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.1.3.9 Function ASCII_calloc_float_3D

`int` **ASCII_calloc_float_3D** (`float ****value, int rows, int columns, int length`) Function

Description:
Allocate memory for a three-dimensional array of float.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.1.3.10 Function ASCII_calloc_float_4D

`int` **ASCII_calloc_float_4D** (`float *****value, int rows, int columns, int length, int fourth_dimension`) Function

Description:
Allocate memory for a four-dimensional array of float.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.1.3.11 Function ASCII_calloc_double_4D

`int` **ASCII_calloc_double_4D** (`double *****value, int rows, int columns, int length, int fourth_dimension`) Function

Description:
Allocate memory for a four-dimensional array of double.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.1.3.12 Function ASCII_calloc_float_5D

`int` **ASCII_calloc_float_5D** (`float ******value, int rows, int columns, int length, int fourth_dimension, int fifth_dimension`) Function

Description:
Allocate memory for a five-dimensional array of float.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.1.3.13 Function ASCII_calloc_float

`int` ASCII_calloc_float (`float ***value, int rows, int columns`) Function

Description:
Allocate memory for a two-dimensional array of float.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.14 Function ASCII_free_int

`int` ASCII_free_int (`int **value, int rows`) Function

Description:
Free memory, which has been allocated with ASCII_calloc_int.

Parameters:

`int **value`
Two-dimensional array of int

`int rows` Number of rows, specified by caller

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.15 Function ASCII_free_double

`int` **ASCII_free_double** (`double **value, int rows`) Function

Description:
Free memory, which has been allocated with ASCII_calloc_double.

Parameters:

`double **value`
Two-dimensional array of double

`int rows` Number of rows, specified by caller

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.16 Function ASCII_free_float

`int` **ASCII_free_float** (`float **value, int rows`) Function

Description:
Free memory, which has been allocated with ASCII_calloc_float.

Parameters:

`float **value`
Two-dimensional array of float

`int rows` Number of rows, specified by caller

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.17 Function ASCII_free_double_3D

`int` **ASCII_free_double_3D(double** `***value, int rows, int columns`**)** Function

Description:
Free memory, which has been allocated with ASCII_calloc_double_3D.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.1.3.18 Function ASCII_free_float_3D

`int` **ASCII_free_float_3D(float** `***value, int rows, int columns`**)** Function

Description:
Free memory, which has been allocated with ASCII_calloc_float_3D.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.1.3.19 Function ASCII_free_float_4D

`int` **ASCII_free_float_4D(float** `****value, int rows, int columns, int length`**)** Function

Description:
Free memory, which has been allocated with ASCII_calloc_float_4D.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.1.3.20 Function ASCII_free_double_4D

`int` **ASCII_free_double_4D(double** `****value, int rows, int columns, int length`**)** Function

Description:
Free memory, which has been allocated with ASCII_calloc_double_4D.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.1.3.21 Function ASCII_free_float_5D

`int` **ASCII_free_float_5D(float** `*****value, int rows, int columns, int length, int fourth_dimension`**)** Function

Description:
Free memory, which has been allocated with ASCII_calloc_float_5D.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.1.3.22 Function ASCII_readfile

`int` **ASCII_readfile** (`char *filename, char ***array`) Function

Description:

Read an ASCII file into a two-dimensional array of strings; before calling ASCII_readfile, the file must be parsed with ASCII_checkfile, and memory must be allocated with ASCII_calloc_string.

Parameters:

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

3.1.3.23 Function ASCII_string2double

`int` **ASCII_string2double** (`double **value, char *** string, int rows, int columns`) Function

Description:

Convert a two-dimensional array of strings to a two-dimensional array of double.

Parameters:

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

3.1.3.24 Function ASCII_string2float

`int` **ASCII_string2float** (`float **value, char *** string, int rows, int columns`) Function

Description:

Convert a two-dimensional array of strings to a two-dimensional array of float.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.25 Function ASCII_file2double

`int` **ASCII_file2double** (`char *filename, int *rows, int *max_columns, int *min_columns, double ***value`) Function

Description:
Parse an ASCII file and store data in a twodimensional array value[row][column]; memory allocation for value is done automatically. rows is the number of (not empty) rows of the file, max_columns is the maximal number of columns of the file and min_columns is the minimal number of columns of all (not empty) lines; the dimension of the array value is rows * max_columns; strings that cannot be interpreted as floating point number are converted to 0; rows with less than max_columns columns are filled up with NAN; the allocated memory can be freed with ASCII_free_double (value, rows).

Parameters:

`char *filename`
Name of the file which should be parsed

`int *rows` Number of rows, set by function

`int *min_columns`
Minimum number of columns, set by function

`int *max_columns`
Maximum number of columns, set by function

`double ***value`
Pointer to a two-dimensional array of double, value [0 ... rows-1][0 ... max_columns-1]. Memory is allocated automatically.

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.26 Function ASCII_file2float

`int` **ASCII_file2float** (`char *filename, int *rows, int *max_columns, int *min_columns, float ***value`) Function

Description:

Read an ASCII file and store data in a twodimensional array value[row][column]; memory allocation for value is done automatically. rows is the number of (not empty) rows of the file, max_columns is the maximal number of columns of the file and min_columns is the minimal number of columns of all (not empty) lines; the dimension of the array value is rows * max_columns; strings that cannot be interpreted as floating point number are converted to 0; rows with less than max_columns columns are filled up with NAN; the allocated memory can be freed with ASCII_free_float (value, rows).

Parameters:

`char *filename`
: Name of the file which should be parsed

`int *rows` Number of rows, set by function

`int *min_columns`
: Minimum number of columns, set by function

`int *max_columns`
: Maximum number of columns, set by function

`float ***value`
: Pointer to a two-dimensional array of float, value [0 ... rows-1][0 ... max_columns-1]. Memory is allocated automatically.

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

3.1.3.27 Function ASCII_column

`double` ***ASCII_column** (`double **value, int rows, int column`) Function

Description:

Extract a specified column from a two-dimensional array of double.

Parameters:

Return value:

Pointer to the column.

Example:

Files: none

Known bugs:
none

Author:

3.1.3.28 Function ASCII_column_float

`float` ***ASCII_column_float** (`float **value, int rows, int column`) Function

Description:
Extract a specified column from a two-dimensional array of float.

Parameters:

Return value:
Pointer to the column.

Example:

Files: none

Known bugs:
none

Author:

3.1.3.29 Function ASCII_row

`double` ***ASCII_row** (`double **value, int columns, int row`) Function

Description:
Extract a specified row from a two-dimensional array of double.

Parameters:

Return value:
Pointer to the row.

Example:

Files: none

Known bugs:
none

Author:

3.1.3.30 Function read_1c_file

`int` **read_1c_file** (`char *filename, double **first, int *n`) Function

Description:
Read an ASCII file with (at least) 1 column. Only the first column is returned in array first; n is the number of values returned. Memory allocation for first is done automatically; field can be freed with a simple free().

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.31 Function read_1c_file_float

`int` **read_1c_file_float** (`char *filename, float **first, int *n`) Function

Description:
Read an ASCII file with (at least) 1 column. Only the first column is returned in array first; n is the number of values returned. Memory allocation for first is done automatically; field can be freed with a simple free().

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.32 Function read_2c_file

`int` **read_2c_file** (`char *filename, double **first, double **second, int *n`) Function

Description:
Read an ASCII file with (at least) 2 columns. Only the first two column are returned in arrays first and second. n is the number of values returned. Memory allocation for first and second is done automatically; fields can be freed with a simple free().

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.33 Function read_2c_file_float

`int` **read_2c_file_float** (`char *filename, float **first, float **second, int *n`) Function

Description:
Read an ASCII file with (at least) 2 columns to a float array. Only the first two column are returned in arrays first and second. n is the number of values returned. Memory allocation for first and second is done automatically; fields can be freed with a simple free().

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.34 Function read_3c_file

`int` **read_3c_file** (`char *filename, double **first, double **second, double **third, int *n`) Function

Description:
Read an ASCII file with (at least) 3 columns. Only the first three columns are returned in arrays first, second, and third. n is the number of values returned. Memory allocation for first, second, and third is done automatically; fields can be freed with a simple free().

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.35 Function read_3c_file_float

`int` **read_3c_file_float** (`char *filename, float **first, float **second, float **third, int *n`) Function

Description:
Read an ASCII file with (at least) 3 columns to a float array. Only the first three columns are returned in arrays first, second, and third. n is the number of values returned. Memory allocation for first, second, and third is done automatically; fields can be freed with a simple free().

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.36 Function read_4c_file

`int` **read_4c_file** (`char *filename, double **first, double **second, double **third, double **fourth, int *n`) Function

Description:
Read an ASCII file with (at least) 4 columns. Only the first four columns are returned in arrays first, second, third, and fourth. n is the number of values returned. Memory allocation for first, second, third, and fourth is done automatically; fields can be freed with a simple free().

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.37 Function read_4c_file_float

`int` **read_4c_file_float** (`char *filename, float **first, float **second, float **third, float **fourth, int *n`) Function

Description:
Read an ASCII file with (at least) 4 columns to a float array. Only the first four columns are returned in arrays first, second, third, and fourth. n is the number of values returned. Memory allocation for first, second, third, and fourth is done automatically; fields can be freed with a simple free().

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.38 Function read_5c_file

`int` **read_5c_file** (`char *filename, double **first, double **second, double **third, double **fourth, double **fifth, int *n`) Function

Description:
Read an ASCII file with (at least) 5 columns. Only the first five columns are returned in arrays first, second, third, fourth and fifth. n is the number of values returned. Memory allocation for first, second, third, fourth, and fifth is done automatically; fields can be freed with a simple free().

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.39 Function read_5c_file_float

`int` **read_5c_file_float** (`char *filename, float **first, float **second, float **third, float **fourth, float **fifth, int *n`) Function

Description:

Read an ASCII file with (at least) 5 columns to a float array. Only the first five columns are returned in arrays first, second, third, fourth, and fifth. n is the number of values returned. Memory allocation for first, second, third, and fourth is done automatically; fields can be freed with a simple free().

Parameters:

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

3.1.3.40 Function read_6c_file

`int` **read_6c_file** (`char *filename, double **first, double **second, double **third, double **fourth, double **fifth, double **sixth, int *n`) Function

Description:

Read an ASCII file with (at least) 6 columns. Only the first six columns are returned in arrays first, second, third, fourth, fifth, and sixth. n is the number of values returned. Memory allocation for the result arrays is done automatically; fields can be freed with a simple free().

Parameters:

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

3.1.3.41 Function read_8c_file

`int` **read_8c_file** (`char *filename, double **first, double **second, double **third, double **fourth, double **fifth, double **sixth, double **seventh, double **eighth, int *n`) Function

Description:

Read an ASCII file with (at least) 8 columns. Only the first eight columns are returned in arrays first, second, third, fourth, fifth, sixth, seventh, and eigths. n is the number of values returned. Memory allocation for the result arrays is done automatically; fields can be freed with a simple free().

Parameters:

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

3.1.3.42 Function read_9c_file

`int` **read_9c_file** (`char *filename, double **first, double **second, double **third, double **fourth, double **fifth, double **sixth, double **seventh, double **eighth, double **ninth, int *n`) Function

Description:

Read an ASCII file with (at least) 9 columns. Only the first eight columns are returned in arrays first, second, third, fourth, fifth, sixth, seventh, eigths, and ninth. n is the number of values returned. Memory allocation for the result arrays is done automatically; fields can be freed with a simple free().

Parameters:

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

3.1.3.43 Function substr

`char` ***substr(char** `*buffer, char *string, int start, int length`) Function

Description:
Create substring starting at position start with length length. Result is written to buffer (which MUST be allocated before).

Parameters:

Return value:
Pointer to the substring.

Example:

Files: none

Known bugs:
none

Author:

3.1.3.44 Function ASCII_parse

`int` **ASCII_parse** (`char *string, char *separator, char ***array, int *number`) Function

Description:
Parse string to an array of single words. Memory for an array of string pointers is allocated automatically. array[i] points to the address of word #i in string! Word separator is specified in separator. Number of words is returned in number. Characters following the comment character are ignored.

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.1.3.45 Function ASCII_parsestring

`int` **ASCII_parsestring** (`char *string, char ***array, int *number`) Function

Description:
For compatibility reasons: ASCII_parsestring is just a call to ASCII_parse() with field separator set to " \t\v\f\r\n"

Parameters:

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.2 Numeric functions

3.2.1 Usage of the Numeric Library

In order to use the functions provided by the numeric library, #include <numeric.h> in your source code and link with libRadtran_c.a.

Example: Example for a source file:

```
...
#include "../src_c/numeric.h"
...
```

Linking of the executable, using the GNU compiler gcc:

```
gcc -o test test.c -lRadtran_c -L../lib
```

3.2.2 General comments to the Numeric Library

The numeric library provides various numeric functions. The source code is split up into the following source files:

cnv.c Data convolution.

equation.c Solve systems of linear equations.

function.c Miscellaneous functions.

integrat.c Numerical integration.

linear.c Linear interpolation.

regress.c Linear regression and related things.

spl.c Spline interpolation and approximation.

3.2.3 Numeric Library functions

3.2.3.1 Function convolute

int convolute (`double *x_spec, double *y_spec, int spec_num, double *x_conv, double *y_conv, int conv_num, double **x_spec_conv, double **y_spec_conv, int *spec_conv_num`) Function

Description:

Convolute a dataset (x_spec[i], y_spec[i]) with another data set (x_conv[i], y_conv[i]). The data sets must obey the following principles: (1) Both datasets must be defined in equidistant steps; (2) the step width must be the same for both datasets; and (3) 0 must be a point of the grid of the convolution function, x_conv[]. The results is stored in (x_spec_conv[i], y_spec_conv[i]), i=0...spec_conv_num, the memory of which is allocated automatically.

Parameters:

`double *x_spec`
x values of the data points, i=0...spec_num-1

`double *y_spec`
y values of the data points, i=0...spec_num-1

`int spec_num`
number of data points

`double *x_conv`
x values of the convolution function, i=0...conv_num-1

`double *y_conv`
y values of the convolution function, i=0...conv_num-1

`int conv_num`
number of convolution function data points

`double **x_spec_conv`
x values of the convoluted spectrum

`double **y_spec_conv`
y values of the convoluted spectrum

`int spec_conv_num`
number of result data points

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.2.3.2 Function int_convolute

int int_convolute (`double *x_spc, double *y_spc, int spc_num, double *x_conv, double *y_conv, int conv_num, double **y_spec_conv`) Function

Description:

Convolute a dataset (x_spc[i], y_spc[i]) with another data set (x_conv[i], y_conv[i]). In contrast to the conv() function provided by this library, the dataset to be convoluted and the convolution function may be defined on different grids. While x_spc[] may be an arbitrary grid, x_conv[] must obey the following principles: (1) The spacing must be equidistant; and (2) 0 must be part of the grid. During the convolution process, the convolution function is interpolated to the grid defined by x_conv[]. It is therefore necessary to specify a fine enough grid x_conv[], even if it only describes, e.g., a triangle. The output is stored in y_spec_conv[], which is evaluated at the original data points x_spc[].

Parameters:

`double *x_spc`
: x values of the data points, i=0...spc_num-1

`double *y_spc`
: y values of the data points, i=0...spc_num-1

`int spc_num`
: number of data points

`double *x_conv`
: x values of the convolution function, i=0...conv_num-1

`double *y_conv`
: y values of the convolution function, i=0...conv_num-1

`int conv_num`
: number of convolution function data points

`double **y_spec_conv`
: convoluted spectrum on the original grid x_spc, i=0...spc_num-1

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.2.3.3 Function solve_gauss

`int` **solve_gauss** (`double **A, double *b, int n, double **res`) Function

Description:

Solve a system of n linear equations, A*x = b, using the Gauss algorithm The pivot element is determined using 'relative column maximum strategy'. For a description of the algorithm see H.R.Schwarz: "Numerische Mathematik", pg. 21. Memory for the result vector res is allocated automatically.

Parameters:

`double **A`
: Matrix[n x n] (see above).

`double *b` Vector[n] (see above).

`double n` Number of equations.

`double **res`
: Pointer to the result vector[n]; if no unique solution exists, *res will be set to NULL.

Return value:

0 if o.k., <0 if no unique solution.

Example:

Files: none

Known bugs:

none

Author:

3.2.3.4 Function solve_five

`int` **solve_five** (`double **A, double *b, int n, double **res`) Function

Description:

Solve a system of n linear equations, A*x = b, where A is a five-diagonal matrix; for details see Engeln-Muellges, pg.95ff. Memory for the result vector is allocated automatically. If possible, solve_five_ms() should be preferred to solve_five(), because much less memory is required by the latter.

Parameters:

`double **A`
: Matrix[n x n] (see above).

`double *b` Vector[n] (see above).

`double n` Number of equations.

`double **res`
: Pointer to the result vector[n]; if no unique solution exists, *res will be set to NULL.

Return value:
0 if o.k., <0 if no unique solution.

Example:

Files: none

Known bugs:
none

Author:

3.2.3.5 Function solve_five_ms

`int` **solve_five_ms** (`double **A, double *b, int n, double **res`) Function

Description:
Same function as solve_five(), but much more memory efficient on the cost of some user-friendlyness. In contrast to solve_five(), A is defined as a n*5-matrix, rather than a n*n-matrix.

Parameters:

`double **A`
Matrix[n x 5] (see above).

`double *b` Vector[n] (see above).

`double n` Number of equations.

`double **res`
Pointer to the result vector[n]; if no unique solution exists, *res will be set to NULL.

Return value:
0 if o.k., <0 if no unique solution.

Example:

Files: none

Known bugs:
none

Author:

3.2.3.6 Function solve_three

`int` **solve_three** (`double **A, double *b, int n, double **res`) Function

Description:
Solve a system of n linear equations, A*x = b, where A is a three-diagonal matrix; for details see Engeln-Muellges, pg.95ff. Memory for the result vector is allocated automatically. If possible, solve_three_ms() should be preferred to solve_three(), because much less memory is required by the latter.

Parameters:

`double **A`
: Matrix[n x n] (see above).

`double *b` Vector[n] (see above).

`double n` Number of equations.

`double **res`
: Pointer to the result vector[n]; if no unique solution exists, *res will be set to NULL.

Return value:
: 0 if o.k., <0 if no unique solution.

Example:

Files: none

Known bugs:
: none

Author:

3.2.3.7 Function solve_three_ms

int solve_three_ms (`double **A, double *b, int n, double **res`) Function

Description:
: Same function as solve_three(), but much more memory efficient on the cost of some user-friendlyness. In contrast to solve_three(), A is defined as a n*3-matrix, rather than a n*n-matrix.

Parameters:

`double **A`
: Matrix[n x 3] (see above).

`double *b` Vector[n] (see above).

`double n` Number of equations.

`double **res`
: Pointer to the result vector[n]; if no unique solution exists, *res will be set to NULL.

Return value:
: 0 if o.k., <0 if no unique solution.

Example:

Files: none

Known bugs:
: none

Author:

3.2.3.8 Function fsolve_three_ms

int fsolve_three_ms (`float **A, float *b, int n, float **res`) Function

Description:

Similar to solve_three_ms, only all data are of type float instead of float. Same function as solve_three(), but much more memory efficient on the cost of some user-friendlyness. In contrast to solve_three(), A is defined as a n*3-matrix, rather than a n*n-matrix.

Parameters:

`float **A` Matrix[n x 3] (see above).

`float *b` Vector[n] (see above).

`float n` Number of equations.

`float **res`
Pointer to the result vector[n]; if no unique solution exists, *res will be set to NULL.

Return value:

0 if o.k., <0 if no unique solution.

Example:

Files: none

Known bugs:

none

Author:

3.2.3.9 Function double_equal

int double_equal (`double a, double b`) Function

Description:

Compare two float values; returns 0, if the relative difference is bigger than DOUBLE_RELATIVE_ERROR, which is 1E-10 here. The intention of this function is to avoid roundoff errors when comparing two floats.

Parameters:

`double a` First float to be compared.

`double b` Second float to be compared.

Return value:

1 if 'equal', 0 if not equal.

Example:

Files: none

Known bugs:

none

Author:

3.2.3.10 Function sort_long

`void` **sort_long** (`long *x1, long *x2`) Function

Description:
Sort two long integers in ascending order.

Parameters:

`long *x1` Pointer to first integer.

`long *x2` Pointer to second integer.

Return value:
None.

Example:

Files: none

Known bugs:
none

Author:

3.2.3.11 Function sort_double

`void` **sort_double** (`double *x1, double *x2`) Function

Description:
Sort two doubles in ascending order.

Parameters:

`double *x1`
Pointer to first double.

`double *x2`
Pointer to second double.

Return value:
None.

Example:

Files: none

Known bugs:
none

Author:

3.2.3.12 Function fak

`double` **fak** (`long n`) Function

Description:

Calculate the faculty n! of an integer number n.

Parameters:

`long n` Function input.

Return value:

Result n!

Example:

Files: none

Known bugs:

none

Author:

3.2.3.13 Function integrate

`double` **integrate** (`double *x_int, double *y_int, int number`) Function

Description:

Integrate a function which is defined at discrete x-values over the whole defined range. x values must be sorted in ascending order. ATTENTION: integrate does not check this, but will rather return a wrong result. integrate approximates the integral with a trapezoidal rule.

Parameters:

`double *x_int`
x values of the data points.

`double *y_int`
y values of the data points.

`int number`
Number of data points.

Return value:

Integral of the function over the whole range (double).

Example:

Files: none

Known bugs:

none

Author:

3.2.3.14 Function integrate_spline

`int` **integrate_spline** (`double *x, double *y, int number, double a, double b, double *integral`) Function

Description:

Calculate integral y(x) dx between a and b by interpolating the data points (x[i], y[i]) with natural cubic splines.

Parameters:

`double *x` Vector[0..number-1] of x-values.

`double *x` Vector[0..number-1] of y-values.

`int number`
Number of data points (x[i],y[i]).

`double a` Integration start.

`double b` Integration end.

`double *intgral`
Calculated integral.

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

3.2.3.15 Function linear_coeffc

`int` **linear_coeffc** (`double *x, double *y, int number, double **a0, double **a1, double **a2, double **a3`) Function

Description:

Calculate coefficients for linear interpolation; memory for coefficients will be allocated automatically! These function has been created for compatibility with the spline interpolation functions; for this reason four coefficients are calculated, but a2[] and a3[] are set to zero. The interpolation may be done with calc_spline_values().

Parameters:

`double *x` x values of the data points, i=0...number-1

`double *y` y values of the data points, i=0...number-1

`int number`
number of datapoints

`double **a0`
: array of coefficients, i=0...number-1

`double **a1`
: array of coefficients, i=0...number-1

`double **a2`
: array of coefficients, i=0...number-1, set to zero

`double **a3`
: array of coefficients, i=0...number-1, set to zero

Return value:
: 0 if o.k., <0 if error

Example:

Files: none

Known bugs:
: none

Author:

3.2.3.16 Function slinear_coeffc

`int slinear_coeffc` (`double *x, double *y, int number, double **a0, double **a1, double **a2, double **a3`) Function

Description:
: Like linear_coeffc(), but sort data before calculating coefficients. Please note that the fields x and y themselves are sorted by slinear_coeffc() which is an important pre-requisite when they are passed to calc_splined_value() later.

Parameters:

`double *x` x values of the data points, i=0...number-1

`double *y` y values of the data points, i=0...number-1

`int number`
: number of datapoints

`double **a0`
: array of coefficients, i=0...number-1

`double **a1`
: array of coefficients, i=0...number-1

`double **a2`
: array of coefficients, i=0...number-1, set to zero

`double **a3`
: array of coefficients, i=0...number-1, set to zero

Return value:
: 0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.2.3.17 Function gauss

double gauss (double x, double mu, double sigma) Function

Description:
Calculate a Gauss function for given average and standard deviation.

Parameters:

double x x value where the Gauss function is to be evaluated.

double mu Average.

double sigma
Standard deviation.

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.2.3.18 Function regression

int regression (double *x, double *y, int n, double *a, double *b, double *sigma_a, double *sigma_b, double *correlation) Function

Description:
Calculate coefficients for y = a + b*x by linear regression.

Parameters:

double *x Vector (0..n-1) of x-values.

double *y Vector (0..n-1) of y-values.

int n Number of data points.

double *a Coefficient a.

double *b Coefficient b.

`double *sigma_a`
: Standard deviation of a.

`double *sigma_b`
: Standard deviation of b.

`double *correlation`
: Correlation coefficient.

Return value:
: 0 if o.k., <0 if error

Example:

Files: none

Known bugs:
: none

Author:

3.2.3.19 Function weight_regression

`double` **weight_regression** (`double *x, double *y, double *sigma, int n, double *a, double *b, double *sigma_a, double *sigma_b`) Function

Description:
: Calculate coefficients for y = a + b*x by *weighted* linear regression. Each data point is weighted with (1 / sigma[i]**2)

Parameters:

`double *x` Vector (0..n-1) of x-values.

`double *y` Vector (0..n-1) of y-values.

`double *sigma`
: Vector (0..n-1) of weighting coefficients.

`int n` Number of data points.

`double *a` Coefficient a.

`double *b` Coefficient b.

`double *sigma_a`
: Standard deviation of a.

`double *sigma_b`
: Standard deviation of b.

`double *correlation`
: Correlation coefficient.

Return value:
: 0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.2.3.20 Function spline

`int` **spline** (`double *x, double *y, int number, double start, double step, int *newnumber, double **new_x, double **new_y`) Function

Description:
Interpolate between given data points using natural cubic splines. The input data (x,y) are interpolated to an equidistant grid (start, step). Memory for result vectors will be allocated automatically.

Parameters:

`double *x` x[0..number-1], x-values of the input data.

`double *y` y[0..number-1], y-values of the input data.

`int number`
Number of input data points (x[i], y[i]).

`double start`
Start point for the output grid.

`double step`
Step of the output grid; data are interpolated to start, start+step, start+2*step, ...

`int *newnumber`
Number of output data points.

`double *x_new`
x_new[0..newnumber], x-values of the interpolated data points.

`double *y_new`
y_new[0..newnumber], y-values of the interpolated data points.

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.2.3.21 Function spline_coeffc

`int` **spline_coeffc** (`double *x, double *y, int number, double **a0, double **a1, double **a2, double **a3`) Function

Description:

Calculate coefficients for natural cubic spline. These coefficients may be used as input to calc_splined_value() in order to interpolate the data points at arbitrary x values. Memory for the vectors a0, a1, a2, and a3 will be allocated automatically.

Parameters:

`double *x` x[0..number-1], x-values of the input data.

`double *y` y[0..number-1], y-values of the input data.

`int number`
Number of input data points (x[i], y[i]).

`double **a0`
Pointer to vector of 0th order spline coefficients.

`double **a1`
Pointer to vector of 1st order spline coefficients.

`double **a2`
Pointer to vector of 2nd order spline coefficients.

`double **a3`
Pointer to vector of 3rd order spline coefficients.

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

3.2.3.22 Function fspline_coeffc

`int` **fspline_coeffc** (`float *x, float *y, int number, float **a0, float **a1, float **a2, float **a3`) Function

Description:

Similar to spline_coeffc, only all data are of type float instead of double. Calculate coefficients for natural cubic spline. These coefficients may be used as input to calc_splined_value() in order to interpolate the data points at arbitrary x values. Memory for the vectors a0, a1, a2, and a3 will be allocated automatically.

Parameters:

`float *x` x[0..number-1], x-values of the input data.

`float *y` y[0..number-1], y-values of the input data.

`int number`
Number of input data points (x[i], y[i]).

`float **a0`
Pointer to vector of 0th order spline coefficients.

`float **a1`
Pointer to vector of 1st order spline coefficients.

`float **a2`
Pointer to vector of 2nd order spline coefficients.

`float **a3`
Pointer to vector of 3rd order spline coefficients.

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.2.3.23 Function appspl

`int` **appspl(double** `*x, double *y, double *w, int number, double start, double step, int *newnumber, double **new_x, double **new_y`**)** Function

Description:
Approximate data points using natural cubic splines. The input data (x,y) are interpolated to an equidistant grid (start, step). Memory for result vectors will be allocated automatically.

Parameters:

`double *x` x[0..number-1], x-values of the input data.

`double *y` y[0..number-1], y-values of the input data.

`double *w` y[0..number-1], weighting coefficients. Increasing w will increase the degree of approximation; extreme cases are w=0 (interpolating spline) and w=infinity (linear regression).

`int number`
Number of input data points (x[i], y[i]).

`double start`
Start point for the output grid.

`double step`
: Step of the output grid; data are interpolated to start, start+step, start+2*step, ...

`int *newnumber`
: Number of output data points.

`double *x_new`
: x_new[0..newnumber], x-values of the interpolated data points.

`double *y_new`
: y_new[0..newnumber], y-values of the interpolated data points.

Return value:
: 0 if o.k., <0 if error

Example:

Files: none

Known bugs:
: none

Author:

3.2.3.24 Function appspl_coeffc

`int appspl_coeffc` (`double *x, double *y, double *w, int number, double **a0, double **a1, double **a2, double **a3`) Function

Description:
: Calculate coefficients for approximating cubic spline. These coefficients may be used as input to calc_splined_value() in order to interpolate the data points at arbitrary x values. Memory for the vectors a0, a1, a2, and a3 will be allocated automatically.

Parameters:

`double *x` x[0..number-1], x-values of the input data.

`double *y` y[0..number-1], y-values of the input data.

`double *w` w[0..number-1], weighting coefficients. Increasing w will increase the degree of approximation; extreme cases are w=0 (interpolating spline) and w=infinity (linear regression).

`int number`
: Number of input data points (x[i], y[i]).

`double **a0`
: Pointer to vector of 0th order spline coefficients.

`double **a1`
: Pointer to vector of 1st order spline coefficients.

`double **a2`
: Pointer to vector of 2nd order spline coefficients.

`double **a3`
: Pointer to vector of 3rd order spline coefficients.

Return value:
: 0 if o.k., <0 if error

Example:

Files: none

Known bugs:
: none

Author:

3.2.3.25 Function calc_splined_value

`int` **calc_splined_value** (`double xnew, double *ynew, double *x, int number, double *a0, double *a1, double *a2, double *a3`) Function

Description:
: Interpolate/approximate data to x_new. Basis are the spline coefficients which have been determined either by spline_coeffc or by appspl_coeffc, or even by linear_coeffc.

Parameters:

`double xnew`
: x-value to be interpolated.

`double *ynew`
: Pointer to the calculated y-value.

`double *x` x[0..number-1], original x-values.

`int number`
: Number of original x-values.

`double **a0`
: Pointer to vector of 0th order spline coefficients.

`double **a1`
: Pointer to vector of 1st order spline coefficients.

`double **a2`
: Pointer to vector of 2nd order spline coefficients.

`double **a3`
: Pointer to vector of 3rd order spline coefficients.

Return value:
: 0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.2.3.26 Function fcalc_splined_value

int fcalc_splined_value (`float xnew, float *ynew, float *x, int number, float *a0, float *a1, float *a2, float *a3`) Function

Description:
Similar to calc_splined_value, only all data are of type float instead of double. Interpolate/approximate data to x_new. Basis are the spline coefficients which have been determined by fspline_coeffc.

Parameters:

`float xnew`
x-value to be interpolated.

`float *ynew`
Pointer to the calculated y-value.

`float *x` x[0..number-1], original x-values.

`int number`
Number of original x-values.

`float **a0`
Pointer to vector of 0th order spline coefficients.

`float **a1`
Pointer to vector of 1st order spline coefficients.

`float **a2`
Pointer to vector of 2nd order spline coefficients.

`float **a3`
Pointer to vector of 3rd order spline coefficients.

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.2.3.27 Function linear_eqd

`int` **linear_eqd** (`double *x, double *y, int number, double start, double step, int *newnumber, double **new_x, double **new_y`) Function

Description:

Interpolate linearely between given data points (x[i], y[i]). The input data (x,y) are interpolated to an equidistant grid (start, step). Memory for result vectors will be allocated automatically. The syntax of linear_eqd is identical to spline.

Parameters:

`double *x` x[0..number-1], x-values of the input data.

`double *y` y[0..number-1], y-values of the input data.

`int number`
Number of input data points (x[i], y[i]).

`double start`
Start point for the output grid.

`double step`
Step of the output grid; data are interpolated to start, start+step, start+2*step, ...

`int *newnumber`
Number of output data points.

`double *x_new`
x_new[0..newnumber], x-values of the interpolated data points.

`double *y_new`
y_new[0..newnumber], y-values of the interpolated data points.

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

3.3 Fortran/C array functions

3.3.1 Usage of the Fortran/C array functions

In order to use the functions provided by the fortran_and_c library, #include <fortran_and_c.h> in your source code and link with libRadtran_c.a.

Example: Example for a source file:

```
...
#include <fortran_and_c.h>
...
```

Linking of the executable, using the GNU compiler gcc:

```
gcc -o test test.c -lRadtran_c -L../lib
```

3.3.2 General comments to the Fortran/C functions

The fortran_and_c library provides functions converting from multidimensional C arrays to one-dimensional fortran arrays that can be input to fortran routines, and functions for converting one-dimensional fortran compatible arrays to multidimensional C arrays.

These functions are useful when calling fortran functions and subroutines from C.

3.3.3 Fortran/C Library functions

3.3.3.1 Function c2fortran_2D_float_ary

`float` ***c2fortran_2D_float_ary** (`int dim1, int dim2, float** ary_2D`) Function

Description:

Convert 2D C float array to 1D column float array and map it such that it can be passed to a fortran routine.

Parameters:

`int dim1` first dimension of C array

`int dim2` second dimension of C array

`float **ary_2D`
pointer to the 2D C array

Return value:

Pointer to the 1D fortran compatible array. Memory allocation is done automatically and can be freed with a simple free().

Example:

Files: none

Known bugs:

none

Author: Arve Kylling

3.3.3.2 Function fortran2c_2D_float_ary

`float **`**fortran2c_2D_float_ary** (`int dim1, int dim2, float* ary_1D`) Function

Description:
Convert 1D fortran compatible float array to 2D C float array. Space for the returned ary is automatically allocated.

Parameters:

`int dim1` first dimension of C array

`int dim2` second dimension of C array

`float *ary_1D`
pointer to the 1D fortran compatible array

Return value:
Pointer to the 2D C array. Memory allocation is done automatically.

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.3.3.3 Function fortran2c_2D_float_ary_noalloc

`void` **fortran2c_2D_float_ary_noalloc** (`int dim1, int dim2, float* ary_1D, float** ary_2D`) Function

Description:
Convert 1D fortran compatible float array to a pre-allocated 2D C float array.

Parameters:

`int dim1` first dimension of C array

`int dim2` second dimension of C array

`float *ary_1D`
pointer to the 1D fortran compatible array

`float *ary_2D`
pointer to the 2D C compatible array

Return value:

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.3.3.4 Function fortran2c_2D_double_ary_noalloc

`void` **fortran2c_2D_double_ary_noalloc** (`int dim1, int dim2, double* ary_1D, double** ary_2D`) Function

Description:
Convert 1D fortran compatible double array to 2D C double array.

Parameters:

`int dim1` first dimension of C array

`int dim2` second dimension of C array

`double *ary_1D`
pointer to the 1D fortran compatible array

Return value:
Pointer to the 2D C array. Memory allocation is done automatically.

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.3.3.5 Function fortran2c_2D_double_ary

`double **`**fortran2c_2D_double_ary** (`int dim1, int dim2, double* ary_1D`) Function

Description:
Convert 1D fortran compatible double array to 2D C double array. Space for the returned ary is automatically allocated.

Parameters:

`int dim1` first dimension of C array

`int dim2` second dimension of C array

`double *ary_1D`
pointer to the 1D fortran compatible array

Return value:
Pointer to the 2D C array. Memory allocation is done automatically.

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.3.3.6 Function fortran2c_3D_float_ary_noalloc

`void` **fortran2c_3D_float_ary_noalloc** (`int dim1, int dim2, int dim3, float *ary_1D, float ***ary_3D`) Function

Description:
Convert 1D fortran compatible float array to a pre-allocated 3D C float array.

Parameters:

`int dim1` first dimension of C array

`int dim2` second dimension of C array

`float *ary_1D`
pointer to the 1D fortran compatible array

Return value:

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.3.3.7 Function fortran2c_3D_float_ary

`float ***`**fortran2c_3D_float_ary** (`int dim1, int dim2, int dim3, float* ary_1D`) Function

Description:
Convert 1D fortran compatible float array to 3D C float array. Space for the returned ary is automatically allocated.

Parameters:

`int dim1` first dimension of C array

`int dim2` second dimension of C array

`float *ary_1D`
pointer to the 1D fortran compatible array

Return value:
Pointer to the 3D C array. Memory allocation is done automatically.

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.3.3.8 Function fortran2c_4D_float_ary

`float ****`**fortran2c_4D_float_ary** (`int dim1, int dim2, int dim3, int dim4, float* ary_1D`) Function

Description:
Convert 1D fortran compatible float array to 4D C float array. Space for the returned ary is automatically allocated.

Parameters:

`int dim1` first dimension of C array

`int dim2` second dimension of C array

`float *ary_1D`
pointer to the 1D fortran compatible array

Return value:
Pointer to the 4D C array. Memory allocation is done automatically.

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.3.3.9 Function fortran2c_4D_double_ary_noalloc

`void` **fortran2c_4D_double_ary_noalloc** (`int dim1, int dim2, int dim3, int dim4, double* ary_1D, double ****ary_4D`) Function

Description:
Convert 1D fortran compatible double array to 4D C double array.

Parameters:

`int dim1` first dimension of C array

`int dim2` second dimension of C array

`double *ary_1D`
pointer to the 1D fortran compatible array

Return value:
Pointer to the 4D C array. Memory allocation is done automatically.

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.3.3.10 Function fortran2c_4D_double_ary

`double ****fortran2c_4D_double_ary (int dim1, int dim2, int dim3, int dim4, double* ary_1D)` Function

Description:
Convert 1D fortran compatible double array to 4D C double array. Space for the returned ary is automatically allocated.

Parameters:

`int dim1` first dimension of C array

`int dim2` second dimension of C array

`double *ary_1D`
pointer to the 1D fortran compatible array

Return value:
Pointer to the 4D C array. Memory allocation is done automatically.

Example:

Files: none

Known bugs:
none

Author: Arve Kylling

3.4 Sun-Earth Astronomical Relationship

3.4.1 Usage of the Sun library

In order to use the functions provided by the sun library, #include <sun.h> in your source code and link with libRadtran_c.a.

Example: Example for a source file:

```
...
#include "../src_c/sun.h"
...
```

Linking of the executable, using the GNU compiler gcc:

```
gcc -o test test.c -lRadtran_c -L../lib
```

3.4.2 General comments to the Sun Library

The sun library provides functions for solar zenith and azimuth angle and sun-earth-distance calculations. All formulas have been taken from Iqbal, "An introduction to solar radiation".

3.4.3 Sun Library functions

3.4.3.1 Function eccentricity

`double` **eccentricity** (`int day`) Function

Description:

Calculate the eccentricity correction factor E0 = (r0/r)**2 according to Iqbal, page 3. This factor, when multiplied with the irradiance, accounts for the annual variation of the sun-earth-distance.

Parameters:

`int day` day of year (leap day is usually **not** counted.

Return value:

The eccentricity (double) for the specified day.

Example:

Files: none

Known bugs:

none

Author:

3.4.3.2 Function declination

`double` **declination** (`int day`) Function

Description:

Calculate the declination for a specified day (Iqbal, page 7).

Parameters:

`int day` day of year (leap day is usually **not** counted.

Return value:

The declination in degrees (double) for the specified day.

Example:

Files: none

Known bugs:

none

Author:

3.4.3.3 Function equation_of_time

int equation_of_time (int day) Function

Description:
Calculate the equation of time for a specified day (Iqbal, page 11).

Parameters:

int day day of year (leap day is usually **not** counted.

Return value:
The equation of time in seconds (double) for the specified day.

Example:

Files: none

Known bugs:
none

Author:

3.4.3.4 Function LAT

int LAT (int time_std, int day, double longitude, double long_std) Function

Description:
Calculate the local apparent time for a given standard time and location.

Parameters:

int time_std
Standard time [seconds since midnight].

int day day of year (leap day is usually **not** counted.

double longitude
Longitude [degrees] (West positive).

double long_std
Standard longitude [degrees].

Return value:
The local apparent time in seconds since midnight (double).

Example:

Files: none

Known bugs:
none

Author:

3.4.3.5 Function solar_zenith

`double` **solar_zenith** (`int time, int day, double latitude, double longitude, double long_std`) Function

Description:
Calculate the solar zenith angle for a given time and location.

Parameters:

`int time` Standard time [seconds since midnight].

`int day` day of year (leap day is usually **not** counted.

`double latitude`
Latitude [degrees] (North positive).

`double longitude`
Longitude [degrees] (West positive).

`double long_std`
Standard longitude [degrees].

Return value:
The solar zenith angle [degrees].

Example:

Files: none

Known bugs:
none

Author:

3.4.3.6 Function solar_azimuth

`double` **solar_azimuth** (`int time, int day, double latitude, double longitude, double long_std`) Function

Description:
Calculate the solar azimuth angle for a given time and location.

Parameters:

`int time` Standard time [seconds since midnight].

`int day` day of year (leap day is usually **not** counted.

`double latitude`
Latitude [degrees] (North positive).

`double longitude`
Longitude [degrees] (West positive).

`double long_std`
Standard longitude [degrees].

Return value:
The solar azimuth angle [degrees].

Example:

Files: none

Known bugs:
none

Author:

3.4.3.7 Function day_of_year

int day_of_year (int day, int month) Function

Description:
Calculate the day of year for given date (leap days are not considered.

Parameters:

`int day` Day of month (1..31).

`int month` Month (1..12).

Return value:
The day of year (int); -1 if error.

Example:

Files: none

Known bugs:
none

Author:

3.4.3.8 Function zenith2time

int zenith2time (int day, double zenith_angle, double latitude, double longitude, double long_std, int *time1, int *time2) Function

Description:
Calculate the times for a given solar zenith angle, day of year and location.

Parameters:

`int day` day of year

`double zenith_angle`
Solar zenith angle [degrees].

`double latitude`
Latitude [degrees] (North positive).

`double longitude`
Longitude [degrees] (West positive).

`double long_std`
: Standard longitude [degrees].

`int *time1`
: 1st time of occurence.

`int *time2`
: 2nd time of occurence.

Return value:
: 0 if o.k., <0 if error.

Example:

Files: none

Known bugs:
: none

Author:

3.4.3.9 Function Gregorian2Julian

`int` Gregorian2Julian (`int d, int m, int y, int *jd`) Function

Description:
: Convert from Gregorian day (day, month, year) to Julian day (by the astronomical definition). This function, in combination with Julian2Gregorian() is very useful to answer questions like "which date is 666 days after December 31, 1999?" Algorithm from H.F. Fliegel and T.C. Van Flandern, "A Machine Algorithm for Processing Calendar Dates", Communications of the Association for Computing Machinery (CACM), Vol. 11, No. 10, 657, 1968.

Parameters:

`int d` Day of month (1..31).

`int m` Month (1..12).

`int y` Year (attention

`int *jd` The Julian day, to be calculated.

Return value:
: 0 if o.k., <0 if error.

Example:

Files: none

Known bugs:
: none

Author:

3.4.3.10 Function Julian2Gregorian

int Julian2Gregorian(int `*d, int *m, int *y, int jd`**)** Function

Description:

Convert from Julian day (by the astronomical definition) to Gregorian day (day, month, year) to . This function, in combination with Gregorian2Julian() is very useful to answer questions like "which date is 666 days after December 31, 1999?" Algorithm from H.F. Fliegel and T.C. Van Flandern, "A Machine Algorithm for Processing Calendar Dates", Communications of the Association for Computing Machinery (CACM), Vol. 11, No. 10, 657, 1968.

Parameters:

`int *d` Day of month (1..31), to be calculated.

`int *m` Month (1..12), to be calculated.

`int *y` Year, to be calculated.

`int jd` The Julian day.

Return value:

0 if o.k., <0 if error.

Example:

Files: none

Known bugs:

none

Author:

3.4.3.11 Function location

int location (`char *locstr, double *latitude, double *longitude, double *long_std`) Function

Description:

Return latitude, longitude, and standard longitude for a given location.

Parameters:

`double *latitude`
Latitude (North positive).

`double *longitude`
Longitude (West positive).

`double *long_std`
Standard longitude (West positive).

`char *location`
String identifying the location.

Return value:
0 if o.k., <0 if error.

Example:

Files: none

Known bugs:
none

Author:

3.5 Mie calculations

3.5.1 Usage of the Mie library

In order to use the functions provided by the mie library, #include <miecalc.h> in your source code and link with libRadtran_c.a.

Example: Example for a source file:

```
...
#include "../src_c/miecalc.h"
...
```

Linking of the executable, using the GNU compiler gcc:

```
gcc -o test test.c -lRadtran_c
```

3.5.2 General comments to the Mie Library

The Mie library provides functions for Mie calculations, interfacing the MIEV0 and BH-MIE codes by Warren Wiscombe (ftp://climate.gsfc.nasa.gov/pub/wiscombe) and Bohren and Huffman (ftp://astro.princeton.edu/draine/scat/bhmie/). Functions for evaluating the phase function and the integrated phase function are also provided.

3.5.3 Mie Library functions

3.5.3.1 Function mie_calc

`int` **mie_calc** (`mie_inp_struct input, mie_out_struct *output, int program, int medium, mie_complex crefin, float wavelength, float radius, float temperature, mie_complex *ref`) Function

Description:

Mie calculations, using the MIEV0 and BHMIE codes by Warren Wiscombe (ftp://climate.gsfc.nasa.gov/pub/wiscombe) and Bohren and Huffman (ftp://astro.princeton.edu/draine/scat/bhmie/).

Parameters:

`mie_inp_struct input`
: mie input structure (see src_c/miecalc.h)

`mie_out_struct *output`
: mie output structure (see src_c/miecalc.h)

`int program`
: MIEV0 or BHMIE

`int medium`
: WATER, ICE, or USER; if USER, the refractive index is read from crefin

`mie_complex crefin`
: Complex refractive index (both numbers positive)

`float wavelength`
: Wavelength [micron]

`float radius`
: Droplet radius [micron]

`float temperature`
: Temperature [K]

`mie_complex *ref`
: Complex index of refraction

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

Syntax and parameters of this function are subject to change.

Author:

3.5.3.2 Function mie_calc_sizedist

`int` **mie_calc_sizedist** (`mie_inp_struct input, mie_out_struct *output, int program, int medium, mie_complex crefin, float wavelength, float temperature, double *x_size, double *y_size, int n_size, double *beta, double *omega, double *g, mie_complex *ref`) Function

Description:

Mie calculations, using the MIEV0 and BHMIE codes by Warren Wiscombe (ftp://climate.gsfc.nasa.gov/pub/wiscombe) and Bohren and Huffman (ftp://astro.princeton.edu/draine/scat/bhmie/).

Parameters:

`mie_inp_struct input`
: mie input structure (see src_c/miecalc.h)

`mie_out_struct *output`
: mie output structure (see src_c/miecalc.h)

`int program`
: MIEV0 or BHMIE

`int medium`
: WATER, ICE, or USER; if USER, the refractive index is read from crefin

`mie_complex crefin`
: Complex refractive index (both numbers positive)

`float wavelength`
: wavelength [micron]

`float temperature`
: temperature

`double *x_size`
: size distribution, radius [um]

`double *y_size`
: size distribution, n(r)

`int n_size`
: size distribution, number of radii

`double *beta`
: extinction coefficient [km-1] per unit liquid water content (returned)

`double *omega`
: Single scattering albedo (returned)

`double *g` Asymmetry parameter (returned)

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

Syntax and parameters of this function are subject to change.

Author:

3.5.3.3 Function phase_function

`int` phase_function (`float *moment, int L`) Function

Description:

Calculate the phase function from its moments and output to a file "phase.dat". The phase function $p(\mu)$ is

$$p(\mu) = \sum_{m=0}^{\infty} (2m+1) \cdot k_m \cdot P_m(\mu)$$

where k_m is the m'th moment and $P_m(\mu)$ is the m'th Legendre polynomial.

Parameters:

`float *moment`
: moments of the phase function, f[0, ..., L-1]

`int L`
: number of moments

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

Syntax and parameters of this function are subject to change.

Author:

3.5.3.4 Function cumulative_probability

`int` cumulative_probability (`float *moment, int L`) Function

Description:

Calculate the cumulative probability distribution from the moments of the phase function and output to file "cumulative.dat". The algorithm is described by B.R. Barkstrom, "An efficient algorithm for choosing scattering directions in Monte Carlo work with arbitrary phase functions", J.Q.S.R.T., 53, 23-38, 1995. The phase function $p(\mu)$ is

$$p(\mu) = \sum_{m=0}^{\infty} (2m+1) \cdot k_m \cdot P_m(\mu)$$

where k_m is the m'th moment and $P_m(\mu)$ is the m'th Legendre polynomial.

Parameters:

`float *moment`
: moments of the phase function, f[0, ..., L-1]

`int L`
: number of moments

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
Syntax and parameters of this function are subject to change.

Author:

3.5.3.5 Function read_mie_table

`int` **read_mie_table** (`char *filename, double *r0, double *dr, int *nreff, double *wavelen, double *nre, double *nim, double **extinc, double **albedo, int **nleg, float ***legen, int quiet`) Function

Description:
Read Frank Evans' Mie table, similar to his subroutine READ_MIE_TABLE provided in plotmietab.f; the Mie table can be either a single-column file with the moments of the phase function, or a file complying with Frank Evans' 'plotmietab' routine, containing moments for different droplet sizes. IMPORTANT: First, read_mie_table() looks for a file 'filename'.cdf which, when available, is interpreted as the netCDF version of the file. For the netCDF format of the Mie table file, have a look at tools/cloudprp2cdf.sh which converts the ASCII to the netCDF version. ATTENTION: Frank Evans' stores Legendre coefficients f(l), not moments p(l): f = p * (2*l + 1)

Parameters:

`char *filename`
filename where data is stored

`double *r0`
smallest effective radius found in filename

`double *dr`
effective radius step

`int *nreff`
number of effective radii

`double *wavelen`
wavelength [nm]

`double *nre`
real part of the refractive index

`double *nim`
imaginary part of the refractive index

`double **extinc`
array[0 ... nreff-1] of extinction coefficients

`double **albedo`
array[0 ... nreff-1] of single scattering albedos

`int **nleg`
array[0 ... nreff-1] storing the number of Legendre coeffc.

`float ***legen`
array[0 ... nreff-1][0 ... nleg] of Legendre coefficients

`int quiet` 'Shut up!' flag

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.5.3.6 Function read_mie_table_lambda

`int read_mie_table_lambda` (`char *filename, double *wavelen, int quiet`) Function

Description:
Similar to read_mie_table, but reads only wavelength

Parameters:

`char *filename`
filename where data is stored

`double *wavelen`
wavelength [nm]

`int quiet` 'Shut up!' flag

Return value:
0 if o.k., <0 if error

Example:

Files: none

Known bugs:
none

Author:

3.6 Miscellaneous

3.6.1 Usage of the Miscellaneous library

In order to use the functions provided by the misc library, #include <misc.h> in your source code and link with libRadtran_c.a.

3.6.2 Miscellaneous functions

3.6.2.1 Function air_refraction

`double` **air_refraction** (`double lambda`) Function

Description:

Calculate index of refraction of air for 'standard air' according to the 1997/98 'CRC handbook of Chemistry and Physics'. 'Standard air' refers to a temperature of 15 deg C and a pressure of 1013.25 mbar. The index of refraction n is defined as a function of vacuum wavelength, but due to the slow variation of n, the errors are negligibly small when using the air wavelength as input for air_refraction(). The formula is valid between 200 and 2000 nm.

Parameters:

`double lambda`
Wavelength in nm.

Return value:
Index of refraction (double)

Example:

Files: none

Known bugs:
none

Author:

3.6.2.2 Function vac2air

`int` **vac2air** (`double *lambda, double *irradiance, int rows, char reverse, char linear, double **irradiance_shifted`) Function

Description:

Shift a spectrum from vacuum to air or vice versa. The shifted data are regridded to the original wavelength grid. The index of refraction is calculated with air_refraction().

Parameters:

`double *lambda`
Original wavelength grid, i = 0..rows

`double *irradince`
: Original irradiance data, i = 0..rows

`int rows`
: Number of data pairs.

`char *reverse`
: If 0, convert from vacuum to air, else vice versa.

`char *linear`
: If 0, use spline interpolation, else linear.

`double **irradiance\_shifted`
: Shifted irradiance, i = 0..rows, referring to the original wavelength grid; memory is allocated automatically.

Return value:
: 0 if o.k., <0 if error

Example:

Files: none

Known bugs:
: none

Author:

3.6.2.3 Function snowalbedo

`void` **snowalbedo(double** `omega, double tau, double gg, double surface_albedo, double umu0, double* albedo_diffuse, double* albedo_direct`) Function

Description:
: Calculate the diffuse and direct albedo as formulated by Wiscombe and Warren, Journal of the Atmospheric Sciences, vol, 37, 2712-2733, 1980. Equation numbers below refer to equations in their paper.

Parameters:

`double omega`
: snow single scattering albedo

`double tau`
: snow optical depth

`double gg` snow assymmetry factor

`double surface_albedo`
: albedo of underlying surface

`double umu0`
: cosine of solar zenith angle

`double albedo_diffuse`
: diffuse snow albedo, Eq. 6

`double albedo_direct`
direct snow albedo, Eq. 3

Return value:
None

Example:

Files: none

Known bugs:
none

Author: Arve Kylling, arve.kylling@nilu.no

4 Fortran library functions

4.1 Fortran functions in libRadtran

4.1.1 Function wcloud

`SUBROUTINE` **wcloud(** `wavlen, newsiz, nlyr, path, nstring, wccon, wceffr, wc_dtau, wc_gg, wc_ssa, zd, wclyr`**)** Function

Description:

`wcloud` calculates the asymmetry factor, the single scattering albedo and the extinction optical depth of water clouds for a single wavelength. The parameterization due to Hu and Stamnes (1993) is used.

Parameters:

`REAL lambda`
: The wavelength in nanometers. (input)

`LOGICAL newsiz`
: Set `newsiz=.TRUE.` whenever new water cloud liquid water content and/or effective radius are given. If `wccon` and `wceffr` are the same as in the previous call set `newsiz = .FALSE.` to save computer time. Must be equal to `.TRUE.` on the first call. (input)

`INTEGER nlyr`
: Number of atmospheric layers. (input)

`CHARACTER*(*) path`
: The filepath to water cloud parameterization files. (input)

`REAL(*) wccon`
: The liquid water content of each layer in grams per cubic meter. nlyr+1 elements.(input)

`REAL(*) wceffr`
: The water droplet effective radius in microns. nlyr+1 elements. (input)

`REAL(*) wc_dtau`
: The water cloud optical depth of each layer. nlyr elements. (output)

`REAL(*) wc_gg`
: The water cloud asymmetry factor of each layer. nlyr elements. (output)

`REAL(*) wc_ssa`
: The water cloud single scattering of each layer. nlyr elements. (output)

`REAL(*) zd`
: The altitude of each level in km. `zd(nlyr)` is the bottom of the atmosphere. nlyr+1 elements. (input)

`INTEGER wclyr`
: If wclyr is .EQ. 0, the cloud properties are defined per level, otherwise per layer. (input)

Return value:

Example:

See '`uvspec.c`'.

Files:

'`DISORT.MXD`' with parameter `mxcly` must be present in the same directory as '`wcloud.f`'.

Known bugs:

none

Author: Arve Kylling

References

- Anderson, G.P., S.A. Clough, F.X. Kneizys, J.H. Chetwynd and E.P. Shettle 1986, 'AFGL Atmospheric Constituent Profiles (0-120 km)', AFGL-TR-86-0110, AFGL (OPI), Hanscom AFB, MA 01736.
- Cox, C. and W. Munk, 'Measurement of the roughness of the sea surface from photographs of the sun's glitter', 1954, Journal of the Optical Society of America, 44, 838-850.
- Cox, C. and W. Munk, 'Statistics of the sea surface derived from sun glitter', 1954, Journal of Marine Research, 13, 198-227.
- Dahlback, A. and K. Stamnes, 1991, 'A new spherical model for computing the radiation field available for photolysis and heating at twilight', Planet. Space Sci., 39, 671-683.
- Dahlback, A., 1996, 'Measurement of biologically effective UV doses, total ozone abundances, and cloud effects with multichannel, moderate bandwidth filter instruments', Applied Optics, 6514-6521.
- Daumont, D., J. Brion, J. Charbonnier, and J. Malicet, 1992, 'Ozone UV Spectroscopy I: Absorption Cross-Sections at Room Temperature', J. Atm. Chem., 15, 145-155.
- Edwards, D.P., 1992, 'GENLN2: A general line-by-line atmospheric transmittance and radiance model: Version 3.0 description and users guide', National Center for Atmospheric Research (NCAR), NCAR/TN-367+STR, Boulder, Colorado.
- Evans, K.F. and G.L. Stephens, 1991, 'A new polarized atmospheric radiative transfer model', J. Quant. Spectrosc. Radiat. Transfer, 46, 413-423.
- Fu, Q. and K.N. Liou, 1992, 'On the correlated k-distribution method for radiative transfer in nonhomogeneous atmospheres', J. Atm. Sci., 49, 2139-2156.
- Fu, Q. and K.N. Liou, 1993, Parameterization of the radiative properties of cirrus clouds, J. Atm. Sci., 50, 2008-2025.
- Fu, Q., 1996, 'An accurate parameterization of the solar radiative properties of cirrus clouds in climate models', Journal of Climate, 9, 2058-2082.
- Fu, Q., P. Yang, and W.B. Sun, 1998, 'An accurate parameterization of the infrared radiative properties of cirrus clouds in climate models', Journal of Climate, 11, 2223-2237.
- Garcia, R., and C. Siewert 1985, 'Benchmark results in radiative transfer', Transp. Theory and Stat. Physics, 14, 437-484.
- Hu, Y.X. and K. Stamnes 1993, 'An accurate parameterization of the radiative properties of water clouds suitable for use in climate models', J. Climate, 6, 728-742.
- Kato, S., T.P. Ackerman, J.H. Mather, and E.E. Clothiaux, 1999, 'The k-distribution method and correlated-k approximation for a shortwave radiative transfer model' J. Quant. Spectrosc. Radiat. Transfer, 62, 109-121.
- Key, J., P. Yang, B. Baum, and S. Nasiri, 2002, 'Parameterization of shortwave ice cloud optical properties for various particle habits', J. Geophys. Res., 107, D13, 10.1029/2001JD000742.
- Kratz, D.P., 1995, 'The correlated k-distribution technique as applied to the AVHRR channels', J. Quant. Spectrosc. Radiat. Transfer, 53, 501-517.

- Kylling, A., K. Stamnes, and S.-C. Tsay, 1995, 'A reliable and efficient two-stream algorithm for radiative transfer; Documentation of accuracy in realistic layered media', accepted for publication in J. of Atmospheric Chemistry.
- Malicet, J., D. Daumont, J. Charbonnier, C. Parisse, A. Chakir, and J. Brion, 1995, 'Ozone UV Spectroscopy II: Absorption Cross-Sections and Temperature Dependence', J. Atm. Chem., 21, 263-273.
- Mayer, B., G. Seckmeyer and A. Kylling, 1997, 'Systematic long-term comparison of spectral UV measurements and UVSPEC modeling results', J. Geophys. Res., 102, 8755–8767.
- Mayer, B., A. Kylling, S. Madronich, and G. Seckmeyer, 1998, Enhanced absorption of UV radiation due to multiple scattering in clouds: experimental evidence and theoretical explanation, J. Geophys. Res., 103 (D23), 31241-31254.
- Molina, L. T. and M. J. Molina, 1986, 'Absolute Absorption Cross Sections of Ozone in the 185- to 350-nm Wavelength Range', J. Geophys. Res., 91, 14,501-14,508.
- Nakajima, T. and M. Tanaka, 'Effect of wind-generated waves on the transfer of solar radiation in the atmosphere-ocean system', 1983, J. Quant. Spectrosc. Radiat. Transfer, 29, 521-537.
- Nakajima, T., and M. Tanaka, 1988, 'Algorithms for radiative intensity calculations in moderately thick atmospheres using a truncation approximation', J. Quant. Spectrosc. Radiat. Transfer, 40, 51-69.
- Nicolet, M. ,1984, 'On the molecular scattering in the terrestrial atmosphere: an empirical formula for its calculation in the homosphere', Planet. Space Sci., 32, 1467-1468.
- Pierluissi, J.H., and G.-S. Peng, 1985, 'New molecular transmission band models for LOWTRAN', Optical Engineering, 24, 541-547.
- Rahman, H. and B. Pinty, and M.M. Verstraete, 1993, 'Coupled Surface-Atmosphere Reflectance (CSAR) Model. 2. Semiempirical Surface Model Usable With NOAA Advanced Very High Resolution Radiometer Data', J. Geophys. Res. 98, 20791-20801.
- Ricchiazzi, P., S. Yang, C. Gautier, and D. Sowle, 1998, 'SBDART: A research and teaching software tool for plane-parallel radiative transfer in the Earth's atmosphere', Bull. Am. Met. Soc. 79, 2101-2114.
- Shettle E. P. (1989), 'Models of aerosols, clouds and precipitation for atmospheric propagation studies', in AGARD Conference Proceedings No. 454, Atmospheric propagation in the uv, visible, ir and mm-region and related system aspects.
- Stamnes, K., 1982, 'Reflection and Transmission by a Vertically Inhomogeneous Planetary Atmosphere', Planet. Space Sci. 30, 727-732.
- Stamnes, K., S.-C. Tsay, W. Wiscombe and K. Jayaweera, 1988, 'Numerically stable algorithm for discrete-ordinate-method radiative transfer in multiple scattering and emitting layered media', Applied Optics, 27, 2502.
- Stamnes, K., J. Slusser, and M. Bowen, 1991, 'Derivation of Total Ozone Abundance and Cloud Effects from Spectral Irradiance Measurements', Applied Optics, 30, 4418-4426.
- Wiscombe, W., 1977, 'The Delta-M Method: Rapid Yet Accurate Radiative Flux Calculations, J. Atmos. Sci. 34, 1408-1422.

- Wiscombe, W.J. and J.W. Evans, 1977, 'Exponential-sum fitting of radiative transmission functions', Journal of Computational Physics, 24, 416-444.
- Wiscombe, W.J. and S.G. Warren, 1980, 'A Model for the Spectral Albedo of Snow. I: Pure Snow', J. Atm. Sci., 37, 2712-2733.
- Yang, P. and K.N. Liou and K. Wyser, and D. Mitchell, 2000, 'Parameterization of the scattering and absorption properties of individual ice crystals', J. Geophys. Res., 105(D4), 4699-4718.
- Yang, P., B.-C. Gao, B.A.Baum ,X.H. Yong, W.J. Wiscombe, S.-C. Tsay, D.M. Winker, and S.L.Nasiri, 2001, 'Radiative properties of cirrus clouds in the infrared (8-13 micron) spectral region', J. Quant. Spectrosc. Radiat. Transfer, 70, 473-504.

Table of Contents