

# libRadtran

---

library for radiative transfer calculations  
Edition 1.0 for libRadtran version 1.4  
December 2008

Bernhard Mayer, Arve Kylling, Ulrich Hamann, Claudia Emde

---

Copyright © 1997-2008 Bernhard Mayer, Arve Kylling, Ulrich Hamann, Claudia Emde.

This edition of the libRadtran documentation is consistent with version 1.4 of libRadtran.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the same conditions as for modified versions.

## 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 Bernhard Mayer ([bernhard.mayer@dlr.de](mailto:bernhard.mayer@dlr.de)), Arve Kylling ([arve.kylling@helse-sunnmore.no](mailto:arve.kylling@helse-sunnmore.no)), Ulrich Hamann ([ulrich.hamann@dlr.de](mailto:ulrich.hamann@dlr.de)), and Claudia Emde ([claudia.emde@dlr.de](mailto:claudia.emde@dlr.de)), the following people have contributed to libRadtran or helped out in various other ways (the list is almost certainly incomplete - please let us know if we forgot somebody):

- 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](mailto:kato@aerosol.larc.nasa.gov)) provided the correlated-k tables described in Kato et al. (1999).
- Tom Charlock ([t.p.charlock@larc.nasa.gov](mailto:t.p.charlock@larc.nasa.gov)), Quiang Fu ([qfu@atm.dal.ca](mailto:qfu@atm.dal.ca)), and Fred Rose ([f.g.rose@larc.nasa.gov](mailto:f.g.rose@larc.nasa.gov)) provided the most recent version of the Fu and Liou code.
- David Kratz ([kratz@aquila.larc.nasa.gov](mailto: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](mailto: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](mailto: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](mailto: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](mailto:luca.bugliaro@dlr.de)) wrote the analytical TZS solver (thermal, zero scattering).
- Sina Lohmann ([sina.lohmann@dlr.de](mailto:sina.lohmann@dlr.de)) reduced the “overhead time” for reading the Kato et al. tables dramatically which resulted in a speedup of a factor of 2 in a twostr solar irradiance calculation.
- Detailed ice cloud properties were provided by Bryan Baum ([bryan.baum@ssec.wisc.edu](mailto:bryan.baum@ssec.wisc.edu)).
- Yongxiang Hu ([yongxiang.hu-1@nasa.gov](mailto:yongxiang.hu-1@nasa.gov)) provided the delta-fit program used to calculate the Legendre coefficients for `ic_properties` `baum`.
- UCAR/Unidata for providing the *netCDF* library.
- 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 1.4.

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, **uvspec** 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 absorption, 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 67.

### 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 where 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 68.

### 1.4 ... and much more

Apart from the above described programs, the ‘bin’ 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 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. 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. Be aware that also option in another `included` input file will overwrite options specified before.

##### 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 explicitly 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`.

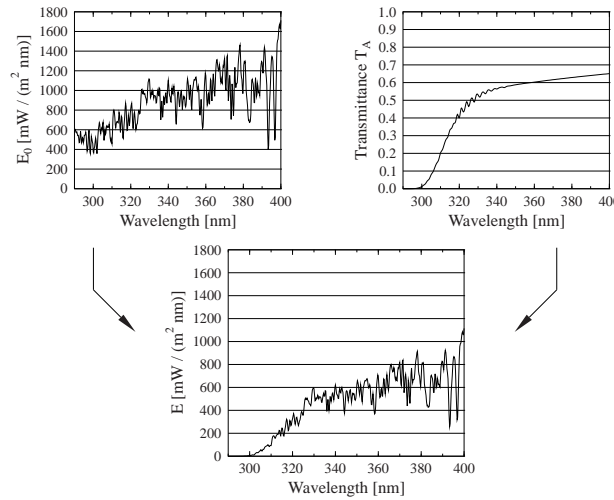
A wavelength dependent **surface albedo** may be specified using `albedo_file` instead of `albedo`. Non-Lambertian surface reflectance (BRDF) for vegetation and water may also be defined (please note that these require the use of `rte_solver disort2`). The BRDF of Vegetation is specified using `rpv_rho0`, `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_pchl`, 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 explicitly 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`', there will only be output if the wavelength 327.14 is explicitly 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:

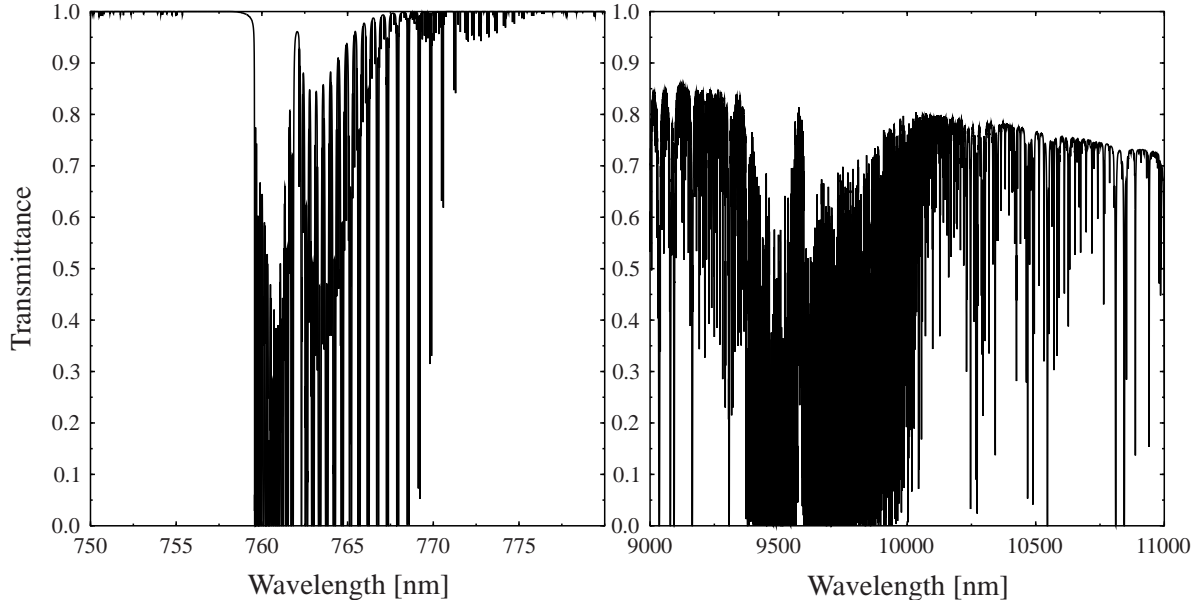
1. **Spectrally resolved calculation** in the UV and visible spectral ranges;
2. **Line-by-line calculation** with user-defined molecular absorption data;
3. **The correlated-k method**.
4. **Pseudo-spectral calculation** with exponential-sum-fit, from LOWTRAN; code adopted from SBDART (Ricchiazzi et 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 O<sub>3</sub> and NO<sub>2</sub>.

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 O<sub>2</sub>A-absorption band around 760nm and a region within the infrared window around 10 micrometer.



All spectral lines in the left figure are due to absorption by oxygen, while the ones in the right figure are due to ozone, water vapour, and CO<sub>2</sub>. 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:

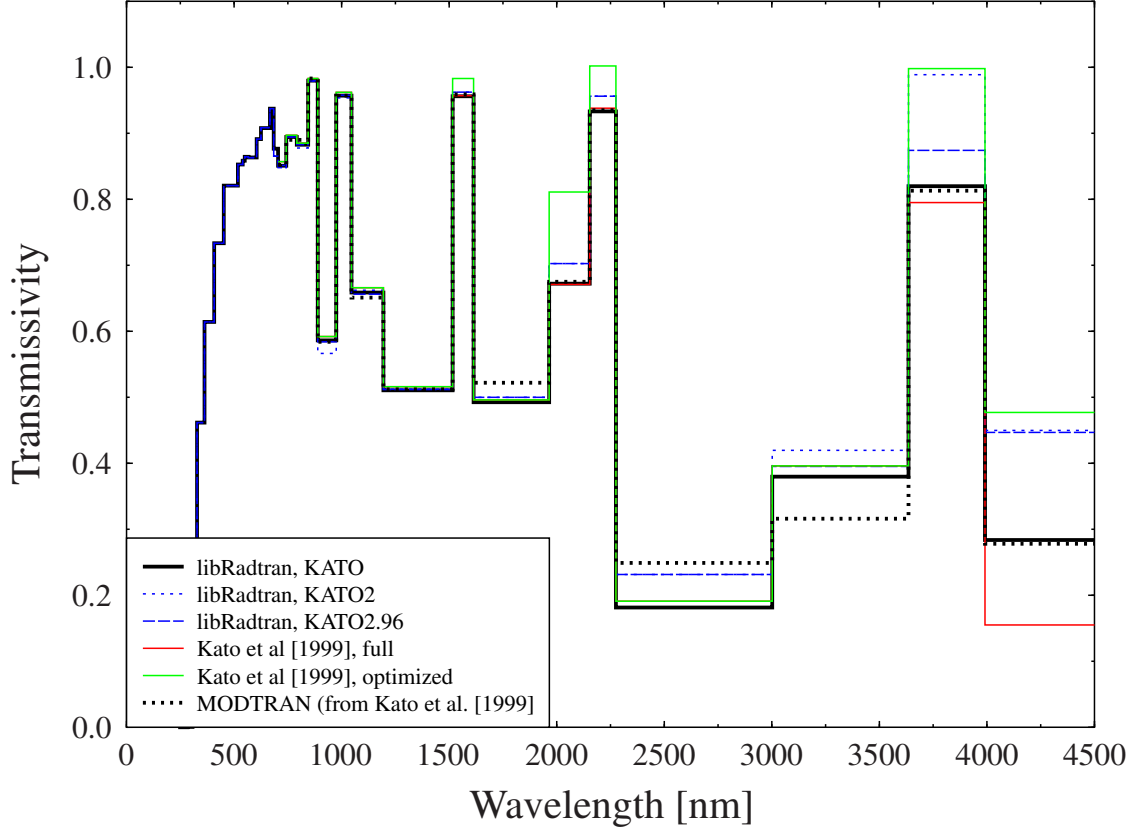
**Kato** The original tables provide by Seiji Kato which should correspond to the full version described in Kato et al. (1999); 575 subbands total, that is, 575 calls to the `rte_solver`

**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 micrometer which is probably due to the treatment of band overlap. For this reasons, the results for the individual 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 micrometer 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.  $W / (m^2 nm)$ , but integrated over the spectral bands, e.g.  $W / m^2$ .

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<sup>-1</sup> 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<sup>-1</sup> 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 explicitly

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

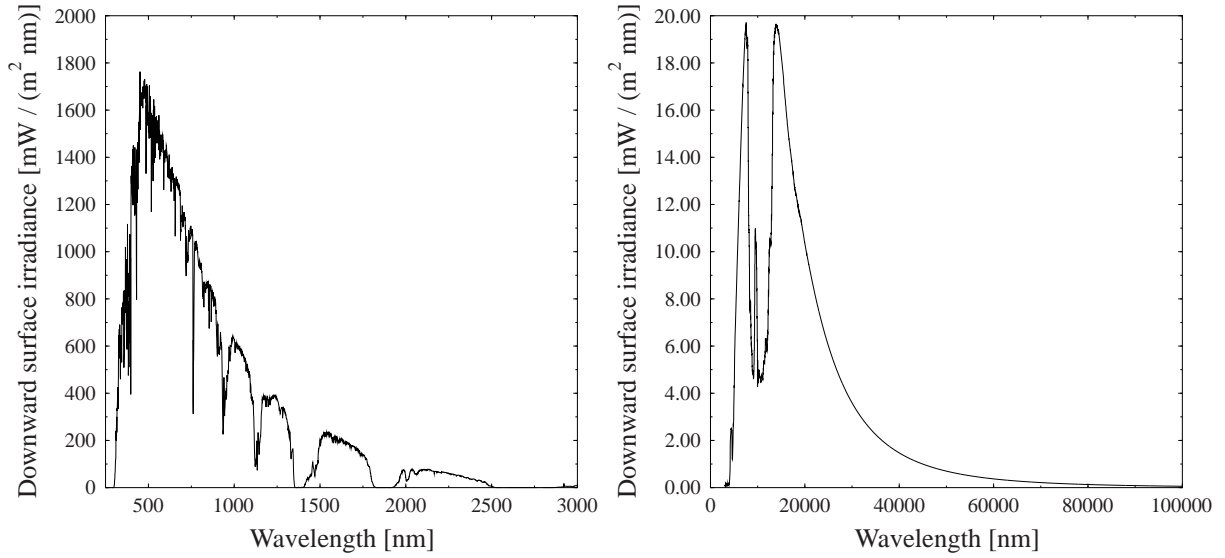
source thermal

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

correlated_k SBDART # select SBDART molecular absorption

output per_nm

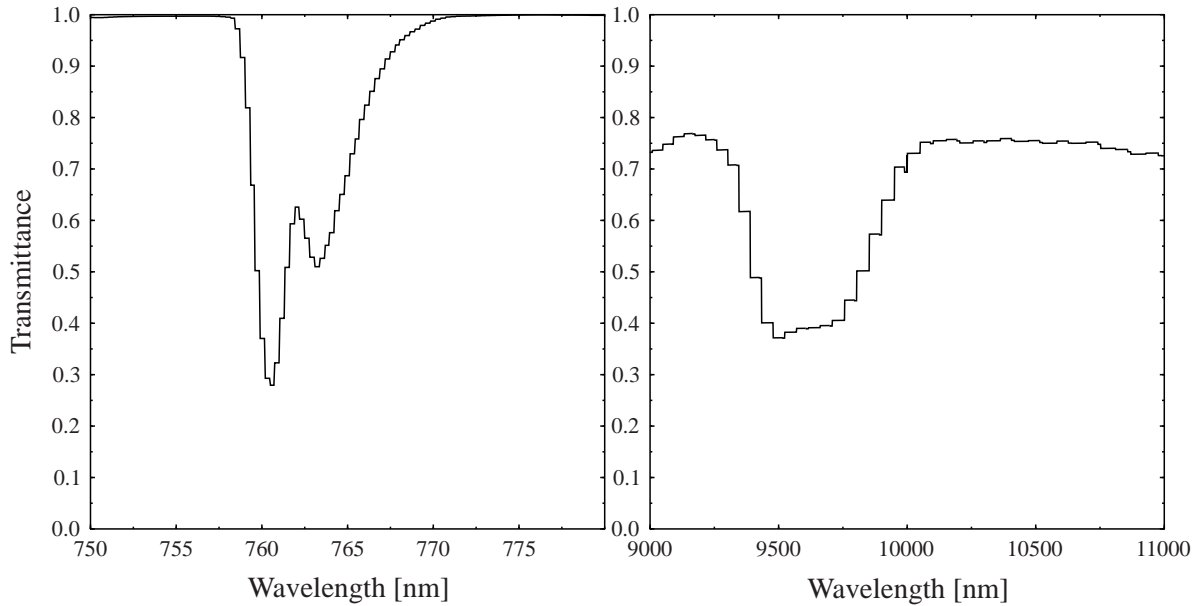
quiet
```



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 micrometer in the thermal. Please note the following points:

- Thermal radiation is per default output in W/(m<sup>2</sup> cm<sup>-1</sup>), if the bandwidth is equal to 1 cm<sup>-1</sup> (default for `correlated_k` LOWTRAN calculations). Otherwise the output is the integrated flux over the wavenumber interval specified by `thermal_bandwith`, `thermal_bands_file`, or by the `correlated_k` option (Kato, Kato2, Kato2.96, Fu, AVHRR\_KRATZ, or Generic). To convert e.g. to W/(m<sup>2</sup> nm) use `output_per_nm` or multiply with  $k/\lambda$  where  $k$  is the wavenumber [cm<sup>-1</sup>] 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<sup>-1</sup> which translates to 0.3nm at 750nm and to 50nm at 10 micrometer. 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 explicitly 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 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 layer or 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/WCSIMPLE.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 <sup>3</sup> )	(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/m<sup>3</sup>], and the effective droplet radius [micrometer]. Per default (since version 1.4), these quantities are interpreted as layer quantities, and in the above example, the cloud would extend from 2 to 5km, with e.g. a LWC of 0.2 g/m<sup>3</sup> for the layer between 4 and 5km. Before version 1.4 the `wc_file` was interpreted as level quantities (unless `wc_layer` was specified). That is, the value 0.2 g/m<sup>3</sup> referred to altitude 4.0km, as e.g. in a radiosonde profile. The properties of each layer were calculated as average over the adjacent levels. E.g. the single scattering properties for the model layer between 3 and 4km were obtained by averaging over the two levels 3km and 4km. To allow definition of sharp cloud boundaries, clouds were only formed if both liquid water contents above and below the respective layer were larger than 0. Hence, in the above example, the layers between 2 and 3 as well as between 3 and 4km were cloudy while those between 1 and 2km and between 4 and 5km were not. To switch to the old behaviour, use `wc_level`.

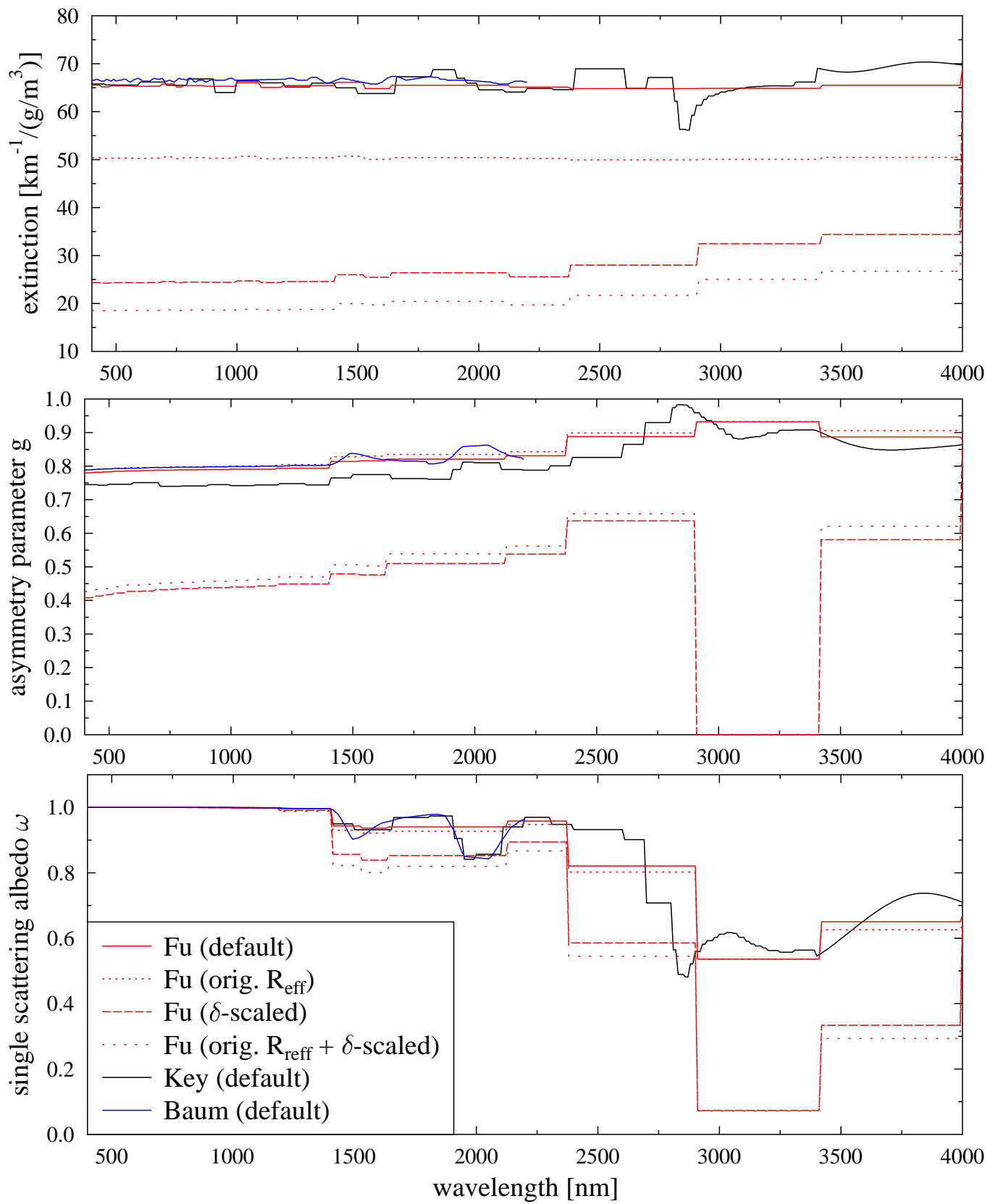
To make sure that the clouds really look as you want them to look, it is recommended to use the `verbose` option. This option shows not only where the cloud is actually placed, it rather tells the user exactly how LWC and effective radius are translated into optical properties, depending on the choice 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 define `wc_properties_interpolate` to automatically interpolate the Mie properties to the internal wavelength grid. Although this is an extremely useful option, please use it carefully because it might consume enormous 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 explicitly 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 Baum et al. (2005a,b, 2007) use the same definition whereas Fu (1996) actually uses another definition (see explanation of `ic_properties`). Finally, the sharp forward peak which is typical for ice particles can also be 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 with an effective radius of 25 micrometers as a function of wavelength. If treated consistently, all parameterizations Key et al. (2002), Fu (1996) and Baum (2005a,b) provide nearly identical results (solid lines, default settings in `uvspec`). If the definition of effective radius by Fu (1996) and delta-scaling is applied the optical properties look different. The effect of delta scaling on a radiative transfer calculation is that the direct irradiance is increased and the diffuse irradiance is decreased, whereas the global irradiance remains unchanged. The definition of the effective radius has a smaller effect but it modifies also the global irradiance. Note that the parameterization by Baum et al. (2005a,b, 2007) is plotted only up to 2200 nm. The reason is that it does not cover the full spectral region, it is available for two spectral regions (from 0.4–2.2 micrometers and from about 3–100 micrometers).



### 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 SPSPDISORT

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.

<code>cmu</code>	Computational polar angles from <code>polradtran</code> .
<code>down_flux</code> , <code>up_flux</code>	The total (direct+diffuse) downward ( <code>down_flux</code> ) and upward ( <code>up_flux</code> ) irradiances. Same units as extraterrestrial irradiance.
<code>lambda</code>	Wavelength (nm)
<code>edir</code>	Direct beam irradiance (same unit as extraterrestrial irradiance, e.g mW/(m <sup>2</sup> nm) if using the 'atlas3' spectrum in the 'data/solar_flux' directory.)
<code>edn</code>	Diffuse down irradiance, i.e. total minus direct beam (same unit as <code>edir</code> ).
<code>eup</code>	Diffuse up irradiance (same unit as <code>edir</code> ).
<code>uavg</code>	The mean intensity. Proportional to the actinic flux: To obtain the actinic flux, multiply the mean intensity by 4 pi (same unit as <code>edir</code> ).

<code>uavgdir</code>	Direct beam contribution to the mean intensity. (same unit as <code>edir</code> ).
<code>uavgdn</code>	Diffuse downward radiation contribution to the mean intensity. (same unit as <code>edir</code> ).
<code>uavgup</code>	Diffuse upward radiation contribution to the mean intensity. (same unit as <code>edir</code> ).
<code>u0u</code>	The azimuthally averaged intensity at <code>numu</code> user specified angles <code>umu</code> . (units of e.g. $\text{mW}/(\text{m}^2 \text{ nm sr})$ if using the ‘atlas3’ spectrum in the ‘data/solar_flux’ directory.)
<code>uu</code>	The radiance (intensity) at <code>umu</code> and <code>phi</code> user specified angles (unit e.g. $\text{mW}/(\text{m}^2 \text{ nm sr})$ if using the ‘atlas3’ spectrum in the ‘data/solar_flux’ directory.)
<code>uu_down, uu_up</code>	The downwelling and upwelling radiances (intensity) at <code>cmu</code> and <code>phi</code> angles (unit e.g. $\text{mW}/(\text{m}^2 \text{ nm sr})$ if using the ‘atlas3’ spectrum in the ‘data/solar_flux’ directory.)

The total downward irradiance is given by

$$\text{irr\_down} = \text{edir} + \text{edn}$$

The total mean intensity is given by

$$\text{uavg} = \text{uavgdir} + \text{uavgdn} + \text{uavgup}$$

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. Be aware that also option in another `included` input file will overwrite options specified before.

The various input parameters are described in detail below.

#### `absorption`

Switch off absorption by individual minor trace gases which are currently only included when `correlated_k lowtran` is chosen. The syntax is

`absorption species on/off`

where `species` may be one of `O4`, `N2`, `CO`, `SO2`, `NH3`, `NO`, `HNO3`. By default all are switched on.

**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-defined **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 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<sup>-1</sup>.

**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. The asymmetry parameter defined with this option is constant with wavelength. If you require spectral dependence please use **aerosol\_files**. 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_no_scattering`

Switch off scattering by aerosols.

#### `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. Please note that only the single-scattering albedo, the scattering phase function, and the wavelength-dependence of the extinction coefficient are affected by the Mie calculation while the absolute value of the extinction coefficient is taken from other sources; generally, the extinction coefficient at the first *internal* wavelength is taken from whatever is available (either default Shettle (1989) or user-defined); the extinction at all other wavelengths is scaled according to the Mie calculation. For this reason, the absolute numbers are not relevant - only the shape of the size distribution matters. In detail: 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 at least 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`. *It is recommended to avoid this option and rather to calculate the aerosol optical properties externally e.g. with mie and to pass them to uvspec with aerosol\_files.*

**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. Please see the description of **aerosol\_refrac\_file** to learn how the optical properties are set up. *It is recommended to avoid this option and rather to calculate the aerosol optical properties externally e.g. with mie and to pass them to uvspec with aerosol\_files.*

**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-defined **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-defined **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 [micrometer] 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. Please note that only the single-scattering albedo, the scattering phase function, and the wavelength-dependence of the extinction coefficient

are affected by the Mie calculation while the absolute value of the extinction coefficient is taken from other sources; generally, the extinction coefficient at the first *internal* wavelength is taken from whatever is available (either default Shettle (1989) or user-defined); the extinction at all other wavelengths is scaled according to the Mie calculation. For this reason, the absolute numbers are not relevant - only the shape of the size distribution matters. For details see also the description of `aerosol_refrac_file`. *It is recommended to avoid this option and rather to calculate the aerosol optical properties externally e.g. with mie and to pass them to uvspec with aerosol\_files.*

#### `aerosol_species_file`

Specify mass density profiles of a mixture of aerosol types.

```
aerosol_species_file profile [aero_1 aero_2 ... aero_n]
```

where `aero_1` to `aero_n` are the aerosol species to be included. For each of these species, the optical properties are read from the `aerosol_species_library`, e.g. the OPAC data set provided with libRadtran. The profile file needs to include vertical profiles for each of these species. This file can be either in *netCDF*-format (automatically recognized filename extension `.nc` or `.cdf`) or in ASCII format. The format of the ASCII file is:

```
z1      dens(aero_1, z1) dens(aero_2, z1) ... dens(aero_n, z1)
z2      dens(aero_1, z2) dens(aero_2, z2) ...
.        .                .
.        .                .
```

where `z` is the height in km, and `dens` are the aerosol mass densities in g/m<sup>3</sup>. Please make sure to include one column for each of the species `aero_1` to `aero_n` listed after `aerosol_species_file`. For *netCDF* input it is also possible to specify the unit 'kg kg<sup>-1</sup>'; the data are then automatically converted to g/m<sup>3</sup>.

#### `aerosol_species_library`

With this option the *directory* is specified where the optical property files for all aerosols species used in the `aerosol_species_file` are expected: For each species defined in `aerosol_species_file`, *netCDF*-file `species_name.nc`, (e.g. `INSO.nc`), which contains the optical properties of the aerosol species, has to be provided. The *netcdf* format is the one produced by the libRadtran mie tool.

At the libRadtran webpage we provide the OPAC data set (Hess et al. 1998) which can be directly used with uvspec:

```
aerosol_species_library OPAC
```

OPAC contains following aerosol species:

```
INSO  # insoluble
WASO  # water_soluble
SOOT  # soot
SSAM  # sea_salt_accumulation_mode
SSCM  # sea_salt_coarse_mode
MINM  # mineral_nucleation_mode
MIAM  # mineral_accumulation_mode
MICM  # mineral_coarse_mode
MITR  # mineral_transported
```

SUSO # sulfate\_droplets

#### 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. The single scattering albedo defined with this option is constant with wavelength. If you require spectral dependence please use `aerosol_files`. 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 allow wavelength-dependent aerosol optical thickness please use either `aerosol_angstrom` or `aerosol_files`. Comments start with `#`. Empty lines are ignored.

#### aerosol\_visibility

Horizontal visibility in km. Affects the profile according to Shettle (1989) and the optical thickness.

#### aerosol\_vulcan

Aerosol situation above 2 km. Integer.

- 1 Background aerosols.
- 2 Moderate volcanic aerosols.
- 3 High volcanic aerosols.
- 4 Extreme volcanic 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. An arbitrary wavelength grid may be chosen as the albedo will be interpolated linearly to the wavelength grid used for the radiation calculation. Comments start with `#`. Empty lines are ignored. A large collection of spectral albedos is available e.g. at <http://speclib.jpl.nasa.gov/>.

#### albedo\_library

Albedo libraries are a collection of spectral albedos of different surface types. This option must be used either with `surface_type` or `surface_type_map`, in order to select the specific surface type. There are two possibilities for libraries: the built-in IGBP library or a user defined albedo library.

The built-in library of the International Geosphere Biosphere Programme is selected with `albedo_library` IGBP. The IGBP library contains 20 surface types:

```

1 evergreen_needle_forest
2 evergreen_broad_forest
3 deciduous_needle_forest
4 deciduous_broad_forest
5 mixed_forest
6 closed_shrubs
7 open_shrubs
8 woody_savanna
9 savanna
10 grassland
11 wetland
12 cropland
13 urban
14 crop_mosaic
15 antarctic_snow
16 desert
17 ocean_water
18 tundra
19 fresh_snow
20 sea_ice
```

Surface types 1 - 17 are defined by the International Geosphere Biosphere Programme (IGBP); additionally there are tundra, fresh\_snow, and sea\_ice surface types. The spectral albedo of the ground is determined as a function of solar zenith angle, precipitable water, and clouds. The spectral resolution equals the grid of the correlated-k Fu/Liou parameterisation. This library originates from the NASA CERES/SARB Surface Properties Project, see Belward and Loveland (1996).

For creating your own albedo library use `albedo_library path`, where `path` is the path of the directory where the albedo data is stored. The files are expected to have the names `albedo_01.dat`, `albedo_02.dat`, ... If `surface_type` 1 is specified the albedo from `albedo_01.dat` will be used, and so on. Each file is required to 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 linearly to the wavelength grid used for the radiation calculation. Comments start with `#`. Empty lines are ignored. This option is similar to `albedo_file`, except that it offers an easy way to use the option `surface_type_map` in combination with albedo files.)

#### `albedo_map`

*This option is preliminary and still subject to change (no wavelength dependency yet)! A possibility to specify a wavelength independent albedo with the help of a *netCDF* file, which is used in combination with the options `latitude`, `longitude`, and `time`.*

```
albedo_map file [variable_name]
```

where **file** is the location of the *netCDF* file. With the optional argument the name of the albedo variable in the *netCDF* file can be specified (the default name is **AL**). The albedo must be provided as function of **latitude** and **longitude** **AL(lat, lon)**, and may also depend on time **AL(time, lat, lon)**. The latitude, longitude, and time grids must be provided as doubles **double lat(lat)**, **double lon(lon)**, and **double time(time)**. **libRadtran** reads the value at the nearest pixel to the given **latitude** and **longitude**. No spatial interpolation or averaging of the values is done. If a time-dependent albedo is provided, the albedo data nearest to the specified time will be selected (or linear interpolated if **time\_interpolate** is switched on).

**altitude** Set the bottom level in the model atmosphere provided in **atmosphere\_file** to be at the given altitude above sea level (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** above the surface. E.g. to calculate the radiation field 1 km above the surface at a location at 0.73 km above sealevel, one would specify '**altitude 0.73**' and '**zout 1.0**'. If an altitude is specified which is below the lowest level in the **atmosphere\_file**, the atmospheric profiles are extrapolated assuming a constant gradient for temperature and mixing ratios. 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.

**altitude\_map**

Specifies an altitude map which is used in combination with **latitude**, **longitude** in order to select the altitude for the simulation. No interpolation is done between the pixels of the map. The format of the call is:

```
altitude_map file [variable_name]
```

where **file** is the location of the altitude map file. The map is expected to be in *netCDF* format. The file must contain **double lat(lat)**, **double lon(lon)**, and the altitude variable, where **variable\_name** is the name of the surface elevation variable in the *netCDF* file. The default name is **Z**. The altitude variable must be **altitude(lat, lon)**. For format discription see also the example map included in **libRadtran**, **data/altitude/ELEVATION\_GTOPO\_10min.cdf**. To use this map in **uvspec**, you may also use **altitude\_map GTOPO**. This map has a resolution of 10 arc minutes and the unit of the altitude is meter. Please

note that this resolution might not be adequate for your application. If an altitude in the map is below the lowest level of the `atmosphere_file`, the atmospheric profiles are extrapolated assuming a constant gradient for temperature and mixing ratios.

#### `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 trace 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 hPa
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: For air as an ideal gas the density can be calculated from pressure and temperature,  $\text{dens} = p / kT$ . `uvspec` will check if this relation is fulfilled and will stop 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
```

If no `atmosphere_file` is defined, `libRadtran` will automatically select one. If the information `time`, `latitude` and `longitude` are provided in the input file

libRadtran will choose from the first 5 files, otherwise it takes the U.S. Standard atmosphere.

#### brightness

Convert radiances / irradiances to equivalent 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 micrometer 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 micrometer 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 micrometer 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/KATO2.96/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 check which is initiated with `make check`.

#### `cox_and_munk_pcl`

Pigment concentration for Cox and Munk ocean 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. The default value is 0.01 mg/m-3. To switch on Cox and Munk BRDF, specify any of the `cox_and_munk` options and define at least `cox_and_munk_u10`.

#### `cox_and_munk_pcl_map`

A possibility to specify pigment concentration (in mg/m3) for the Cox and Munk ocean BRDF with the help of an *netCDF* file, which is used in combination with options `latitude`, `longitude`, and `time`.

`cox_and_munk_pcl_map file [variable_name]`

where `file` is the location of the *netCDF* file. `libRadtran` reads the value at the nearest pixel to the given `latitude` and `longitude`. No spatial interpolation or averaging of the values is done.

The default name of the pigment concentration variable is `chlorophyll`, but can be changed with the optional argument `variable_name`. The pigment concentration must be provided as function of `latitude` and `longitude`, `chlorophyll(lat, lon)`, or additionally may also depend on time `chlorophyll(time, lat, lon)`. If a time-dependent pigment concentration is specified, the pigment concentration will be interpolated according to the option `time_interpolate`. All grids must be provided in the file as `double lat(lat)`, `double lon(lon)`, and `double time(time)`.

#### `cox_and_munk_sal`

Salinity for Cox and Munk ocean BRDF (in "per mille", 0.1‰; this unit is equivalent to the other common units for salinity, ppt - parts per thousand, psu - practical salinity unit); 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. The default value is 34.3. To switch on Cox and Munk BRDF, specify any of the `cox_and_munk` options and define at least `cox_and_munk_u10`.

#### `cox_and_munk_sal_map`

Specify ocean salinity (in ppt) for the Cox and Munk ocean BRDF with the help of an *netCDF* file, which is used in combination with the options `latitude`, `longitude`, and `time`.

`cox_and_munk_sal_map file [variable_name]`

where `file` is the location of the *netCDF* file. `libRadtran` reads the value at the nearest pixel to the given `latitude` and `longitude`. No spatial interpolation or averaging of the values is done.

The expected name of the pigment concentration variable is per default `salinity`, but can be changed with the optional argument `variable_name`.

The pigment concentration must be provided as function of `latitude` and `longitude`, `salinity(lat, lon)`, or additionally may also depend on time `salinity(time, lat, lon)`. If a time-dependent salinity is specified, the salinity will be interpolated according to the option `time_interpolate`. All grids must be provided as `double lat(lat)`, `double lon(lon)`, and `double time(time)`.

#### `cox_and_munk_u10`

Wind speed for Cox and Munk ocean BRDF (in m/s); at present only available with `rte_solver DISORT2`. The wind speed is the most important parameter affecting ocean BRDF. 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. To switch on Cox and Munk BRDF, specify any of the `cox_and_munk` options and define at least `cox_and_munk_u10`.

#### `cox_and_munk_u10_map`

Specify wind speed (in m/s) for the Cox and Munk ocean BRDF with the help of an *netCDF* file, which is used in combination with the options `latitude`, `longitude`, and `time`.

`cox_and_munk_u10_map file`

where `file` is the location of the *netCDF* file. `libRadtran` reads the value at the nearest pixel to the given `latitude` and `longitude`. No spatial interpolation or averaging of the values is done.

The file must contain the elements of the wind vector `U10` and `V10`. These must be specified as functions of `latitude` and `longitude` `U10(lat, lon)`, `V10(lat, lon)`, or additionally may also depend on time `U10(time, lat, lon)`, `V10(time, lat, lon)`. If the variable time is present in the file, the wind speed will be interpolated according to the option `time_interpolate`. All grids must be provided as `double lat(lat)`, `double lon(lon)`, and `double time(time)`.

`crs_file` May be used to specify cross sections of `O3`, `NO2`, `BRO`, `OCLO`, or `HCHO` to be used instead 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 species for which the cross section applies. The cross section file has two columns:

1. `wavelength (nm)`
2. `cross section (cm2)`

#### `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 distance), that is, no correction is applied to the extraterrestrial irradiance **solar\_file**. Alternatively **time** may be used for that purpose.

**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 species column [unit]
```

where **species** is one of O3, O2, H2O, CO2, NO2, BRO, OCLO, or HCHO, see also **dens\_file**. The second argument is the total column value, and the optional third argument is the unit, in which the column is given. The unit can be DU (Dobson units) or CM\_2 (molecules/cm<sup>2</sup>). The default units are DU for O3, and CM\_2 for all other gases. It is possible to have several **dens\_column** commands in the input file (maximum one per species).

```
dens_column NO2 1.2 DU
```

**dens\_file**

Specify density profiles (or matrix, see below) of various trace gases to be included in the radiative transfer calculation. The entry of the input file looks like this:

```
dens_file gas_species [unit] filename
```

At the moment following **gas\_species** are included: ozone (O3), nitrogen dioxide (NO2), water vapor (H2O), bromine oxide (BRO), chlorine dioxide (OCLO), formaldehyde (HCHO), and carbon dioxide (CO2). The gas species is identified by their abbreviations given in the parenthesis above. Unit is an optional argument to define the unit of the density. The profiles can be given in particles per cm<sup>3</sup> (cm-3), in particles per m<sup>3</sup> (m-3), as volume mixing ratio (vmr), as mass mixing ratio (mmr), or as relative humidity (only for water). The default for unit is cm-3. The model expects a density file with two columns:

1. **z(km)** Altitude above sea level in km.
2. **density of trace gas [in the specified unit]**  
The density of the trace gas

The altitude grid may be different from the **atmosphere file**. All densities inside the range of the **dens\_file** are replaced. For all other altitudes the values from the **atmosphere\_file** are used. If the density is specified as -1 at a level, the value from **atmosphere\_file** is used.

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:

```

0.0      SZA1      SZA2 ...
z(1)     dens(1,1)  ...
z(2)     .         .
.        .         .

```

where  $z(i)$  are the altitude levels above sea level in km, SZA is the solar zenith angle in degrees, and dens is the density [in the specified unit] of the trace gases as 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 can 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`

Provide output in flexstor format. Must 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-defined `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`

Include information on some of the input parameters in the output. May not be combined with `flexstor`. Please note that the information provided is rather incomplete because this option was introduced quite early and was never updated. For a more complete information please use the `verbose` option.

**heating\_rate**

Calculation of heating rates. Output is only provided at altitudes specified by **zout**. To get heating rate profiles a number of altitudes must thus be specified. Heating rates is the change of temperature with time in units of K/day. For spectral calculations the default output is a matrix:

```

0.0      zout1      zout2 ...
lambda1  heating_rates ...
lambda2      .
.           .
.           .

```

For integrated calculations (**output sum** or **output integrate**) the default output is in two columns with column 1 being the altitude and column 2 the heating rates. The output of **heating\_rate** can also be specified with the **output\_user** option. 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**, **Kato2.96**, and **LOWTRAN**. Attention: For spectral calculations, the extraterrestrial spectrum is assumed to be in mW/(m<sup>2</sup> nm).

Two different methods are implemented to calculate the heating rate, which can be selected with an optional keyword:

```
heating_rate [method]
```

where **method** may be either **layer\_cd** (heating rates are derived from centered differences of the flux, this is the default method) or **local** (heating rates are derived from the actinic flux). Attention: **heating\_rate local** introduces new levels into the profile which slightly affects the model output.

**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 cubic meter, and column 3 the effective particle radius in micrometer. The ice water content and effective 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. 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 layer properties. Before version 1.4 the default was level properties: The optical depth of a layer was calculated using information from the upper and lower levels defining the layer, see **ic\_layer** and **ic\_level**. To switch to the old behaviour, use **ic\_level**. See the section about ice 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 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<sup>-1</sup>.

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

**Column 4-(nmom+4):**

The moments of the scattering phase function.

Note that for **rte\_solver disort2** it makes good sense to make the number of moments larger than **nstr** because all moments are used in the calculation. 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 without delta-scaling. Please note that this was changed on July 22, 2008: Before, delta-scaling was switched on by default which might have caused some confusion, because irradiance calculations were not consistent with the other ice cloud parameterizations implemented in uvspec. Using the Fu (1996) parameterization in combination with one of **ic\_set\_tau/tau550/gg/ssa** or **ic\_scale\_gg/ssa** you now get consistent results with all other ice cloud parameterizations.

**ic\_fu\_reff**

If **ic\_fu\_reff fu** is specified, the parameterization uses the original definition of the effective radius as specified in Fu (1996/98). By default it uses the same definition of the effective radius as the Key et al. (2002), Yang et al. (2000) and Baum et al. (2005) parameterizations; 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 micrometer (rosette-4) while

others are only available for wavelengths larger than 3 micrometer (droxtal, spheroid).

#### `ic_ipa_files`

A two-column file, defining ice cloud property files (see `ic_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.

`ic_layer` Interpret ice cloud properties as layer properties (this is the default behaviour since version 1.4; see also `ic_file`). Cloud properties are assumed to be constant within each layer. The layer reaches from the level where the properties are defined in the `ic_file` to the level above. For example, the following lines

#	z	IWC	R_eff
#	(km)	(g/m <sup>3</sup> )	(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.

`ic_level` Interpret cloud properties as level properties (this was the default behaviour before version 1.4; see also `ic_file`). If `ic_level` is defined, a `ic_file` would be interpreted as follows:

#	z	IWC	R_eff
#	(km)	(g/m <sup>3</sup> )	(um)
	5.000	0	0
	4.000	0.2	12.0
	3.000	0.1	10.0
	2.000	0.1	8.0

The value 0.2 g/m<sup>3</sup> 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.

#### `ic_no_scattering`

Switch off scattering by ice clouds.

#### `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), whereas uvspec uses non delta-scaled properties unless the option `ic_fu_tau scaled` is specified. By default the parameterization by Fu (1996) is treated consistently with all other ice cloud parameterizations. For wavelengths up to 4 micrometer Fu (1996) is used while for wavelengths larger than 4 micrometer 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 linearly interpolated in between. If you use this option, please see also the discussion of `ic_fu_tau` and `ic_fu_reff`. The allowed range for the effective radius is from 9.315 - 65.120 micrometer.

- `echam4`      Use the 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 micrometer and 0.68 - 4.0 micrometer. Within the two ECHAM4 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 wavelength region from 0.2 to 5.0 micrometer and is available for the following habits: solid-column, hollow-column, aggregate, rosette-4, rosette-6, and plate.
- `yang`             Parameterization similar to Key et al. (2002) but based on more recent single scattering calculations. Below 3.4 micrometer it actually equals the Key et al. (2002) parameterization while from 3.4 - 100 micrometer new coefficients have been calculated with much higher wavelength resolution and better accuracy. Hence, yang should give a reasonably consistent approximation from 0.2 - 100 micrometer, suitable for spectrally resolved calculations of radiance and irradiance. The covered range for the effective radius depends on the `ic_habit`. (In micrometer: solid columns [5.96, 84.22], hollow columns [4.97, 70.24], rough aggregate [3.55, 108.10], rosettes-4



[2.77, 45.30], rosettes-6 [2.85, 46.01], plates [4.87, 48.18], dendrites [0.45, 1.88], droxtals [9.48, 293.32], spheroids [6.58, 203.39]).

**baum** Use ice cloud parameterization from Baum et al. (2005), <http://www.ssec.wisc.edu/~baum/Cirrus/IceCloudModels.html>. The phase function is parameterized by 128 Legendre coefficients, calculated with the delta-fit method from Hu et al. (2000). This parameterization covers the region from 0.4 to 2.2 micrometer. If very high accuracy is needed e.g. in the vicinity of the halo, the forward peak, or the backscatter peak, **ic\_properties baum\_detailed** is recommended.

**baum\_detailed** See **ic\_properties baum**; the phase function is parameterized by 10,000 Legendre coefficients which provides in most cases a more accurate representation of the Halo etc. For this option you need to obtain the respective tables from the libRadtran webpage (100 MBytes!).

**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 micrometer. For spectral or pseudo-spectral calculations **ic\_properties\_interpolate** has to be defined explicitly to initiate the interpolation of the optical properties to the internal wavelength grid. Please note that this option may be extremely memory-consuming 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 the mie-tool of the **libRadtran-tool** box (see **output\_user cloudprp**) or by Frank Evans' **cloudprp**; for each of the internal (computational) wavelengths, a separate file is expected (this option is subject to change).

The default property is **fu**.

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), and Baum et al. (2005) use the general definition

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

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  $20\mu\text{m}$  with `ic_properties fu` and `ic_fu_reff fu` and  $1.299 \cdot 20\mu\text{m} = 26\mu\text{m}$  with `ic_properties yang` would give comparable results for hexagonal columns. To use the original definition of the effective radius by Fu et al. (1996,1998) use `ic_fu_reff fu`!

#### `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-consuming because for each internal wavelength a full set of Legendre moments of the phase function is stored (up to several thousands).

#### `ic_saturate`

With this option, the relative humidity inside ice clouds can easily be adjusted to a user-defined value, e.g. saturated with respect to water. This option has one mandatory and one optional argument:

`ic_saturate switch [relative_humidity]`

where `switch` is `on`, `off`, or `ipa`. The second optional argument determines the relative humidity in % (with respect to water!) inside the cloud. The default value is 100. `ipa` is only relevant for independent column calculations.

Details for independent column simulations: Using `switch on`, the air in *all* columns will be saturated, if there is a cloud in at least one of the columns (this option should be used for stratiform clouds). Using `switch ipa`, only cloudy columns are affected (this option should be used for convective cloud fields.)

**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.02A.afgls.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.

`n2o_mixing_ratio`

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

`no_absorption`

Switch all (molecular, aerosol, cloud, and ice cloud) absorption off. Please note that this option simply sets the absorption optical thickness to 0. If used together with `xxx_set_tau` this might be a bit confusing but probably the most logical way. E.g. when using `aerosol_default` and `aerosol_set_tau 1`, the aerosol optical thickness is set to 1, with 0.940539 scattering and 0.059461 absorption. If `no_absorption` is added, the absorption optical thickness is set to 0 while the scattering optical thickness is preserved at 0.940539 (even though 1 was specified by the user). We find this the most logical solution of the problem because by switching `no_absorption` off and on one tests the effect of the absorber in an isolated way, rather than mixing absorption and scattering. The same is true for water and ice clouds. Note, that thermal emission of molecules is also switched off.

<b>no_molecular_absorption</b>	Switch off molecular absorption.
<b>no_rayleigh</b>	Switch Rayleigh scattering off.
<b>no_scattering</b>	Switch scattering off.
<b>no2_column_du</b>	Obsolete, use <b>dens_column</b> instead. Set the NO2 column to a given value. The column is integrated between the user-defined <b>altitude</b> and TOA (top of atmosphere). The value must be in Dobson units. If value is negative or <b>no2_column_du</b> is not specified, the NO2 column is not scaled.
<b>no2_column_moleccm-2</b>	Obsolete, use <b>dens_column</b> instead. Set the NO2 column to a given value. The column is integrated between the user-defined <b>altitude</b> and TOA (top of atmosphere). The value must be in molecules / cm2. If value is negative or <b>no2_column_moleccm-2</b> is not specified, the NO2 column is not scaled.
<b>nscat</b>	The order of scattering for the <b>sos</b> radiative transfer equation solver. Default is 20. May also be used with the <b>sdisort</b> solver. If set to 1 <b>sdisort</b> will run in single scattering mode while if set in to 2, <b>sdisort</b> runs in full multiple scattering mode.
<b>nrefrac</b>	For the <b>rte_solver sdisort</b> refraction may be included by specifying <b>nrefrac</b> . If refraction is included also set parameter <b>refraction_file</b> .  0            No refraction, default. 1            Refraction included using fast, but harsh method. 2            Refraction included using slow, but accurate method.
<b>nstr</b>	Number of streams used to solve the radiative transfer equation. Default is 6 for fluxes and 16 for radiances. (For <b>rte_solver disort</b> and <b>disort2</b> only even nstr are possible.)
<b>o2_mixing_ratio</b>	The mixing ratio of O2 in ppm; scale the profile so that the mixing ratio at the user-defined <b>altitude</b> assumes the specified value.
<b>o3_crs</b>	Choose between the following ozone cross sections.  <b>Bass_and_Paur</b> Bass and Paur ozone cross section.  <b>Molina</b> Molina and Molina (1986) ozone cross section.  <b>Daumont</b> Ozone cross section by Daumont et al. (1992), Malicet et al. (1995).  Molina and Molina is default.

**optimize\_fortran**

When this option is activated, the minimum dimensions of the FORTRAN arrays for the specific input conditions are determined and written to **stderr**.

**output**

Output processing. Choose between the following options:

**sum** Sum output over wavelength. Useful in combination with the **correlated\_k** option (Kato, Kato2, Kato2.96, Fu, AVHRR\_KRATZ).

**integrate**

Integrate output over wavelength for solar and over wavenumber for thermal simulations. Useful for spectral calculations and **correlated\_k** LOWTRAN.

**per\_nm** Output is given in W/(m<sup>2</sup> nm) or mW/(m<sup>2</sup> nm) (W or mW is determined by the extraterrestrial spectrum.)

**per\_cm-1** Output is given in W/(m<sup>2</sup> cm<sup>-1</sup>) or mW/(m<sup>2</sup> cm<sup>-1</sup>).

**per\_ck\_band**

Output is given in W/m<sup>2</sup> or mW/m<sup>2</sup> per correlated-k band. (This option can not be used for spectral calculations and **correlated\_k** LOWTRAN in the solar range.)

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

**output\_user**

User defined output. This option is case sensitive. 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 and the output of **rte\_solver polradtran** are not affected.

**lambda** Wavelength in nm.

**wavenumber**

Wave number in cm<sup>-1</sup>.

**sza** solar zenith angle

**zout** Output altitude in km.

**edir, eglo, edn, eup, enet, esum**

Direct, global, diffuse downward, and diffuse upward irradiance. Net is global - upward, sum is global + upward.

**uu** Radiances: **uu(umu(0),phi(0)) ... uu(umu(0),phi(m)) ... uu(umu(n),phi(0)) ... uu(umu(n),phi(m))**

**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.

**heat**          Heating rate in K/day.

It is also possible to gain some information about the atmosphere and the clouds:

**p**              pressure [hPa], ,

**T, T\_d**        temperature [K], dewpoint temperature [K]

**T\_sur**        surface temperature [K]

**theta**        potential temperature [K]

**theta\_e**      equivalent potential temperature [K]

**n\_xxx**        number density of the gas xxx [cm<sup>-3</sup>]

**rho\_xxx**      mass density of the gas xxx [kg/m<sup>3</sup>]

**mmr\_xxx**     mass mixing ratio of the gas xxx [kg/kg]

**vmr\_xxx**     volume mixing ratio of the gas xxx [m<sup>3</sup>/m<sup>3</sup>]

**rh**            relative humidity over water [percent]

**rh\_ice**       relative humidity over ice [percent]

**c\_p**           specific heat capacity of the air (humidity and temperature dependent)

**CLWC**        cloud liquid water content [g/m<sup>3</sup>]

**CIWC**        cloud ice water content [g/m<sup>3</sup>]

**TCC**         total cloud cover [0-1]

where **xxx** is one of AIR, O3, O2, H2O, CO2, NO2, BRO, OCLO, or HCHO.

#### **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.

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

```

Sensor in the North (looking South)  0 deg
Sensor in the East  (looking West)   90 deg
Sensor in the South (looking North) 180 deg
Sensor in the West  (looking East)  270 deg

```

For all one-dimensional solvers the absolute azimuth does not matter, but only the relative azimuth **phi-phi0**.

**phi0**          Azimuth angle of the sun (0 to 360 degrees).

Sun in the South	0 degrees
Sun in the West	90 degrees
Sun in the North	180 degrees
Sun in the East	270 degrees

For all one-dimensional solvers the absolute azimuth does not matter, but only the relative azimuth `phi-phi0`.

#### `polradtran_aziorder`

Order of Fourier azimuth series: 0 (default for irradiance) is azimuthally symmetric case. For radiance computation a higher order is required, thus the default for radiances is 4. This option is only relevant for `rte_solver polradtran`.

#### `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. This option is only relevant for `rte_solver polradtran`.

#### `polradtran_nstokes`

Number of Stokes parameters

- 1 for I (no polarization, default)
- 2 for I,Q,U (Since V is very small in the atmosphere, it makes sense to compute only I,Q,U. This saves computation time and memory).
- 3 for I,Q,U,V

Default is 1.

#### `polradtran_quad_type`

Type of quadrature used:

- G gaussian
- D double gaussian,
- L Lobatto

E (default)

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

Default E. This option is only relevant for `rte_solver polradtran`.

#### `polradtran_src_code`

Radiation sources included:

- 0 none
- 1 solar
- 2 thermal
- 3 both



Default 1. This option is only relevant for `rte_solver polradtran`.

**pressure** The surface pressure (at the user-defined `altitude`) in hPa. The pressure profile as well as air, O2 and CO2 density profiles are scaled accordingly.

**pressure\_out**

Specify the output levels in pressure coordinates. The syntax is

```
pressure_out p1 p2 ...
```

where '`p1 p2 ...`' are the output levels in hPa. The pressure output levels must be sorted in decreasing order. Output pressure levels must be within the range defined in the `atmosphere_file`. You can also use `toa` for top of atmosphere and `sur` for surface altitude and `cpt` for cold point tropopause.

**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 specified, informative messages are turned off. See also `verbose`.

**radiosonde**

This option allows to change the temperature and pressure profile, and optionally to specify one or more density profiles. The entry in the input file looks like this:

```
radiosonde filename [gas_species] [unit] ...
```

Currently the following `gas_species` are included: ozone (O3), nitrogen dioxide (NO2), water vapor (H2O), bromine oxide (BRO), chlorine dioxide (OCLO), formaldehyde (HCHO), and carbon dioxide (CO2). Each gas species is identified by its abbreviations given in parentheses above. Unit is an optional argument to defines the unit of the density. The profiles can be given in particles per cm3 (CM-3), in particles per m3 (M-3), as volume mixing ratio (VMR), as mass mixing ratio (MMR), or as relative humidity (RH) (only for water). The default unit is RH for water vapour, MMR for ozone, and CM-3 for all other gases. The radiosonde file must have (2 + number of gases) columns:

- 1            pressure in hPa
- 2            temperature in Kelvin
- 3, 4 ...    density of trace gas in the specified unit

A new z-grid will be calculated, starting at `altitude` and assuming a linear temperature variation between levels. The air density will be recalculated according to the ideal gas law, and the density of the well mixed gases O2 and CO2 will be scaled accordingly. The atmospheric data above the radiosonde data is taken from the `atmosphere_file` level by level, starting at the first pressure level above the radiosonde data. The z-grid of the `atmosphere` file in this height region is shifted accordingly. Also if the density in the radiosonde file is specified as -1 at a level, the value from the `atmosphere_file` is used. Possible calls are

`radiosonde ../examples/radiosonde.dat`  
 just in order to change the temperature and pressure profile, or  
`radiosonde ../examples/radiosonde2.dat H2O RH O3 MMR NO2`  
 where water vapour density will be given as relative humidity, ozone as mass mixing ratio, and NO2 in cm-3 (default).

#### `radiosonde_levels_only`

The atmosphere considered in the simulation has the same height range as the data in the `radiosonde`-file. No further levels are added above those. This option has only an effect in combination with `radiosonde`.

#### `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.

`Penndorf` Penndorf (1957) Rayleigh scattering cross section.

Bodhaine et al. is default.

#### `rayleigh_depol`

Rayleigh depolarization factor; the Rayleigh scattering phase function is  $p(\mu) = a + b \cdot \mu^2$  where  $a = 1.5 \cdot (1 + \text{depol}) / (2 + \text{depol})$  and  $b = 1.5 \cdot (1 - \text{depol}) / (2 + \text{depol})$ . 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  $\theta$  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  $\theta$  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 is 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 linearly 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 Meerkotter, 2000). A wavelength dependent sigma is not yet available.
- rpv\_t1** Constant RPV t1, to be used for snow (Deguenther and Meerkotter, 2000). A wavelength dependent sigma is not yet available.
- rpv\_t2** Constant RPV t2, to be used for snow (Deguenther and Meerkotter, 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`.
- rpv\_library** The rpv libraries are collections of spectral BRDFs of different surface types, This option must be used either with `surface_type` or `surface_type_map`, in order to select the specific surface type.  
For using a `rpv_library` write

**rpv\_library library\_path**

where **library\_path** is the path of the directory, where the BRDF data is stored. The files are expected to have the names **IGBP.01.rpv**, **IGBP.02.rpv**, ... If **surface\_type 1** is specified the BRDF from **IGBP.01.rpv** will be used, and so on. Each file must have the structure like an **rpv\_file**. (This option is quite the same as **rpv\_file**, except that it offers you an easy way to use the option **surface\_type\_map** in combination with your **rpv\_files**.)

**rte\_solver**

Set the radiative transfer equation solver to be used. If not specified the default **rte\_solver** is **disort2**. 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/](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. For your application please use **rte\_solver disort2** which is the advanced version, unless you e.g. want to explore how a specific feature of disort2 (e.g. the Nakajima intensity correction) improves the disort result.

**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/](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. The full implementation of the pol-

Radtran solver is quite new (version 1.4). If you find unusual behaviour, please contact the libRadtran authors.

- |                         |  |
|-------------------------|--|
| <code>twostr</code>     | The two-stream radiative transfer solver described by Kylling et al. (1995), in pseudo-spherical geometry.   |
| <code>twostrpp</code>   | The two-stream radiative transfer solver described by Kylling et al. (1995), in plane-parallel geometry.   |
| <code>sos</code>        | A scalar pseudospherical successive orders of scattering code. Works for solar zenith angles smaller than 90 degrees. Can calculate azimuthally averaged radiances. Set <code>nscat</code> to specify the order of scattering.   |
| <code>montecarlo</code> | The MYSTIC code, see <a href="http://www.bmayer.de/mystic.html">http://www.bmayer.de/mystic.html</a> . Note that MYSTIC is not part of the libRadtran distribution at present. However, it has been given to some users on a collaborative basis.  |
| <code>tzs</code>        | 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.   |
| <code>sss</code>        | SSS stands for "solar, single scattering" and is an analytical single scattering approximation which might be reasonable for an optically thin atmosphere. Please note that SSS does only radiance calculations at top of the atmosphere. This is an experimental solver - be careful!   |
| <code>null</code>       | 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 <code>verbose</code> . |

Default: `disort2`

#### `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. Please note that prior to convolution the spectrum is interpolated to the wavelength steps of the slit function. For this reason, make sure that the resolution of the slit function is high enough even if the slit function is e.g. a simple triangle which could in principle be described with 3 grid points.

**solar\_file**

Location of file holding the extraterrestrial spectrum.

```
solar_file filename [unit]
```

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 chooses for the radiative transfer calculation. Comments start with **#**. Empty lines are ignored.

For some purposes it is useful to tell libRadtran the units of the spectrum. This can be done with the optional second argument. If **unit** is set to **per\_nm** libRadtran assumes that the unit of the spectrum is W/(m<sup>2</sup> nm), if set to **per\_cm-1** it assumes W/(m<sup>2</sup> cm<sup>-1</sup>). Note that **solar\_file** is ignored if **correlated\_k** is specified.

**source**

Solar or thermal source. Set to either **solar** or **thermal**. Solar radiation is per default output in W/(m<sup>2</sup> nm) for spectral and **correlated\_k** LOWTRAN calculations. For all other **correlated\_k** options the output is integrated over the wavelength band. Thermal radiation is per default output in W/(m<sup>2</sup> cm<sup>-1</sup>), if the bandwidth is equal to 1 cm<sup>-1</sup> (default for **correlated\_k** LOWTRAN calculations). Otherwise the output is the integrated flux over the wavenumber interval specified by **thermal\_bandwidth**, **thermal\_bands\_file**, or by the **correlated\_k** option (Kato, Kato2, Kato2.96, Fu, AVHRR\_KRATZ, or Generic).

**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 290.0, 290.5, 291.0, ..., 364.5, 365.0. 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.

**surface\_temperature\_map**

Specify a surface-temperature map with a *netCDF* file which is used in combination with the options **latitude**, **longitude**, and **time**.

`surface_temperature_map file [variable_name]`

where `file` is the location of the *netCDF* file. `libRadtran` reads the value at the nearest pixel to the given `latitude` and `longitude`. No spatial interpolation or averaging of the values is done.

#### `surface_type`

With this option the `surface_type` is selected. This option can be used with `albedo_library` in order to select an spectral albedo or with `rpv_library` in order to select a BRDF function.

`surface_type surface_type_number`

where `surface_type_number` is an integer starting from 0, where 0 refers to a black surface and the following numbers to the entries in the specified `library`.

#### `surface_type_map`

Specify a surface type map, which is used in combination with `albedo_library`, `latitude`, and `longitude` in order to select the surface type relevant for the simulation. No pixel interpolation is done. The format of the call is:

`surface_type_map file [variable_name]`

where `file` is the location of the surface type map file. The map is expected to be in *netCDF* format. The file must contain the variables `double lat(nlat)`, `double lon(nlon)`, and `byte surface_type (nlat, nlon)`. If the name of the surface type variable is different, the optional argument can be used in order to specify the variable name. For format specification see also `data/albedo/IGBP/SURFACE.TYPE.IGBP.10min.cdf`.

For using the IGBP map, the call is `surface_type_map IGBP`. This map has a resolution of 10 minutes and contains the surface types 1 to 18 defined in the `albedo_library` IGBP. Fresh snow and sea ice are not included, as their extent is too variable. Attention: That implies e.g. that the Arctic is considered `ocean_water` and not `sea_ice`!

Locations on the pixel boundaries are interpreted as the pixel northward and eastward respectively. E.g. location 0 N, 0 E is interpreted like the pixel ranging from 0 to 10min North and from 0 to 10min East.

**sza**      The solar zenith angle (degrees). The default solar zenith angle is 0.

**sza\_file**      Location of solar zenith angle file for wavelength-dependent solar zenith angle. This is actually useful if you want to simulate an instrument which scans so slowly that the solar zenith angle may change significantly during the wavelength scan. 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<sup>-1</sup> for thermal calculations. The default is 1 cm<sup>-1</sup>. This option is ignored if used together with **correlated\_k** KATO/KATO2/KATO2.96/FU/AVHRR\_KRATZ.

**time**

Specifies the time to simulate.

**time** YYYY MM DD hh mm ss

where YYYY is the year, MM the month, DD the day, hh the hour, mm the minute, ss the second in UTC. The time information will be used for a couple of things: **time** in combination with **latitude**, **longitude**, and any **map**-option is used to select the location where to read the input data.

**time** is used to correct extraterrestrial irradiance for the Sun-Earth distance with the day of year. If not given, the Earth-Sun distance is 1 AU (i.e. equinox distance).

**time** in combination with **latitude** and **longitude** is used to calculate the solar zenith angle if no **sza** is specified.

**time** in combination with **latitude** and **longitude** is used to choose a suitable default atmosphere file, if no **atmosphere\_file** is specified.

**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 (downwelling radiance, looking upward) values (if any) and on through positive (upwelling radiance, looking downward) values. Must not be zero.
- 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` (depending on your shell you might need a slightly different syntax). The irradiances and radiances will be written to 'uvspec.out' while all diagnostic messages go into 'verbose.txt'. See also **quiet**.
- 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.
- 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 (LWC) in grams per cubic meter, and column 3 the effective droplet radius in micrometer. Empty lines are ignored. Comments start with **#**. 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. An example of a cloud is given in 'examples/WC.DAT'.
- Per default the cloud properties are interpreted as layer properties. Before version 1.4 the default was level properties: The optical depth of a layer was calculated using information from the upper and lower levels defining the layer, see **wc\_layer** and **wc\_level**. To switch to the old behaviour, use **wc\_level**. 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 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<sup>-1</sup>.

**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** Interpret cloud properties as layer properties (this is the default behaviour since version 1.4; see also **wc\_file**). 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\_level** Interpret cloud properties as level properties (this was the default behaviour before version 1.4; see also **wc\_file**). If **wc\_level** is defined, a **wc\_file** would be interpreted as follows:

```
#      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 value 0.2 g/m<sup>3</sup> 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_no_scattering`

Switch off scattering by water clouds.

#### `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 micrometer and 0.68 - 4.0 micrometer; 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 micrometer. For spectral or pseudo-spectral calculations `wc_properties_interpolate` has to be defined explicitly to initiate the interpolation of the optical properties to the internal wavelength grid. Please note that this option may be extremely memory-consuming because for each internal wavelength a full set of Leg-

endre 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 the mie-tool of the **libRadtran-tool** box (see **output\_user cloudprp**) or 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-consuming because for each internal wavelength a full set of Legendre moments of the phase function is stored (up to several thousands).

#### **wc\_saturate**

With this option, the relative humidity inside water clouds can easily be adjusted to a user-defined value, e.g. saturated with respect to water. This option has one necessary and one optional argument:

**ic\_saturate switch [relative\_humidity]**

where **switch** is **on**, **off**, or **ipa**. The second optional argument determines the relative humidity in % (with respect to water!) inside the cloud. The default value is 100. **ipa** is only relevant for independent column calculations. Details for independent column simulations: Using **switch on**, the air in *all* columns will be saturated, if there is a cloud in at least one of the columns (this option should be used for stratiform clouds). Using **switch ipa**, only cloudy columns are affected (this option should be used for convective cloud fields.)

#### **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.

**zout**

This option is used to specify the output altitudes in km *above surface altitude*. One or more altitudes may be specified in increasing magnitude.

```
zout 0 1 2 3 4 5 ...
```

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 do 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**. You can also use **toa** for top of atmosphere and **sur** for surface altitude and **cpt** for cold point tropopause.

**zout\_sea** like **zout**, but *above sea surface*

**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:

```
z_interpolate property interpolation_method
```

Properties which may be specified are:

O3, O2, H2O, CO2, NO2, BRO, OClO, HCHO

T temperature (here linmix is not suitable)

Possible interpolation methods are:

- linear** The specified property (number density of the gas or temperature) varies linearly with height.
- log** The specified property (number density of the gas or temperature) varies logarithmically with height. 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 logarithmically varying air density) varies linearly with height.

For all gas densities the default interpolation method is **linmix**, for temperature it is **linear**.

#### **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** option 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 optical properties are determined from the interpolated atmospheric properties. If **heating\_rate** or **rte\_solver polradtran** is specified, **zout\_interpolate** will also be automatically activated. **zout\_interpolate** generally causes smoother variation of the optical properties.

## **2.2 mie**

**mie** performs 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 separated 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.

#### `aerosol_type`

With this option Mie calculations are performed for the specified aerosol type. The aerosol properties (refractive index, size distribution, density, humidity) are taken from the OPAC database (see Hess et.al., 1998). Possible types are

<code>inso</code>	Water insoluble aerosol consists mostly of soil particles with a certain amount of organic material.
<code>waso</code>	Water soluble aerosol originates from gas to particle conversion and consists of various types of sulfates, nitrates, and other, also organic water-soluble substances.
<code>soot</code>	Soot is absorbing black carbon, which is not soluble in water. In reality soot particles have a chain-like character, which of course is not accounted for in Mie calculations of optical properties. The optical properties are calculated assuming many very small spherical particles.
<code>ssam</code>	Sea salt particles consist of the various kinds of salt contained in seawater. The different modes are given to allow for a different wind-speed-dependant increase of particle number for particles of different size. This aerosol type represents the accumulation mode.
<code>sscm</code>	Sea salt particles (coarse mode).
<code>minm</code>	Mineral aerosol or desert dust is produced in arid regions. It consists of a mixture of quartz and clay minerals and is modeled with three modes to allow to consider increasing relative amount of large particles for increased turbidity. This aerosol type represents the nucleation mode.
<code>miam</code>	Mineral aerosol (accumulation mode).
<code>micm</code>	Mineral aerosol (coarse mode).
<code>mitr</code>	Mineral transported is used to describe desert dust that is transported over long distances with a reduced amount of large particles.
<code>suso</code>	The sulfate component is used to describe the amount of sulfate found in the Antarctic aerosol. This component is not suited to describe antropogenic sulfate aerosols that are included in the water-soluble component.

**distribution**

If specified the effective radius is converted into a size distribution of droplets.

**distribution** **distribution\_type** **distribution\_parameter**

where distribution type is one of the two following:

**GAMMA**      The Gamma distribution of cloud droplet sizes is

$$n(r) = a r^{\alpha} \exp(-b r),$$

where alpha is the distribution parameter given as second argument. (a and b are determined automatically.) The effective radius of the distribution is  $r_{\text{eff}} = (\alpha+3)/b$ . A typical value for water clouds is alpha=7. For ice clouds a typical value is alpha=1. A large value of alpha gives close to a monodisperse distribution.

**LOGNORMAL**

The lognormal distribution of cloud droplet sizes is

$$n(r) = a/r \exp(-[\ln(r/r_0)]^2 / [2 \sigma^2]),$$

where r0 is the logarithmic mode of the distribution (calculated automatically) and sigma is the standard deviation, which is given by second argument. The effective radius of the distribution is  $r_{\text{eff}} = r_0 \exp(2.5 \sigma^2)$ . A common value for water clouds is sigma=0.35.

**mass\_density**

Specifies the mass density of the medium. Useful in combination with **refrac** **user** and **output\_user** **cloudprop**, as the format of **cloudprop** specifies the extinction coefficient per mass and not per volume as usual in this mie program.

**mie\_program**

Specify which Mie program to use:

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

**MIEVO**    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** **MIEVO**.

**nmom**      Number of moments of the phase function to be calculated (default: 0). Only possible with **mie\_program** **MIEVO**.

**nstokes**    Number of Stokes parameters (default: 1). For **nstokes=1** the Legendre polynomials of the phase function will be calculated. To calculate all phase matrix elements required for polarized radiative transfer, set **nstokes=4**.



**output\_user**

The mie output is one line of output quantities to standard output (stdout) for each wavelength and each particle radius. With this option the user may specify the columns desired for output:

```
output_user output_1 output_2 ... output_n
```

where output\_i is one of following arguments:

**lambda**      Wavelength in nm.

**wavenumber**  
Wave number in cm-1.

**r\_eff**      particle radius in micro meter.

**refrac\_real**  
The real part of the refractive index.

**refrac\_imag**  
The imaginary part of the refractive index.

**qext**      The extinction efficiency factor, if **r\_eff** 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 content 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).

**qsca**      The scattering efficiency factor, if **r\_eff** was specified, or the extinction coefficient [km-1] per unit concentration [cm3/m3], if a **size\_distribution\_file** was specified.

**omega**      The single scattering albedo.

**gg**      The asymmetry parameter.

**sforw**      (Complex) forward-scattering amplitude S1 at 0 degrees.

**sback**      (Complex) back-scattering amplitude S1 at 180 degrees.

**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** The nmom+1 moments (from 0 to nmom, see option **nmom**) 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.

**cloudprp** This is a special option which, if specified, must be the only option of **output\_user**, as **cloudprp** specifies a whole format of the output. In particular this option is useful when a correlated-k wavelength grid is specified with **wavelength**. If specified, the output will be written in a format, which can be directly used by libRadtran. See also output **netcdf**. See libRadtran-manual **ic\_properties** and **wc\_properties** and there the items **mie** and **filename**.

**aerosolprp**

This option is similar to the **cloudprp** option. The only difference is that the effective radius dimension is replaced by humidity values of the aerosol.

**netcdf** This option writes the same output as **cloudprp** and **aerosolprp** directly in netCDF format. The outputfiles can be used by **uvspec** using the options **ic\_properties** and **wc\_properties**.

The default output is:

```
lambda refrac_real refrac_imag qext omega gg spike pmom
```

**r\_eff** The radius [micron] of the particle to calculate single scattering properties of. Used together with the wavelength information to calculate the Mie size parameter.

```
r_eff radius
```

The user can optionally specify a 2nd and 3rd argument to make a loop over several radii:

```
r_eff radius_min radius_max radius_step
```

First calculations is done with **radius\_min**, which will be increased by **radius\_step** until **radius\_max** is reached.

**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.

**wavelength lambda\_min lambda\_max**

The wavelength step is specified by **wavelength\_step**.

For unregular wavelength grid it is also possible to specify a file, where the wavelengthgrid is stored.

**wavelength wvl\_filename**

where **wvl\_filename** is the path and name of the file, which contains the wavelength grid. It is expected that the wavelength values in nm are stored in the second column. For the **correlated\_k** schemes implemented in **uvspec** you can use following abbreviations instead of a filename (in this case **data\_files\_path** must be specified also): **kato**, **kato2**, **kato2.96**, **fu**, and **avhrr\_kratz**.

This option is ignored if **refrac** file is specified.

**wavelength\_step**

The wavelength step, in nm. Ignored if **refrac** file is specified.

**wavelength\_index**

Set the wavelengths to be selected. This might be the normal wavelength grid defined by **wavelength** and **wavelength\_step** or a **correlated\_k** wavelength grid. E.g., in combination with **wavelength 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.

## 2.2.2 The mie output

standard output (stdout) of the mie program is one line for each wavelength and each effective radius. The format of the output line is

**lambda refrac\_real refrac\_imag qext omega gg spike pmom**

The keywords here are the same as in input option **output\_user**.

### 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          Function
                        columns, int length)
```

**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\_malloc\_string.

Parameters:

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

### 3.1.3.4 Function ASCII\_malloc\_int

**int ASCII\_malloc\_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. index range of the last index is variable

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\_4D\_arylen

**int ASCII\_calloc\_double\_4D\_arylen** (double \*\*\*\*\*value, int rows, int columns, int levels, int \*length) Function

Description:

Allocate memory for a four-dimensional array of double. index range of the last index is variable

Parameters:

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author: Ulrich Hamann

### 3.1.3.9 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

### 3.1.3.10 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

### 3.1.3.11 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

### 3.1.3.12 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

### 3.1.3.13 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

### 3.1.3.14 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:



### 3.1.3.15 Function ASCII\_calloc\_short

**int ASCII\_calloc\_short** (short \*\*\*value, int rows, int columns)

Function

Description:

Allocate memory for a two-dimensional array of short.

Parameters:

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

### 3.1.3.16 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:

### 3.1.3.17 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:

### 3.1.3.18 Function ASCII\_free\_float

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

Function

Description:

Free memory, which has been allocated with ASCII\_malloc\_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:

### 3.1.3.19 Function ASCII\_free\_short

```
int ASCII_free_short (short **value, int rows)
```

Function

Description:

Free memory, which has been allocated with ASCII\_malloc\_short.

Parameters:

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

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

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:

### 3.1.3.20 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

### 3.1.3.21 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

### 3.1.3.22 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

### 3.1.3.23 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

### 3.1.3.24 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

### 3.1.3.25 Function `ASCII_free_float_6D`

```
int ASCII_free_float_6D (float *****value, int rows, int          Function
                        columns, int length, int fourth_dimension, int fifth_dimension)
```

Description:

Free memory, which has been allocated with `ASCII_calloc_float_6D`.

Parameters:

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

Example:

Files: none

Known bugs:  
none

Author: Ulrich Hamann (modified `ASCII_free_float_5D`, Arve Kylling)

### 3.1.3.26 Function `ASCII_free_float_7D`

```
int ASCII_free_float_7D (float *****value, int rows, int          Function
                        columns, int length, int fourth_dimension, int fifth_dimension, int
                        sixth_dimension)
```

Description:

Free memory, which has been allocated with `ASCII_calloc_float_7D`.

Parameters:

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

Example:

Files: none

Known bugs:  
none

Author: Ulrich Hamann (modified `ASCII_free_float_5D`, Arve Kylling)

### 3.1.3.27 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:

### 3.1.3.28 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:

### 3.1.3.29 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:

### 3.1.3.30 Function ASCII\_file2string

**int ASCII\_file2string** (char \*filename, int \*rows, int \*max\_columns, int \*min\_columns, char \*\*\*string) 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; rows with less than max\_columns columns are filled up with ""; the allocated memory can be freed with ASCII\_free\_string (string, rows, max\_columns)

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

**char \*\*\*string**

Pointer to a two-dimensional array of strings, string [0 ... rows-1][0 ... max\_columns-1][0...n\_character-1] Memory is allocated automatically.

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author: Ulrich Hamann, 17.12.2007

### 3.1.3.31 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:

### 3.1.3.32 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:

### 3.1.3.33 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:

### 3.1.3.34 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:

### 3.1.3.35 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:

### 3.1.3.36 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:

### 3.1.3.37 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:

### 3.1.3.38 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:

### 3.1.3.39 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:

### 3.1.3.40 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:

### 3.1.3.41 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:****3.1.3.42 Function read\_4c\_file**

```
int read_4c_file (char *filename, double **first, double Function  

                 **second, double **third, double **fourth, int *n)
```

**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:****3.1.3.43 Function read\_4c\_file\_float**

```
int read_4c_file_float (char *filename, float **first, float Function  

                      **second, float **third, float **fourth, int *n)
```

**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:

### 3.1.3.44 Function read\_5c\_file

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

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:

### 3.1.3.45 Function read\_5c\_file\_float

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

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:

### 3.1.3.46 Function read\_6c\_file

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

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:

### 3.1.3.47 Function read\_7c\_file

```
int read_7c_file (char *filename, double **first, double                Function
                  **second, double **third, double **fourth, double
                  **sixth, double **seventh, int *n)
```

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:

### 3.1.3.48 Function read\_8c\_file

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

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 eighth. 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:

### 3.1.3.49 Function read\_9c\_file

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

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, eighth, 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:



### 3.1.3.50 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:

### 3.1.3.51 Function ASCII\_parse

**int ASCII\_parse** (char \*string, char \*separator, char \*\*\*array, Function  
int \*number)

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:

### 3.1.3.52 Function ASCII\_parsestring

**int ASCII\_parsestring** (char \*string, char \*\*\*array, int Function  
\*number)

**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:

<code>cnv.c</code>	Data convolution.
<code>equation.c</code>	Solve systems of linear equations.
<code>function.c</code>	Miscellaneous functions.
<code>integrat.c</code>	Numerical integration.
<code>linear.c</code>	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,                      Function  
                  double \*x\_conv, double \*y\_conv, int conv\_num, double \*\*x\_spec\_conv,  
                  double \*\*y\_spec\_conv, int \*spec\_conv\_num)

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, Function  
double \*x\_conv, double \*y\_conv, int conv\_num, double \*\*y\_spec\_conv)

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 function to be convoluted 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,  $A \cdot x = b$ , 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,  $A \cdot x = b$ , 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-friendliness. 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-Mueller, 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-friendliness. 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 double. Same function as `solve_three()`, but much more memory efficient on the cost of some user-friendliness. In contrast to `solve_three()`, A is defined as a  $n \times 3$ -matrix, rather than a  $n \times n$ -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_float`

**float `integrate_float`** (float \*x\_int, float \*y\_int, int number)

Function

Description:

Same as `integrate`, but takes float instead of double as input

Parameters:

**float \*x\_int**  
x values of the data points.

**float \*y\_int**  
y values of the data points.

**int number**  
Number of data points.

Return value:

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

Example:

Files: none

Known bugs:  
none

Author:

### 3.2.3.15 Function `integrate_spline`

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

Function

Description:

Calculate integral  $\int y(x) dx$  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 \*y** 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 \*integral**  
Calculated integral.

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:  
none

Author:

### 3.2.3.16 Function `linear_coeffc`

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

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:

### 3.2.3.17 Function `linear_coeffc_fast`

`int linear_coeffc_fast (double *x, double *y, int number, double **a0, double **a1)` Function

Description:

Calculate coefficients for linear interpolation; memory for coefficients will be allocated automatically! In contrast to `linear_coeffc()`, only `a0` and `a1` are used. The interpolation may be done with `calc_linear_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`

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:  
               none

Author:

### 3.2.3.18 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:

### 3.2.3.19 Function `calc_splined_value`

```

int calc_linear_value (double xnew, double *ynew, double *x, int      Function
    number, double *a0, double *a1)

```

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:

### 3.2.3.20 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:

### 3.2.3.21 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  $y = a + b \cdot x$  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:

### 3.2.3.22 Function `weight_regression`

```

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

```

Description:

Calculate coefficients for  $y = a + b \cdot x$  by *weighted* linear regression. Each data point is weighted with  $(1 / \text{sigma}[i]**2)$

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:

### 3.2.3.23 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:

### 3.2.3.24 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:

### 3.2.3.25 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:

### 3.2.3.26 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:

### 3.2.3.27 Function `appspl_coeffc`

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

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:

### 3.2.3.28 Function `calc_splined_value`

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

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:

### 3.2.3.29 Function `fcalcSplinedValue`

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

Description:

Similar to `calcSplinedValue`, only all data are of type float instead of double. Interpolate/approximate data to `xnew`. Basis are the spline coefficients which have been determined by `fsplineCoeff`.

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:

### 3.2.3.30 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 linearly 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**      Function
                                ary_2D)
```

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, Function  
double\* ary\_1D, double\*\* ary\_2D)

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\* Function  
ary\_1D )

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  $E0 = (r_0/r)^2$  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 occurrence.

int *time2
    2nd time of occurrence.

```

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          Function
               program, int medium, mie_complex crefin, float wavelength, float
               radius, float temperature, int nstokes, mie_complex *ref)
```

## 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 or AEROSOL, 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      Function
    *output, int program, int medium, mie_complex crefin, float
    wavelength, float temperature, int nstokes, double *x_size, double
    *y_size, int n_size, double *beta, double *omega, double *g,
    mie_complex *ref)
```

## 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 `read_mie_table`

**int `read_mie_table`** (char \*filename, double \*r0, double \*dr,  
size\_t \*nreff, double \*\*reff, double \*wavelen, double \*nre, double  
\*nim, double \*\*extinc, double \*\*albedo, double \*\*f, int \*\*nleg, float  
\*\*\*\*legen, size\_t \*nphase, int aerosol, int nstokes, 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 src/cloudprp2cdf.sh which converts the ASCII to the netCDF version. ATTENTION: Frank Evans' stores Legendre coefficients f(l), not moments p(l):  $f = p \cdot (2l + 1)$

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 *reff
    effective radius grid

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

double **f
    array[0 ... nreff-1] of delta scaling factors

int **nleg
    array[0 ... nreff-1] storing the number of Legendre coeff.

float ***legen
    array[0 ... nreff-1][0...ip][0 ... nleg] of Legendre coefficients

size_t nphase
    number of phase matrix elements

int aerosol
    aerosol flag

int nstokes
    number of Stokes components in RTE calculation

int quiet 'Shut up!' flag

```

Return value:

0 if o.k., <0 if error

Example:

Files: none

Known bugs:

none

Author:



### 3.5.3.5 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          Function
             reverse, char linear, double **irradiance_shifted)
```

Description:  
Shift a spectrum from vacuum to air or vice versa. The shifted data are re-gridded 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 asymmetry 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, wcon,                      Function  
                       wceffr, wc\_dtau, wc\_gg, wc\_ssa, zd, wclyr)

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 **wcon** 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(\*) **wcon**

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.
- Belward, A., and T. Loveland, 'The DIS 1km Land Cover Data Set, global change', 1996 The IGBP Newsletter, #27, see also: <http://www-surf.larc.nasa.gov/surf/pages/ftp-page.html>
- Baum, B. A., A. J. Heymsfield, P. Yang, and S. T. Bedka, 'Bulk scattering models for the remote sensing of ice clouds. Part 1: Microphysical data and models', 2005, J. Appl. Meteor., 44, 1885-1895
- Baum, B. A., P. Yang, A. J. Heymsfield, S. Platnick, M. D. King, Y.-X. Hu, and S. T. Bedka, 'Bulk scattering models for the remote sensing of ice clouds. Part 2: Narrowband models', 2005, J. Appl. Meteor., 44, 1896-1911.
- Baum, B. A., P. Yang, S. L. Nasiri, A. K. Heidinger, A. J. Heymsfield, and J. Li, 'Bulk scattering properties for the remote sensing of ice clouds. Part 3: High resolution spectral models from 100 to 3250 cm<sup>-1</sup>.', 2007, J. Appl. Meteor. Clim., Vol. 46, 423-434.
- 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, 35, 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.
- Hess, M., P. Koepke, and I. Schult, 'Optical properties of aerosols and clouds: the software package OPAC', *Bull. Am. Met. Soc.*, 79, 831-844, 1998.
- 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.
- Hu, Y.X., B. Wielicki, B. Lin., G. Gibson, S. Tsay and K. Stamnes, 2000, 'A fast and accurate treatment of cloud particle scattering phase functions: singular value decomposition least square fitting', *J. Quant. Spectrosc. Radiat. Transfer*, 65, 681-690.
- 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.
- Wanner, W., X. Li, and A.H. Strahler, 1995, 'On the derivation of kernel-driven models of bi-directional reflectance', *Journal of Geophysical Research*, 100(D10), 21077-21089.
- Wanner, W., A.H. Strahler, B. Hu, P. Lewis, J.-P. Muller, X. Li, C.L. Barker Schaaf, and M.J. Barnsley, 1997, 'Global retrieval of bidirectional reflectance and albedo over land from EOS MODIS and MISR data: Theory and algorithm', *Journal of Geophysical Research*, 102(D14), 17143-17161.
- 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

<b>Preface</b> .....	<b>1</b>
Acknowledgements .....	1
<b>1 A Brief Overview of libRadtran</b> .....	<b>3</b>
1.1 Radiative transfer calculations .....	3
1.2 Ozone retrieval from global irradiance measurements .....	3
1.3 Cloud optical thickness from global irradiance measurements .....	4
1.4 ... and much more .....	4
<b>2 Some useful tools</b> .....	<b>5</b>
2.1 uvspec .....	5
2.1.1 The uvspec input file .....	5
2.1.1.1 Cloudless, aerosol-free atmosphere .....	5
2.1.1.2 Spectral resolution .....	8
2.1.1.3 Aerosol .....	14
2.1.1.4 Water clouds .....	16
2.1.1.5 Ice clouds .....	17
2.1.1.6 Calculation of radiances .....	19
2.1.2 The uvspec output .....	19
2.1.2.1 DISORT, SDISORT and SPSPDISORT ..	19
2.1.2.2 TWOSTR .....	20
2.1.2.3 POLRADTRAN .....	20
2.1.2.4 Description of symbols .....	20
2.1.3 Complete description of input parameters .....	21
2.2 mie .....	60
2.2.1 The mie input file .....	60
2.2.2 The mie output .....	65
2.2.3 Examples of mie input files .....	66
2.3 integrate .....	66
2.4 spline .....	66
2.5 conv .....	66
2.6 addlevel .....	66
2.7 snowalbedo .....	66
2.8 cldprp .....	67
2.9 Geno3tab .....	67
2.9.1 Simple wavelength ratios with Gen_o3_tab .....	67
2.9.2 Bandpassed wavelength ratios with Gen_o3_tab ..	67
2.10 Genwctab .....	68
2.10.1 Simple wavelength ratios with Gen_wc_tab .....	68
2.10.2 Bandpassed wavelength ratios with Gen_wc_tab .....	69

<b>3</b>	<b>C functions in libRadtran</b>	<b>71</b>
3.1	ASCII file access	71
3.1.1	Usage of the ASCII library	71
3.1.2	General comments to the ASCII library	71
3.1.3	ASCII Library functions	72
3.1.3.1	Function ASCII_checkfile	72
3.1.3.2	Function ASCII_calloc_string	73
3.1.3.3	Function ASCII_free_string	74
3.1.3.4	Function ASCII_calloc_int	74
3.1.3.5	Function ASCII_calloc_double	75
3.1.3.6	Function ASCII_calloc_double_3D	75
3.1.3.7	Function ASCII_calloc_double_3D_arylen	75
3.1.3.8	Function ASCII_calloc_double_4D_arylen	76
3.1.3.9	Function ASCII_calloc_double_3D_arylen_restricted	76
3.1.3.10	Function ASCII_calloc_float_3D	77
3.1.3.11	Function ASCII_calloc_float_4D	77
3.1.3.12	Function ASCII_calloc_double_4D	77
3.1.3.13	Function ASCII_calloc_float_5D	78
3.1.3.14	Function ASCII_calloc_float	78
3.1.3.15	Function ASCII_calloc_short	79
3.1.3.16	Function ASCII_free_int	79
3.1.3.17	Function ASCII_free_double	79
3.1.3.18	Function ASCII_free_float	80
3.1.3.19	Function ASCII_free_short	80
3.1.3.20	Function ASCII_free_double_3D	81
3.1.3.21	Function ASCII_free_float_3D	81
3.1.3.22	Function ASCII_free_float_4D	82
3.1.3.23	Function ASCII_free_double_4D	82
3.1.3.24	Function ASCII_free_float_5D	82
3.1.3.25	Function ASCII_free_float_6D	83
3.1.3.26	Function ASCII_free_float_7D	83
3.1.3.27	Function ASCII_readfile	84
3.1.3.28	Function ASCII_string2double	84
3.1.3.29	Function ASCII_string2float	84
3.1.3.30	Function ASCII_file2string	85
3.1.3.31	Function ASCII_file2double	86
3.1.3.32	Function ASCII_file2float	86
3.1.3.33	Function ASCII_column	87
3.1.3.34	Function ASCII_column_float	88
3.1.3.35	Function ASCII_row	88
3.1.3.36	Function read_1c_file	88
3.1.3.37	Function read_1c_file_float	89
3.1.3.38	Function read_2c_file	89
3.1.3.39	Function read_2c_file_float	90
3.1.3.40	Function read_3c_file	90

3.1.3.41	Function read_3c_file_float .....	90
3.1.3.42	Function read_4c_file .....	91
3.1.3.43	Function read_4c_file_float .....	91
3.1.3.44	Function read_5c_file .....	92
3.1.3.45	Function read_5c_file_float .....	92
3.1.3.46	Function read_6c_file .....	93
3.1.3.47	Function read_7c_file .....	93
3.1.3.48	Function read_8c_file .....	94
3.1.3.49	Function read_9c_file .....	94
3.1.3.50	Function substr .....	95
3.1.3.51	Function ASCII_parse .....	95
3.1.3.52	Function ASCII_parsestring .....	95
3.2	Numeric functions .....	96
3.2.1	Usage of the Numeric Library .....	96
3.2.2	General comments to the Numeric Library .....	96
3.2.3	Numeric Library functions .....	97
3.2.3.1	Function convolute .....	97
3.2.3.2	Function int_convolute .....	98
3.2.3.3	Function solve_gauss .....	99
3.2.3.4	Function solve_five .....	99
3.2.3.5	Function solve_five_ms .....	100
3.2.3.6	Function solve_three .....	100
3.2.3.7	Function solve_three_ms .....	101
3.2.3.8	Function fsolve_three_ms .....	102
3.2.3.9	Function double_equal .....	102
3.2.3.10	Function sort_long .....	103
3.2.3.11	Function sort_double .....	103
3.2.3.12	Function fak .....	104
3.2.3.13	Function integrate .....	104
3.2.3.14	Function integrate_float .....	105
3.2.3.15	Function integrate_spline .....	105
3.2.3.16	Function linear_coeffc .....	106
3.2.3.17	Function linear_coeffc_fast .....	107
3.2.3.18	Function slinear_coeffc .....	107
3.2.3.19	Function calc_splined_value .....	108
3.2.3.20	Function gauss .....	109
3.2.3.21	Function regression .....	109
3.2.3.22	Function weight_regression .....	110
3.2.3.23	Function spline .....	111
3.2.3.24	Function spline_coeffc .....	112
3.2.3.25	Function fspline_coeffc .....	112
3.2.3.26	Function appspl .....	113
3.2.3.27	Function appspl_coeffc .....	114
3.2.3.28	Function calc_splined_value .....	115
3.2.3.29	Function fcalc_splined_value .....	116
3.2.3.30	Function linear_eqd .....	117
3.3	Fortran/C array functions .....	117
3.3.1	Usage of the Fortran/C array functions .....	117

3.3.2	General comments to the Fortran/C functions ..	118
3.3.3	Fortran/C Library functions .....	118
3.3.3.1	Function c2fortran_2D_float_ary .....	118
3.3.3.2	Function fortran2c_2D_float_ary .....	119
3.3.3.3	Function fortran2c_2D_float_ary_noalloc .....	119
3.3.3.4	Function fortran2c_2D_double_ary_noalloc .....	120
3.3.3.5	Function fortran2c_2D_double_ary .....	120
3.3.3.6	Function fortran2c_3D_float_ary_noalloc .....	121
3.3.3.7	Function fortran2c_3D_float_ary .....	121
3.3.3.8	Function fortran2c_4D_float_ary .....	122
3.3.3.9	Function fortran2c_4D_double_ary_noalloc .....	122
3.3.3.10	Function fortran2c_4D_double_ary .....	123
3.4	Sun-Earth Astronomical Relationship .....	123
3.4.1	Usage of the Sun library .....	123
3.4.2	General comments to the Sun Library .....	124
3.4.3	Sun Library functions .....	124
3.4.3.1	Function eccentricity .....	124
3.4.3.2	Function declination .....	124
3.4.3.3	Function equation_of_time .....	125
3.4.3.4	Function LAT .....	125
3.4.3.5	Function solar_zenith .....	126
3.4.3.6	Function solar_azimuth .....	126
3.4.3.7	Function day_of_year .....	127
3.4.3.8	Function zenith2time .....	127
3.4.3.9	Function Gregorian2Julian .....	128
3.4.3.10	Function Julian2Gregorian .....	129
3.4.3.11	Function location .....	129
3.5	Mie calculations .....	130
3.5.1	Usage of the Mie library .....	130
3.5.2	General comments to the Mie Library .....	130
3.5.3	Mie Library functions .....	130
3.5.3.1	Function mie_calc .....	130
3.5.3.2	Function mie_calc_sizedist .....	131
3.5.3.3	Function phase_function .....	133
3.5.3.4	Function read_mie_table .....	133
3.5.3.5	Function read_mie_table_lambda .....	135
3.6	Miscellaneous .....	135
3.6.1	Usage of the Miscellaneous library .....	135
3.6.2	Miscellaneous functions .....	135
3.6.2.1	Function air_refraction .....	135
3.6.2.2	Function vac2air .....	136
3.6.2.3	Function snowalbedo .....	137

<b>4</b>	<b>Fortran library functions . . . . .</b>	<b>139</b>
4.1	Fortran functions in libRadtran . . . . .	139
4.1.1	Function wcloud . . . . .	139
	<b>References . . . . .</b>	<b>141</b>

