

Including user-defined atmosphere grids in Cloudy

Peter van Hoof

Royal Observatory of Belgium, Ringlaan 3, 1180 Brussels, Belgium

March 12, 2007

1 Introduction

Cloudy now features the possibility of including user-defined stellar atmosphere grids with an arbitrary number of dimensions (this is the number of parameters that are varied in the grid). Using such a grid requires two steps. The first is to create an ascii file with all the necessary information. The format of this file will be described in Sect. 2. The second step is to compile this file into a binary atmosphere file, just like any other atmosphere grid used by Cloudy. This is described in Sect. 3. The runtime behavior of the code is described in Sect. 4.

2 The ascii file

Each stellar atmosphere grid needs to be defined in a single ascii file containing all the necessary information. The format is illustrated in Table 1. The aim of this format is to keep the data in the ascii file as close as possible to the original calculations. The file starts with some general information, followed by a block of model parameters, the frequency/wavelength grid, and finally each of the spectral energy distributions (SEDs). No comments of any sort are allowed in the file. It is implicitly assumed that each SED has been rebinned onto the same frequency/wavelength grid. This grid does not need to coincide with the Cloudy grid. The grid will be automatically rebinned onto the Cloudy grid during the compilation stage described in Sect. 3. The supplied frequency/wavelength grid doesn't need to cover the entire Cloudy grid. Extrapolation will be used to fill in the missing data. It is allowed for flux points in the Wien tail to have zero flux. It is however the users responsibility to assure that sufficient data are supplied to enable a safe extrapolation of the spectra to the entire Cloudy grid.

I will now describe each of the items in the ascii file in more detail. The first line contains the magic number that identifies the syntax version. This number is the same in all ascii files that are used by Cloudy. It is checked when Cloudy compiles the ascii file. This document describes version 20060612. The second line contains the number of parameters $ndim$ that are varied in the grid. Typically this number is 2, but it may be any number ≥ 1 and $\leq MDIM$ (the latter parameter is defined in `stars.h`). The default value for `MDIM` is 4, which should be sufficient for all current grids. The parameter can however be increased if the user wishes so. The code, as well as the binary atmosphere files, will then have to be recompiled. The third lines contains the number of parameters $npar$ that are supplied for each atmosphere model. This number may be larger than $ndim$, but not smaller. The additional parameters are printed in the output, so using $npar > ndim$ can be a useful way of supplying additional information about the models. The number of parameters is limited by the parameter `MDIM` defined in `stars.h`. Next follow $npar$ lines, each giving a label for that parameter. This is used in the Cloudy output. If $ndim = 1$ then the first label may be `Teff` or `age`, in all other cases it must be `Teff`. If $ndim \geq 2$ then the second label must be `log(g)`. The third and subsequent labels are completely free. The next line gives the number of atmosphere models $nmod$ in the grid, followed on the next line by the number of frequency/wavelength points $nfreq$ in each of those models. In order to keep the format of the ascii file as close as possible to the original models, it is allowed to use either a frequency or wavelength grid in arbitrary

units. The same holds for the dependent variable, which may either be a flux per frequency or wavelength unit. Both the dependent and independent variable should be entered as linear quantities, so logarithmic units like magnitudes are not supported. With this amount of freedom, the code needs to know how to convert these variables to Cloudy’s internal units. This is defines on the next 4 lines. The next line should contain either `nu` to state that the file contains a frequency grid, or `lambda` to state that it contains a wavelength grid. The next line contains a conversion factor to convert the numbers in the ascii file to a frequency in Hz (case `nu`) or a wavelength in Å (case `lambda`). The conversion factor is multiplied with the numbers in the ascii file. So, if the ascii file would contain a wavelength grid in nanometer, then the conversion factor would have to be 10 to convert it into a grid in Å. The next line defines the data type of the dependent variable. Legal values are `F_nu`, `H_nu` for fluxes per frequency unit, or `F_lambda`, `H_lambda` for fluxes per wavelength unit. The next line contains the conversion factor to convert the numbers in the ascii file to a flux in $\text{erg cm}^{-2} \text{s}^{-1} \text{Hz}^{-1}$ (case `F_nu`, `H_nu`) or $\text{erg cm}^{-2} \text{s}^{-1} \text{Å}^{-1}$ (case `F_lambda`, `H_lambda`). Again the conversion factors are multiplied with the numbers in the ascii file. In the cases `H_nu` and `H_lambda` no implicit factor 4π will be used! So if the ascii file contains Eddington fluxes `H_nu` in $\text{erg cm}^{-2} \text{s}^{-1} \text{Å}^{-1} \text{sr}^{-1}$, then the conversion factor should be 12.56637... All fluxes are eventually renormalized in Cloudy, but nevertheless it is important to get this conversion correct. The effective temperature of interpolated models is internally checked by evaluating the integral:

$$\int_0^{\infty} F_{\nu} d\nu = \sigma T_{\text{eff}}^4.$$

If the result deviates too much from the expected value an error will be printed. When compiling the grid, each model in the grid will be tested in the same manner. A warning will be printed for each model that deviates from the expected value. If all models generate such a warning, you should check your normalization.

This concludes the general setup of the grid. The remainder of the file contains the actual parameters and the data of the atmosphere models. These numbers can be formatted in any way you like, as long as it complies with C formatting rules. More in particular, you can supply more than 1 number per line, as long as they are separated by whitespace. Several forms of Fortran number formatting are not recognized in C! First, using “D-format” numbers (e.g. 1.0d+20) is not allowed as C does not recognize this notation. Second, Fortran may omit the “e” in numbers with large exponents (e.g. 1.0-102). This notation is not recognized either by C. The “e” in the exponential notation may be either lowercase or uppercase. In the ascii file, first the model parameters must be supplied. This means that `nmod` × `npar` numbers must be entered, first all parameters for model #1, then the parameters for model #2, etc. Next follows a block of `nfreq` numbers defining the frequency/wavelength grid. Finally, `nmod` blocks, each containing `nfreq` numbers, must be entered which give the fluxes for each of the atmosphere models. This then concludes the ascii file.

3 Compiling the grid

Ascii files containing the stellar atmosphere grid can have arbitrary names as long as the filename extension is `.ascii`. In the remainder we will assume the file has been named `usergrid.ascii`. In order for the grid to be used by Cloudy, it must be compiled into a binary format. This format is platform dependent, so the binary file needs to be recreated for every machine architecture that you wish to use it on. In order to compile the file, go to the directory that contains the ascii file (this does not need to be the Cloudy data directory), start up Cloudy and type the following command:

```
compile stars "usergrid.ascii"
```

followed by two carriage returns. Cloudy will now create a file `usergrid.mod`. If you did the above outside the normal Cloudy directory, you now need to move the `usergrid.mod` file to the Cloudy data directory. You can now use the grid by including the following in your input file:

```
table star "usergrid.mod" 24,000 4.3
```

Table 1: Example of an ascii file, derived from the kwerner.ascii file. **Comments are not allowed in the file and are only shown here for clarity.**

```

20060612      # magic number
2            # ndim
2            # npar
Teff         # label par 1
log(g)       # label par 2
20           # nmod
513         # nfreq
nu           # type of independent variable (nu or lambda)
3.28984196e+15 # conversion factor for independent variable
F_nu        # type of dependent variable (F_nu/H_nu or F_lambda/H_lambda)
1.00000000e+00 # conversion factor for dependent variable
# nmod sets of npar model parameters
 80000. 5.0 80000. 6.0 80000. 7.0 80000. 8.0
100000. 5.0 100000. 6.0 100000. 7.0 100000. 8.0
120000. 6.0 120000. 7.0 120000. 8.0 140000. 6.0
140000. 7.0 140000. 8.0 160000. 7.0 160000. 8.0
180000. 7.0 180000. 8.0 200000. 7.0 200000. 8.0
# the frequency/wavelength grid, nfreq points
1.0100000e-05 3.0396596e-04 6.0762797e-04 9.1129000e-04 9.8728144e-04
... <505 more frequency points>
1.7655824e+02 1.7940598e+02 1.8225368e+02
# the SED for model 1, nfreq flux points
6.2701018e-11 5.5260639e-08 2.1959737e-07 4.9197354e-07 5.7711065e-07
... <505 more flux points>
9.0885815e-37 2.7039771e-37 7.9828291e-38
... <the SED's for models 2 ... 19>
# the SED for model 20, nfreq flux points
1.4960523e-10 1.3383193e-07 5.3344235e-07 1.1975753e-06 1.4049202e-06
... <505 more flux points>
2.1177483e-13 1.3848120e-13 9.0415034e-14

```

If the grid has two dimensions (Teff and $\log(g)$), you can supply only Teff and $\log(g)$ will default to the highest value in the grid. In all other cases you need to supply exactly `ndim` parameters. See Hazy for further details.

4 Runtime behavior

When Cloudy encounters the following command in its input:

```
table star "usergrid.mod" 24,300 4.3
```

it will try to interpolate the requested model using as few models as possible. This means that if an exact match could be found in the grid, only that model will be used. If an exact match could be found for one parameter, but not another, then only interpolation in the latter parameter will be performed. In general no extrapolation will be performed. The only exception is the $\log(g)$ parameter for which more relaxed rules are used. More on that later. First we will return to the example above. Let's assume that the usergrid contains models for $T_{\text{eff}} = \{ \dots 24,000, 25,000 \dots \}$ and $\log(g) = \{ \dots 4.0, 4.5 \dots \}$. Then Cloudy will perform interpolation using the following 4 models: $(T_{\text{eff}}, \log(g)) = (24,000, 4.0)$, $(24,000, 4.5)$, $(25,000, 4.0)$, and $(25,000, 4.5)$. It will perform linear interpolation on those models using the log of the flux. If any of the models it needs for the interpolation is not present in the grid, the interpolation will fail and Cloudy will stop. However, more relaxed rules exist for $\log(g)$. If it cannot find a model with a given $\log(g)$, it will substitute the model with the nearest $\log(g)$ instead. The reasoning behind this is that in general the spectral energy distribution is not very sensitive to the value of $\log(g)$.