MSG

Rich Townsend & The MSG Team

Jun 06, 2022

USER GUIDE

Multidimensional Spectral Grids (MSG) is an open-source software package that synthesizes stellar spectra and photometric colors via interpolation in pre-calculated grids. MSG is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

CHAPTER

ONE

PRELIMINARIES

1.1 Why use MSG?

Synthesizing stellar spectra from first principles is a complicated endeavor, requiring a detailed understanding of radiative transfer and atomic physics, together with significant computational resources. Therefore, in most circumstances its better to use one of the many pre-calculated grids of spectra published in the astrophysical literature (see, e.g., Lanz & Hubeny, 2003; Lanz & Hubeny, 2007; Kirby, 2011; de Laverny et al., 2012; Husser et al., 2013; Allende Prieto et al., 2018; Chiavassa et al., 2018; Zsargó et al., 2020). However, even with these grids a significant obstacle remains: when atmospheric parameters fall between the grid nodes, some kind of interpolation is necessary in order to evaluate a spectrum.

MSG is designed to solve this problem. It's not the first software package that offers stellar spectral interpolation (see, e.g., Blanco-Cuaresma et al., 2014; Allende-Prieto & Apogee Team, 2015); however, with spectral interpolation as its *sole* focus, it offers a combination of features unmatched by other packages:

- scalability — MSG handles grids that are much larger (on disk) than available computer memory.
- extensibility — MSG handles grids with an arbitrary number of dimensions.
- portability — MSG is platform-agnostic and provide APIs for the programming languages (Fortran, C, Python) most commonly used in Astronomy.
- performance — MSG provides smooth and accurate interpolates with minimal computational cost.
- robustness — MSG gracefully handles missing data caused by holes and/or ragged boundaries in the grid.

Together, these features mean that MSG is flexible and powerful while remaining straightforward to use: it's the perfect condiment to add flavor to your science!.

1.2 Obtaining MSG

The source code for MSG is hosted in the rhdtownsend/msg git repository on GitHub. Instructions for downloading and installing the software can be found in the *Quick Start* chapter.

1.3 Citing MSG

If you use MSG in your research, please cite the following papers:

- Townsend R. H. D., Lopez A., 2022, *Journal of Open-Source Software*, in preparation

Be sure to also cite the source of the grid data that you're using with MSG. For instance, if you're working with one of the *CAP18 grids*, you should cite Allende Prieto et al. (2018).

1.4 Development Team

MSG remains under active development by the following team:

- Rich Townsend (University of Wisconsin-Madison); project leader
- Aaron Lopez (University of Wisconsin-Madison)

1.5 Related Links

- The MESA Software Development Kit (SDK), which provides the compilers and supporting libraries needed to build MSG.

1.6 Acknowledgments

MSG has been developed with financial support from the following grants:

- NSF awards ACI-1663696 and AST-1716436;
- NASA award 80NSSC20K0515.

CHAPTER

TWO

QUICK START

To get started with MSG, follow these five simple steps:

- install the MESA Software Development Kit;
- download the MSG source code;
- unpack the source code using the **`tar`** utility;
- set the `MSG_DIR` environment variable to point to the newly created source directory;
- compile MSG using the command **`make -C $MSG_DIR`**.

For a more in-depth installation guide that covers alternative use-cases, refer to the *Installation* chapter. If the code doesn't compile properly, consult the *Troubleshooting* chapter. Otherwise, proceed to the next chapter where you'll put together your first MSG calculation.

CHAPTER

THREE

PYTHON WALKTHROUGH

This chapter walks through using the MSG Python interface to evaluate spectra and photometric colors for a model of Sirius A (α Canis Majoris A). The code fragments are presented as a sequence of Jupyter notebook cells, but pasting all of the fragments into a Python script should work also.

3.1 Preparation

Before starting Jupyter, ensure that recent versions of the NumPy, Matplotlib and Astropy packages are installed on your system. Also, make sure that the `MSG_DIR` environment variable is set as described in the *Quick Start* chapter.

3.2 Importing the PyMSG Module

Launch Jupyter, create a new Python3 notebook, and initialize the notebook with the following statements:

[1]:
```
# Import standard modules

import sys
import os
import numpy as np
import astropy.constants as con
import astropy.units as unt
import matplotlib.pyplot as plt

# Set paths & import pymsg

MSG_DIR = os.environ['MSG_DIR']

sys.path.insert(0, os.path.join(MSG_DIR, 'python'))
import pymsg

# Set plot parameters

%matplotlib inline
plt.rcParams.update({'font.size': 16})
```

The *sys.path.insert* statement adds the directory `$MSG_DIR/python` to the search path, allowing Python to find and import the `pymsg` module. This module provides the Python interface to MSG.

3.3 Loading & Inspecting the Demo Grid

For demonstration purposes MSG includes a low-resolution *specgrid* (spectroscopic grid) file, `$MSG_DIR/data/grids/sg-demo.h5`[1]. Load this demo grid into memory[2] by creating a new *pymsg.SpecGrid* object:

[2]:
```
# Load the SpecGrid

GRID_DIR = os.path.join(MSG_DIR, 'data', 'grids')

specgrid_file_name = os.path.join(GRID_DIR, 'sg-demo.h5')

specgrid = pymsg.SpecGrid(specgrid_file_name)
```

This object has a number of (read-only) properties that inform us about the parameter space spanned by the grid:

[3]:
```
# Inspect grid parameters

print('Grid parameters:')

for label in specgrid.axis_labels:
    print(f'  {label} [{specgrid.axis_x_min[label]} -> {specgrid.axis_x_max[label]}]')

print(f'  lam [{specgrid.lam_min} -> {specgrid.lam_max}]')
```

```
Grid parameters:
  Teff [3500.0 -> 50000.0]
  log(g) [0.0 -> 5.0]
  lam [2999.9999999999977 -> 9003.490078514782]
```

Here, `Teff` and `log(g)` correspond (respectively) to the $/$ and $\log_{10}(g/^{-2})$ atmosphere parameters, while `lam` is wavelength $\lambda/$.

3.4 Plotting the Flux

With the grid loaded, let's evaluate and plot a flux spectrum for Sirius A. First, store the atmospheric parameters for the star in a dict:

[4]:
```
# Set atmospheric parameters dict from Sirius A's fundamental parameters

M = 2.02 *con.M_sun
R = 1.711 * con.R_sun
L = 25.4 * con.L_sun

Teff = (L/(4*np.pi*R**2*con.sigma_sb))**0.25/unt.K
logg = np.log10(con.G*M/R**2/(unt.cm/unt.s**2))

x = {'Teff': Teff, 'log(g)': logg}

print(Teff,np.log10(Teff))
```

[1] Larger grids can be downloaded separately from MSG; see the *Grid Files* appendix.
[2] Behind the scenes, the *specgrid* data is loaded on demand; see the *Memory Management* section for further details.

```
9906.266444160501 3.9959100048166833
```

(the mass, radius and luminosity data are taken from Liebert et al., 2005). Then set up a uniformly-spaced wavelength abcissa for a spectrum spanning the visible range, 3 000 to 7 000 .

[5]:
```
# Set up the wavelength abscissa

lam_min = 3000.
lam_max = 7000.

lam = np.linspace(lam_min, lam_max, 501)

lam_c = 0.5*(lam[1:] + lam[:-1])
```

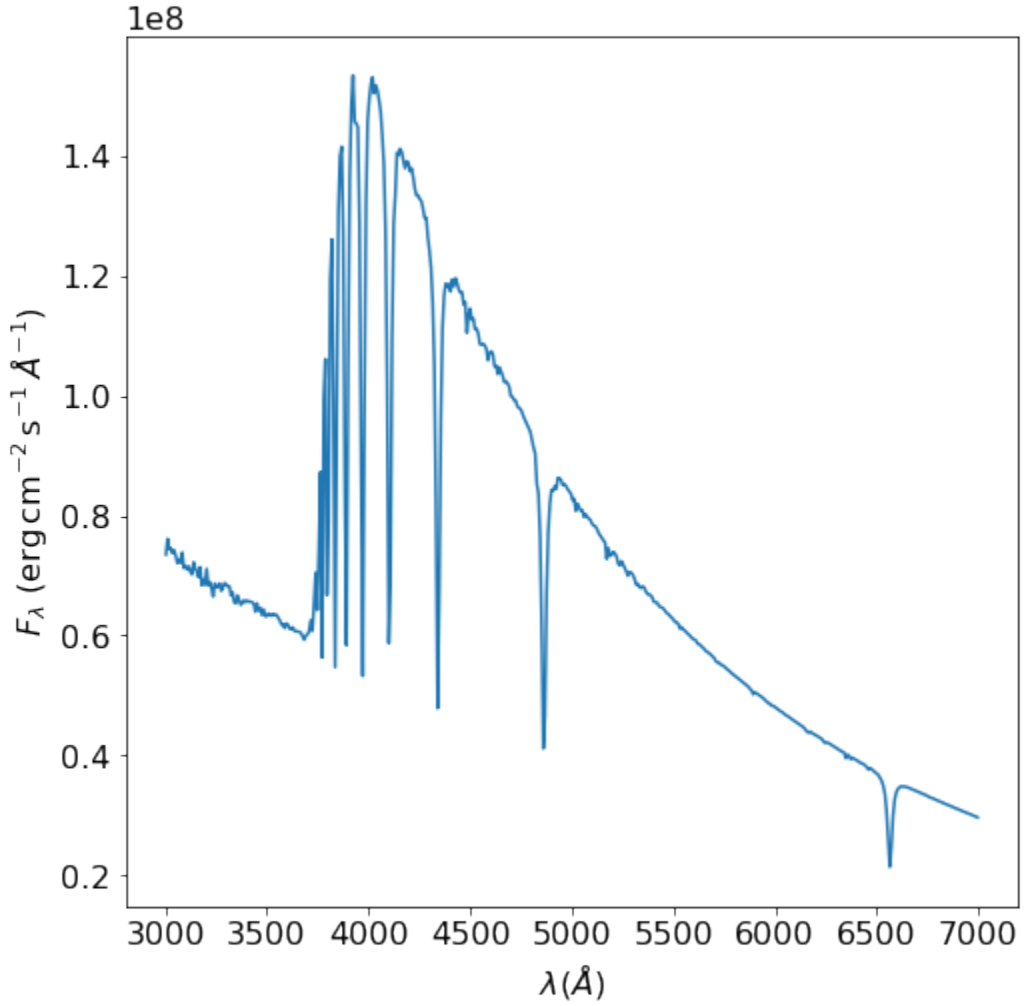
Here, the array `lam` defines the boundaries of 500 wavelength bins $\{\lambda_i, \lambda_{i+1}\}$ $(i = 1, \ldots, 500)$ and the array `lam_c` stores the central wavelength of each bin.

With all our parameters defined, evaluate the flux spectrum using a call to the *pymsg.SpecGrid.flux()* method, and then plot it:

[6]:
```
# Evaluate the flux

F_lam = specgrid.flux(x, lam)

# Plot

plt.figure(figsize=[8,8])
plt.plot(lam_c, F_lam)

plt.xlabel(r'$\lambda ({\AA})$')
plt.ylabel(r'$F_{\lambda}\ ({\rm erg\,cm^{-2}\,s^{-1}}\,\AA^{-1})$')
```

```
x_vec, deriv_vec [9.90626644e+03 4.27691898e+00] [0 0]
```

[6]:
```
Text(0, 0.5, '$F_{\\lambda}\\ ({\\rm erg\\,cm^{-2}\\,s^{-1}}\\,\\AA^{-1})$')
```



This looks about right — we can clearly see the Balmer series, starting with Hα at 6563 .

3.5 Plotting the Intensity

Sometimes we want to know the specific intensity of the radiation emerging from a star's atmosphere; an example might be when we're modeling eclipse or transit phenomena, which requires detailed knowlege of the stellar-surface radiation field. For this, we can use the *pymsg.SpecGrid.intensity()* method.

Here's a demonstration of this method in action, plotting the specific intensity across the Hα line profile for ten different values of the emergence direction cosine $\mu = 0.1, 0.2, \ldots, 1.0$.

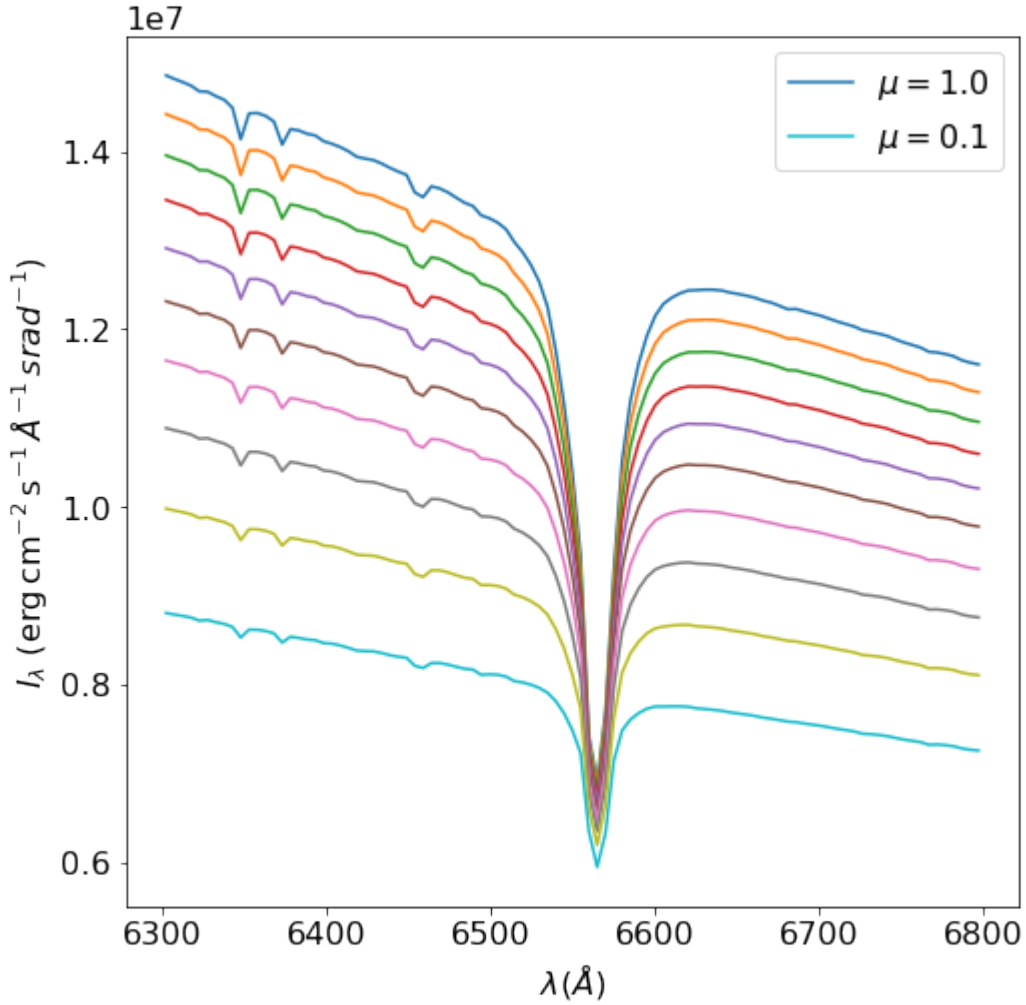
[7]:
```
# Set up the wavelength abscissa

lam_min = 6300.
lam_max = 6800.

lam = np.linspace(lam_min, lam_max, 100)
```

(continues on next page)

(continued from previous page)

```
lam_c = 0.5*(lam[1:] + lam[:-1])

# Loop over mu

plt.figure(figsize=[8,8])

for mu in np.linspace(1.0, 0.1, 10):

    # Evaluate the intensity

    I_lam = specgrid.intensity(x, mu, lam)

    # Plot

    if mu==0.1 or mu==1.0:
        label=r'$\mu={:3.1f}$'.format(mu)
    else:
        label=None

    plt.plot(lam_c, I_lam, label=label)

plt.xlabel(r'$\lambda ({\AA})$')
plt.ylabel(r'$I_{\lambda}\ ({\rm erg\,cm^{-2}\,s^{-1}}\,\AA^{-1}\,srad^{-1})$')

plt.legend()
```

[7]:
```
<matplotlib.legend.Legend at 0x7fdff01a4d00>
```



We can clearly see that limb-darkening in the line core is much weaker than in the continuum — exactly what we expect from such a strong line.

3.6 Evaluating Magnitudes & Colors

As a final step in this walkthrough, let's evaluate the magnitude and colors of Sirius A in the Johnson photometric system. We can do this by creating a new *pymsg.PhotGrid* object for each passband:

[8]:
```
# Load the PhotGrids

PASS_DIR = os.path.join(MSG_DIR, 'data', 'passbands')
filters = ['U', 'B', 'V']

photgrids = {}

for filter in filters:
    passband_file_name = os.path.join(PASS_DIR, f'pb-Generic-Johnson.{filter}-Vega.h5')
    photgrids[filter] = pymsg.PhotGrid(specgrid_file_name, passband_file_name)
```

(for convenience, we store the *pymsg.PhotGrid* objects in a dict, keyed by filter name). In the calls to the object constructor *pymsg.PhotGrid()*, the first argument is the name of a *specgrid* file (i.e., the demo grid), and the second argument is the name of a passband definition file; a limited set of these files is provided in the `$MSG_DIR/data/passbands` subdirectory[3]. The normalized *surface* fluxes of Sirius A are then be found using the *pymsg.PhotGrid.flux()* method:

[9]:
```
# Evaluate the surface fluxes (each normalized to the passband
# zero-point flux)

F_surf = {}

for filter in filters:
    F_surf[filter] = photgrids[filter].flux(x)
```

To convert these into apparent magnitudes, we first dilute them to Earth fluxes using the inverse-square law with a parallax from Perryman et al. (1997), and then apply Pogson's logarithmic formula:

[10]:
```
# Set the distance to Sirius A

p = 0.379
d = 1/p * con.pc

# Evaluate the Earth fluxes

F = {}

for filter in filters:
    F[filter] = F_surf[filter]*R**2/d**2

# Evaluate apparent magnitudes and print out magnitude & colors

mags = {}

for filter in filters:
    mags[filter] = -2.5*np.log10(F[filter])

print(f"V={mags['V']}, U-B={mags['U']-mags['B']}, B-V={mags['B']-mags['V']}")
```

```
V=-1.4472632983389206, U-B=-0.05922507470947869, B-V=0.0012697568194013353
```

Reassuringly, the resulting values are within 10 mmag of Sirius A's apparent V-band magnitude and colors (as given by the star's Simbad entry).

[3] Passband definition files for other instruments/photometric systems can be downloaded separately from MSG; see the *Passband Files* appendix.

CHAPTER

FOUR

FORTRAN WALKTHROUGH

This chapter reprises the steps of the *Python Walkthrough* — evaluating spectra and photometric colors for Sirius A — but now using the MSG Fortran interface.

4.1 Preparation

In your working directory, create a new file `fortran-walkthrough.f90` with the following source code:

```
program fortran_walkthrough

   ! Uses

   use forum_m
   use fmsg_m

   ! No implicit typing

   implicit none

   ! Parameters

   real(RD), parameter :: lam_min = 3000._RD
   real(RD), parameter :: lam_max = 7000._RD
   integer, parameter  :: n_lam = 501

   ! Variables

   type(specgrid_t)     :: specgrid
   type(axis_t)         :: axis
   character(LABEL_LEN) :: label
   integer              :: i
   real(RD)             :: lam(n_lam)
   real(RD)             :: lam_c(n_lam-1)
   real(RD)             :: x_vec(2)
   real(RD)             :: F_lam(n_lam-1)
   integer              :: unit
   type(photgrid_t)     :: photgrid_U
   type(photgrid_t)     :: photgrid_B
   type(photgrid_t)     :: photgrid_V
   real(RD)             :: F_U
```

(continues on next page)

(continued from previous page)

```
   real(RD)              :: F_B
   real(RD)              :: F_V
   real(RD)              :: R2_d2
   real(RD)              :: U
   real(RD)              :: B
   real(RD)              :: V

   ! Load the specgrid

   call load_specgrid('sg-demo.h5', specgrid)

   ! Set atmospheric parameters to correspond to Sirius A

   do i = 1, 2

      call specgrid%get_axis(i, axis)
      call axis%get_label(label)

      select case(label)
      case('log(g)')
         x_vec(i) = 4.277_RD
      case('Teff')
         x_vec(i) = 9906._RD
      case default
         stop 'unrecognized axis label'
      end select

   end do

   ! Set up the wavelength abscissa

   lam = [((lam_min*(n_lam-i) + lam_max*(i-1))/(n_lam-1), i=1,n_lam)]

   lam_c = 0.5_RD*(lam(:n_lam-1) + lam(2:))

   ! Evaluate the flux

   call specgrid%interp_flux(x_vec, lam, F_lam)

   ! Write it to a file

   open(NEWUNIT=unit, FILE='spectrum.dat', STATUS='REPLACE')

   do i = 1, n_lam-1
      write(unit, *) lam_c(i), F_lam(i)
   end do

   close(unit)

   ! Load the photgrids

   call load_photgrid_from_specgrid('sg-demo.h5', 'pb-Generic-Johnson.U-Vega.h5',␣
↪photgrid_U)
```

(continues on next page)

(continued from previous page)

```
   call load_photgrid_from_specgrid('sg-demo.h5', 'pb-Generic-Johnson.B-Vega.h5',␣
↪photgrid_B)
   call load_photgrid_from_specgrid('sg-demo.h5', 'pb-Generic-Johnson.V-Vega.h5',␣
↪photgrid_V)

   ! Evaluate fluxes

   call photgrid_U%interp_flux(x_vec, F_U)
   call photgrid_B%interp_flux(x_vec, F_B)
   call photgrid_V%interp_flux(x_vec, F_V)

   ! Evaluate apparent magnitudes (the droid factor R2_d2 is
   ! R**2/d**2, where R is Sirius A's radius and d its distance)

   R2_d2 = 2.1366E-16_RD

   U = -2.5_RD*LOG10(F_U*R2_d2)
   B = -2.5_RD*LOG10(F_B*R2_d2)
   V = -2.5_RD*LOG10(F_V*R2_d2)

   print *, '  V=', V
   print *, 'U-B=', U-B
   print *, 'B-V=', B-V

   ! Finish

end program fortran_walkthrough
```

A few brief comments on the code:

- The `use forum_m` statement provides access to the Fortran Utility Module (ForUM). For the purposes of the demo program, this module defines the *RD* kind type parameter for double precision real variables.
- The `use fmsg_m` statement provides access to the MSG Fortran interface. Primarily, this interface serves to define the *specgrid_t* and *photgrid_t* datatypes.
- Because Fortran doesn't have `dict` datatypes, the atmosphere parameters must be passed to MSG as a plain array (here, stored in the variable `x_vec`). A `select case` construct is used to make sure the correct values are stored in each array element.

4.2 Compiling

The next step is to compile the demo program. Make sure the `MSG_DIR` environment variable is set, as described in the *Quick Start* chapter. Then, enter the following on the command line:

```
$ gfortran -o fortran-walkthrough fortran-walkthrough.f90 -I$MSG_DIR/include `$MSG_DIR/scripts/fmsg_lin
```

The `-I$MSG_DIR/include` option tells the compiler where to find the module definition (`.mod`) files, while the `$MSG_DIR/scripts/fmsg_link` clause (note the enclosing backticks) runs a link script that returns the compiler flags necessary to link the program against the appropriate libraries.

4.3 Running

To run the code, first create a symbolic link to the demo grid:

```
$ ln -s $MSG_DIR/data/grids/sg-demo.h5
```

Then, execute the command

```
$ ./fortran-walkthrough
```

The code will create a file `spectrum.dat` containing the flux spectrum for Sirius A (as an ASCII table), and print out the apparent V-band magnitude and colors of the star.

CHAPTER

FIVE

C WALKTHROUGH

This chapter reprises the steps of the *Python Walkthrough* — evaluating spectra and photometric colors for Sirius A — but now using the MSG C interface.

5.1 Preparation

In your working directory, create a new file `c-walkthrough.c` with the following source code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include "cmsg.h"

#define N_LAM 501
#define LAM_MIN 3000.
#define LAM_MAX 7000.

int main(int argc, char *argv[]) {

  SpecGrid specgrid;
  PhotGrid photgrid_U;
  PhotGrid photgrid_B;
  PhotGrid photgrid_V;

  char label[17];

  double x_vec[2];

  double lam[N_LAM];
  double lam_c[N_LAM-1];
  double F[N_LAM-1];
  double F_U;
  double F_B;
  double F_V;
  double R2_d2;
  double U;
  double B;
  double V;
```

(continues on next page)

(continued from previous page)

```
FILE *fptr;

// Load the specgrid

load_specgrid("sg-demo.h5", &specgrid, NULL);

// Set atmospheric parameters to correspond to Sirius A

for(int i=0; i < 2; i++) {

  get_specgrid_axis_label(specgrid, i, label);

  if (strcmp(label, "log(g)") == 0) {
    x_vec[i] = 4.277;
  }
  else if (strcmp(label, "Teff") == 0) {
    x_vec[i] = 9906.;
  }
  else {
    printf("invalid label\n");
    exit(1);
  }

}

// Set up the wavelength abscissa

for(int i=0; i < N_LAM; i++) {
  lam[i] = (LAM_MIN*(N_LAM-1-i) + LAM_MAX*i)/(N_LAM-1);
}

for(int i=0; i < N_LAM-1; i++) {
  lam_c[i] = 0.5*(lam[i] + lam[i+1]);
}

// Evaluate the flux

interp_specgrid_flux(specgrid, x_vec, N_LAM, lam, F, NULL, NULL);

// Write it to a file

fptr = fopen("spectrum.dat", "w");

for(int i=0; i < N_LAM-1; i++) {
  fprintf(fptr, "%.17e %.17e\n", lam_c[i], F[i]);
}

fclose(fptr);

// Load the photgrids
```

(continues on next page)

(continued from previous page)

```
  load_photgrid_from_specgrid("sg-demo.h5", "pb-Generic-Johnson.U-Vega.h5", &photgrid_U,␣
↪NULL);
  load_photgrid_from_specgrid("sg-demo.h5", "pb-Generic-Johnson.B-Vega.h5", &photgrid_B,␣
↪NULL);
  load_photgrid_from_specgrid("sg-demo.h5", "pb-Generic-Johnson.V-Vega.h5", &photgrid_V,␣
↪NULL);

  // Evaluate fluxes

  interp_photgrid_flux(photgrid_U, x_vec, &F_U, NULL, NULL);
  interp_photgrid_flux(photgrid_B, x_vec, &F_B, NULL, NULL);
  interp_photgrid_flux(photgrid_V, x_vec, &F_V, NULL, NULL);

  // Evaluate apparent magnitudes (the droid factor R2_d2 is
  // R^2/d^2, where R is Sirius A's radius and d its distance)

  R2_d2 = 2.1366E-16;

  U = -2.5*log10(F_U*R2_d2);
  B = -2.5*log10(F_B*R2_d2);
  V = -2.5*log10(F_V*R2_d2);

  printf("  V=  %.17e\n", V);
  printf("U-B=  %.17e\n", U-V);
  printf("B-V=  %.17e\n", B-V);

  // Clean up

  unload_specgrid(specgrid);

  unload_photgrid(photgrid_U);
  unload_photgrid(photgrid_B);
  unload_photgrid(photgrid_V);

  // Finish

  exit(0);

}
```

A few brief comments on the code:

- The `#include "cmsg.h"` statement includes the header definitions for the MSG C interface.
- Because Fortran doesn't have `dict` datatypes, the atmosphere parameters must be passed to MSG as a plain array (here, stored in the variable `x_vec`). A loop with `strcmp()` calls is used to make sure the correct values are stored in each array element.
- Many of the calls to MSG routines (e.g., *load_specgrid()*, *interp_specgrid_flux()*) contain `NULL` trailing arguments; these correspond to omitted optional arguments.

5.2 Compiling

The next step is to compile the demo program. Make sure the `MSG_DIR` environment variable is set, as described in the *Quick Start* chapter. Then, enter the following on the command line:

```
$ gcc -o c-walkthrough c-walkthrough.c -I$MSG_DIR/include `$MSG_DIR/scripts/cmsg_link`
```

The `-I$MSG_DIR/include` option tells the compiler where to find the header file, while the `$MSG_DIR/scripts/cmsg_link` clause (note the enclosing backticks) runs a link script that returns the compiler flags necessary to link the program against the appropriate libraries.

5.3 Running

To run the code, first create a symbolic link to the demo grid:

```
$ ln -s $MSG_DIR/data/grids/sg-demo.h5
```

Then, execute the command

```
$ ./c-walkthrough
```

The code will create a file `spectrum.dat` containing the flux spectrum for Sirius A (as an ASCII table), and print out the apparent V-band magnitude and colors of the star.

CHAPTER

SIX

HOW MSG WORKS

This chapter expands on the *Python*, *Fortran* and *C* walkthroughs, by describing in detail how MSG evaluates stellar spectra and photometric colors.

6.1 Evaluating a Spectrum

To evaluate a spectrum, MSG interpolates in a grid of pre-calculated spectroscopic data. Focusing on specific intensity evaluation, this involves constructing a (preferably continuous and smooth) function

$$I_\lambda(\mu; \lambda; x, y, z, \ldots)$$

representing the intensity (in units of $^{-2\ -1\ -1\ -1}$) emerging at direction cosine μ and wavelength λ from an atmosphere characterized by a set of N parameters $x, y, z, \ldots$ (which can correspond to quantities such as effective temperature $T_{\rm eff}$, gravity g, etc.).

6.1.1 Limb-Darkening Laws

The μ dependence of the specific intensity is represented using limb-darkening laws. The simplest and most well known is the linear law

$$\frac{I_\lambda(\mu; \ldots)}{I_\lambda(1; \ldots)} = 1 - a(\ldots)\,[1 - \mu] \tag{6.1}$$

where $I_\lambda(1; \ldots)$ represents the normally emergent ($\mu = 1$) intensity and $a(\ldots)$ is the linear limb-darkening coefficient (here the ellipses $\ldots$ represent the other parameters, which have been omitted for brevity). A better characterization involves introducing additional μ-dependent terms on the right-hand side; for instance, the four-coefficient law devised by Claret (2000) is

$$\frac{I_\lambda(\mu; \ldots)}{I_\lambda(1, \ldots)} = 1 - \sum_{k=1}^{4} a_k(\ldots)\left[1 - \mu^{k/2}\right], \tag{6.2}$$

where there are now four limb-darkening coefficients $a_k(\ldots)$.

The advantage of using limb-darkening laws is the ease with which other useful quantities can be calculated. For instance, the emergent flux

$$F_\lambda(\ldots) = \int_0^1 I_\lambda(\mu; \ldots)\,\mu\,\mu \tag{6.3}$$

can be evaluated analytically, as can any of the Eddington (1926) intensity moments (or *E-moments*, as MSG terms them):

$$\mathcal{E}^i_\lambda(\ldots) = \frac{1}{2}\int_0^1 I_\lambda(\mu; \ldots)\,\mu^i\,\mu.$$

MSG supports the following limb-darkening laws:

CONST
Constant law, where I_λ has no dependence on μ whatsoever. This is discussed further below.

LINEAR
Linear law given in equation (6.1) above.

SQRT
Square-root law introduced by Diaz-Cordoves & Gimenez (1992).

QUAD
Quadratic law introduced by Wade & Rucinski (1985).

CLARET
Four-coefficient law introduced by Claret (2000) and given in equation (6.2) above.

The choice of law is made during grid construction (see the *Grid Tools* appendix for more details). The coefficients appearing in the limb-darkening laws (e.g., a and a_k) are typically determined from least-squares fits to tabulations of the specific intensity. In cases where these tabulations include flux but not specific intensity data, the *CONST* law is used; the angle-independent specific intensity is determined so that it produces the correct flux when evaluated using equation (6.3).

6.1.2 Interpolation in Wavelength

The λ dependence of the specific intensity is represented as a piecewise-constant function on a wavelength abscissa $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_M\}$:

$$I_\lambda(\lambda; \ldots) = I_i(\ldots) \qquad \lambda_i \le \lambda < \lambda_{i+1}.$$

(as before, the ellipses represent the omitted parameters). Mapping intensity data onto a new abscissa $\lambda' = \{\lambda'_1, \lambda'_2, \ldots\ \lambda'_{M'}\}$ is performed conservatively, according to the expression

$$I'_i(\ldots) = \frac{\int_{\lambda'_i}^{\lambda'_{i+1}} I_\lambda(\lambda; \ldots)\lambda}{\lambda'_{i+1} - \lambda'_i}.$$

Beyond its simplicity, the advantage of this approach (as compared to higher-order interpolations) is that the equivalent width of line profiles is preserved.

6.1.3 Interpolation in Atmosphere Parameters

The dependence of the specific intensity on atmosphere parameters $(x, y, z, \ldots)$ is represented using cubic tensor product interpolation. The appendices provide a *(relatively) gentle introduction to tensor product interpolation*. The short version is that intensity, flux, etc. are represented as piecewise-cubic functions *in each atmosphere parameter*, constructed to be continuous and smooth at the join between each piecewise region.

Grids often contain holes and/or ragged boundaries (the latter typically arising near the edge of the region of the $T_{\rm eff} - g$ plane corresponding to super-Eddington luminosity). When an interpolation tries to access such missing data, MSG either switches to a lower-order scheme, or (if there simply aren't sufficient data to interpolate) returns with an error (see the *Exception Handling* section below).

6.2 Disk Storage

MSG spectroscopic and photometric grid data are stored on disk in HDF5 files with a bespoke schema. Throughout this documentation, these files are known as *specgrid* and *photgrid* files, respectively.

6.3 Memory Management

It's often the case that the data stored in *specgrid* and *photgrid* files greatly exceed the available computer memory (RAM). MSG handles such situations by loading data into memory only when they are required. These data are retained in memory until a user-defined capacity limit reached (see the `SpecGrid.cache_limit` and `PhotGrid.cache_limit` properties in the *Python Interface*, and corresponding functionality in the *Fortran* and *C* interfaces); then, data are evicted from the memory cache via a least recently used algorithm.

6.4 Evaluating Photometric Colors

To evaluate a photometric color, MSG convolves a stellar spectrum with appropriate passband response function $S'(\lambda)$. This function represents the combined sensitivity of the optical pathway, filter and the detector. The mean flux in the passband is evaluated as

$$\langle F \rangle = \frac{\int_0^\infty F_\lambda(\lambda) S'(\lambda) \lambda}{\int_0^\infty S'(\lambda) \lambda}, \tag{6.4}$$

...meaning that $S'(\lambda)$ is interpreted as an *energy* response function (see appendix A of Bessell & Murphy, 2012 for a discussion of the relationship between S' and the corresponding photon response function S). The apparent magnitude at the location where $\langle F \rangle$ is measured follows as

$$m = -2.5 \log_{10} \left\langle \frac{F}{F_0} \right\rangle,$$

where the normalizing flux F_0 is determined by the zero-point of the photometric system.

For a given response function, the convolution (6.4) can be performed before or after the interpolations discussed above:

- the 'before' option performs the convolution as a pre-processing step using the **`specgrid_to_photgrid`** tool to create a *photgrid* file from a *specgrid* file (as discussed in the *Creating Photometric Grids* section). This is computationally more efficient, but requires a separate *photgrid* file to be created for each passband.
- the 'after' option loads data from a *specgrid* file, but performs the convolution on-the-fly after each spectrum is interpolated. This is computationally less efficient, but incurs no storage requirements beyond the *specgrid* file.

6.5 Exception Handling

When a call to an MSG routine encounters a problem, the course of action depends on which langauge is being used:

- In Python, an exception is thrown with a (reasonably) relevant subtype and error message.
- In Fortran, if the optional integer argument `stat` is present during the call, then on return `stat` is set to an value indicating the nature of the problem (see the *Parameters* chapter for the list of possible values). If `stat` is not present, then execution halts with an error message printed to standard output.
- In C, if the pointer argument `stat` is non-null during the call, then on return the target of `stat` is set to a value indicating the nature of the problem (see the *Enums* chapter for the list of possible values). If `stat` is null, then execution halts with an error message printed to standard output.

CHAPTER

SEVEN

INSTALLATION

This chapter discusses MSG installation in detail. If you just want to get up and running, have a look at the *Quick Start* chapter.

7.1 Pre-Requisites

To compile and run MSG, you'll need the following software components:

- A modern (2003+) Fortran compiler
- The LAPACK linear algebra library
- The LAPACK95 Fortran 95 interface to LAPACK
- The fypp Fortran pre-processor
- The HDF5 data management library

On Linux and MacOS platforms, these components are bundled together in the MESA Software Development Kit (SDK). Using this SDK is *strongly* recommended.

7.2 Building MSG

Download the MSG source code, and unpack it from the command line using the **tar** utility:

```
$ tar xf msg-1.0.tar.gz
```

Set the `MSG_DIR` environment variable with the path to the newly created source directory; this can be achieved e.g. using the **realpath** command[1]:

```
$ export MSG_DIR=$(realpath msg-1.0)
```

Finally, compile MSG using the **make** utility:

```
$ make -j -C $MSG_DIR
```

(the **-j** flags tells **make** to use multiple cores, speeding up the build). If things go awry, consult the *Troubleshooting* chapter.

[1] The **realpath** command is included in the GNU CoreUtils package. Mac OS users can install CoreUtils using MacPorts or Homebrew.

7.3 Custom Builds

Custom builds of MSG can be created by setting certain environment variables, and/or variables in the file *$MSG_DIR/src/build/Makefile*, to the value `yes`. The following variables are currently supported:

DEBUG
: Enable debugging mode (default `no`)

OMP
: Enable OpenMP parallelization (default `yes`)

FPE
: Enable floating point exception checks (default `yes`)

PYTHON
: Enable building of Python interface (default `yes`)

TEST
: Enable building of testing tools (default `yes`)

TOOLS
: Enable building of development tools (default `no`)

If a variable is not set, then its default value is assumed.

7.4 GitHub Access

Sometimes, you'll want to try out new features in MSG that haven't yet made it into a formal release. In such cases, you can check out MSG directly from the rhdtownsend/msg git repository on GitHub:

```
$ git clone --recurse-submodules https://github.com/rhdtownsend/msg.git
```

However, a word of caution: MSG is under constant development, and features in the `main` branch can change without warning.

footnote

CHAPTER

EIGHT

PYTHON INTERFACE

The Python interface is provided through the `pymsg` module, which defines the *pymsg.SpecGrid* and *pymsg.PhotGrid* classes. It is implemented by Cython-wrapped calls to the *C Interface*.

8.1 API Specification

8.1.1 SpecGrid

class pymsg.**SpecGrid**(*file_name*)

The SpecGrid class represents a grid of spectroscopic intensity data.

This grid may be used to interpolate the intensity (or related quantities) across a wavelength abscissa and for a set of atmosphere parameter values.

__init__(*file_name*)

SpecGrid constructor (via loading data from a *specgrid* file).

Parameters
file_name (*string*) – Name of the file

Returns
Constructed object.

Return type
pymsg.SpecGrid

Raises

- **FileNotFound** – If the the file cannot be found.
- **TypeError** – If the file contains an incorrect datatype.

property rank

Number of dimensions in grid.

Type
int

property axis_labels

Atmosphere parameter axis labels.

Type
list

property axis_x_min

Atmosphere parameter axis minima.

Type
dict

property axis_x_max

Atmosphere parameter axis maxima.

Type
dict

property lam_min

Minimum wavelength of grid.

Type
double

property lam_max

Maximum wavelength of grid.

Type
double

property cache_lam_min

Minimum wavelength of grid cache.

Type
double

property cache_lam_max

Maximum wavelength of grid cache.

Type
double

property cache_count

Number of nodes currently held in grid cache.

Type
int

property cache_limit

Maximum number of nodes to hold in grid cache. Set to 0 to disable caching.

Type
double

intensity(*x*, *mu*, *lam*, *deriv=None*)

Evaluate the spectroscopic intensity.

Parameters

- **x** (*dict*) – Atmosphere parameters; keys must match *axis_labels* property, values must be double.b
- **mu** (*double*) – Cosine of angle of emergence relative to surface normal.
- **lam** (*numpy.ndarray*) – Wavelength abscissa (Å).
- **deriv** (*dict, optional*) – Flags indicating whether to evaluate derivative with respect to each atmosphere parameter; keys must match the *axis_labels* property, values must be boolean.

Returns
Spectroscopic intensity (erg/cm^2/s/Å/sr) in bins delineated by lam; length len(lam)-1.

Return type
numpy.ndarray

Raises

- **KeyError** – If *x* does not define all keys appearing in the *axis_labels* property.
- **ValueError** – If *x*, *mu*, or any part of the wavelength abscissa falls outside the bounds of the grid.
- **LookupError** – If *x* falls in a grid void.

E_moment(*x*, *k*, *lam*, *deriv=None*)

Evaluate the spectroscopic intensity E-moment.

Parameters

- **x** (*dict*) – Atmosphere parameters; keys must match *axis_labels* property, values must be double.
- **k** (*int*) – Degree of moment.
- **lam** (*numpy.ndarray*) – Wavelength abscissa (Å).
- **deriv** (*dict, optional*) – Flags indicating whether to evaluate derivative with respect to each atmosphere parameter; keys must match the *axis_labels* property, values must be boolean.

Returns
Spectroscopic intensity E-moment (erg/cm^2/s/Å) in bins delineated by lam; length len(lam)-1.

Return type
numpy.ndarray

Raises

- **KeyError** – If *x* does not define all keys appearing in the *axis_labels* property.
- **ValueError** – If *x*, *k*, or any part of the wavelength abscissa falls outside the bounds of the grid.
- **LookupError** – If *x* falls in a grid void.

D_moment(*x*, *l*, *lam*, *deriv=None*)

Evaluate the spectroscopic intensity D-moment.

Parameters

- **x** (*dict*) – Atmosphere parameters; keys must match *axis_labels* property, values must be double.
- **l** (*int*) – Harmonic degree of moment.
- **lam** (*numpy.ndarray*) – Wavelength abscissa (Å).
- **deriv** (*dict, optional*) – Flags indicating whether to evaluate derivative with respect to each atmosphere parameter; keys must match the *axis_labels* property, values must be boolean.

Returns
Spectroscopic intensity D-moment (erg/cm^2/s/Å) in bins delineated by lam; length len(lam)-1.

Return type
numpy.ndarray

Raises

- **KeyError** – If *x* does not define all keys appearing in the *axis_labels* property.
- **ValueError** – If *x*, *l*, or any part of the wavelength abscissa falls outside the bounds of the grid.
- **LookupError** – If *x* falls in a grid void.

flux(*x*, *lam*, *deriv=None*)

Evaluate the spectroscopic flux.

Parameters

- **x** (*dict*) – Atmosphere parameters; keys must match *axis_labels* property, values must be double.
- **lam** (*numpy.ndarray*) – Wavelength abscissa (Å)
- **deriv** (*dict, optional*) – Flags indicating whether to evaluate derivative with respect to each atmosphere parameter; keys must match the *axis_labels* property, values must be boolean.

Returns
Spectroscopic flux (erg/cm^2/s/Å) in bins delineated by lam; length len(lam)-1.

Return type
numpy.ndarray

Raises

- **KeyError** – If *x* does not define all keys appearing in the *axis_labels* property.
- **ValueError** – If *x* or any part of the wavelength abscissa falls outside the bounds of the grid.
- **LookupError** – If *x* falls in a grid void.

8.1.2 PhotGrid

class pymsg.**PhotGrid**(*file_name*, *passband_file_name=None*)

The PhotGrid class represents a grid of photometric intensity data.

This grid may be used to interpolate the intensity (or related quantities) for a set of atmosphere parameter values.

__init__(*file_name*, *passband_file_name=None*)

PhotGrid constructor (via loading data from a *photgrid* file, or
from a *specgrid* file together with a passband file).

Parameters

- **file_name** (*string*) – Name of grid file.

- **passband_file_name** (*string*) – Name of passband file (if file_name corresponds to a specgrid file)

Returns
Constructed object.

Return type
pymsg.PhotGrid

Raises

- **FileNotFound** – If either file cannot be found.
- **TypeError** – If either file contains an incorrect datatype.

property rank

Number of dimensions in grid.

Type
int

property axis_labels

Atmosphere parameter axis labels.

Type
list

property axis_x_min

Atmosphere parameter axis minima.

Type
dict

property axis_x_max

Atmosphere parameter axis maxima.

Type
dict

property cache_count

Number of nodes currently held in grid cache.

Type
int

property cache_limit

Maximum number of nodes to hold in grid cache. Set to 0 to disable caching.

Type
double

intensity(*x*, *mu*, *deriv=None*)

Evaluate the photometric intensity, normalized to the zero- point flux.

Parameters

- **x** (*dict*) – Atmosphere parameters; keys must match *axis_labels* property, values must be double.
- **mu** (*double*) – Cosine of angle of emergence relative to surface normal.

- **deriv** (*dict, optional*) – Flags indicating whether to evaluate derivative with respect to each atmosphere parameter; keys must match the *axis_labels* property, values must be boolean.

Returns
photometric intensity (/sr).

Return type
double

Raises

- **KeyError** – If *x* does not define all keys appearing in the *axis_labels* property.
- **ValueError** – If *x* or *mu* falls outside the bounds of the grid.
- **LookupError** – If *x* falls in a grid void.

E_moment(*x*, *k*, *deriv=None*)

Evaluate the photometric intensity E-moment, normalized to the zero-point flux.

Parameters

- **x** (*dict*) – Atmosphere parameters; keys must match *axis_labels* property, values must be double.
- **k** (*int*) – Degree of moment.
- **deriv** (*dict, optional*) – Flags indicating whether to evaluate derivative with respect to each atmosphere parameter; keys must match the *axis_labels* property, values must be boolean.

Returns
photometric intensity E-moment.

Return type
double

Raises

- **KeyError** – If *x* does not define all keys appearing in the *axis_labels* property.
- **ValueError** – If *x* or *k* falls outside the bounds of the grid.
- **LookupError** – If *x* falls in a grid void.

D_moment(*x*, *l*, *deriv=None*)

Evaluate the photometric intensity D-moment, normalized to the zero-point flux.

Parameters

- **x** (*dict*) – Atmosphere parameters; keys must match *axis_labels* property, values must be double.
- **l** (*int*) – Harmonic degree of moment.
- **deriv** (*dict, optional*) – Flags indicating whether to evaluate derivative with respect to each atmosphere parameter; keys must match the *axis_labels* property, values must be boolean.

Returns
photometric intensity D-moment.

Return type
double

Raises

- **KeyError** – If *x* does not define all keys appearing in the *axis_labels* property.
- **ValueError** – If *x* or *l* falls outside the bounds of the grid.
- **LookupError** – If *x* falls in a grid void.

flux(*x*, *deriv=None*)

Evaluate the photometric flux, normalized to the zero-point flux.

Parameters

- **x** (*dict*) – Atmosphere parameters; keys must match *axis_labels* property, values must be double.
- **deriv** (*dict, optional*) – Flags indicating whether to evaluate derivative with respect to each atmosphere parameter; keys must match the *axis_labels* property, values must be boolean.

Returns
photometric flux.

Return type
double

Raises

- **KeyError** – If *x* does not define all keys appearing in the *axis_labels* property.
- **ValueError** – If *x* or *l* falls outside the bounds of the grid.
- **LookupError** – If *x* falls in a grid void.

8.2 Importing

The `pymsg` module is provided in the `$MSG_DIR/python` directory. To import the module, this directory should be added to the Python path — for instance by setting the `PYTHONPATH` environment variable, as discussed here.

CHAPTER

NINE

FORTRAN INTERFACE

The Fortran interface is provided through the *fmsg_m* module. This module defines the *specgrid_t*, *photgrid_t* and *axis_t* derived types, together with supporting parameters and procedures.

9.1 API Specification

9.1.1 Derived Types

specgrid_t

type specgrid_t

The specgrid_t type represents a grid of spectroscopic data.

This grid may be used to interpolate the intensity (or related quantities) across a wavelength abscissa and for a set of atmosphere parameter values.

subroutine get_rank(*rank*)

Get the rank (dimension) of the grid.

Parameters
rank *[integer ,out]* :: Returned rank.

subroutine get_axis(*i*, *axis*)

Get an axis of the grid.

Parameters

- **i** *[integer ,in]* :: Index of axis (between 1 and *rank*)
- **axis** *[axis_t ,out]* :: Returned axis.

subroutine get_lam_min(*lam_min*)

Get the minimum wavelength of the grid.

Parameters
lam_min *[real(RD),out]* :: Returned minimum wavelength.

subroutine get_lam_max(*lam_max*)

Get the maximum wavelength of the grid.

Parameters
lam_max *[real(RD),out]* :: Returned maximum wavelength.

`subroutine get_cache_lam_min`(*cache_lam_min*)

Get the minimum wavelength of the grid cache.

Parameters
lam_min *[real(RD),out]* :: Returned minimum wavelength.

`subroutine get_cache_lam_max`(*cache_lam_max*)

Get the maximum wavelength of the grid cache.

Parameters
cache_lam_max *[real(RD),out]* :: Returned maximum wavelength.

`subroutine get_cache_count`(*cache_count*)

Get the number of nodes currently held in the grid cache.

Parameters
cache_count *[integer ,out]* :: Returned number of nodes.

`subroutine get_cache_limit`(*cache_limit*)

Get the grid cache occupancy limit.

Parameters
cache_limit *[integer ,out]* :: Returned occupancy limit.

`subroutine set_cache_lam_min`(*cache_lam_min*, *stat*)

Set the minimum wavelength of the grid cache.

Parameters
lam_min *[real(RD),in]* :: Minimum wavelength.

Options
stat *[integer ,out]* :: Status code.

`subroutine set_cache_lam_max`(*cache_lam_max*, *stat*)

Set the maximum wavelength of the grid cache.

Parameters
cache_lam_max *[real(RD),in]* :: Maximum wavelength.

Options
stat *[integer ,out]* :: Status code.

`subroutine set_cache_limit`(*cache_limit*, *stat*)

Set the grid cache occupancy limit.

Parameters
cache_limit *[integer ,in]* :: Occupancy limit.

Options
stat *[integer ,out]* :: Status code.

`subroutine interp_intensity`(*x_vec*, *mu*, *lam*, *I*, *stat*, *deriv_vec*)

Interpolate the spectroscopic intensity.

Parameters

- **x_vec** (*) *[real(RD),in]* :: Atmosphere parameter values.
- **mu** *[real(RD),in]* :: Cosine of angle of emergence relative to surface normal.
- **lam** (*) *[real(RD),in]* :: Wavelength abscissa (Å).

- **I** (*) *[real(RD),out]* :: Spectroscopic intensity (erg/cm^2/s/Å/sr) in bins delineated by lam; length LEN(lam)-1.

Options

- **stat** *[integer ,out]* :: Status code.
- **deriv_vec** (*) *[logical ,in]* :: Derivative flags.

`subroutine interp_E_moment`(*x_vec*, *k*, *lam*, *E*, *stat*, *deriv_vec*)

Interpolate the spectroscopic intensity E-moment.

Parameters

- **x_vec** (*) *[real(RD),in]* :: Atmosphere parameter values.
- **k** *[integer ,in]* :: Degree of moment.
- **lam** (*) *[real(RD),in]* :: Wavelength abscissa (Å).
- **E** (*) *[real(RD),out]* :: Spectroscopic intensity E-moment (erg/cm^2/s/Å) in bins delineated by lam; length LEN(lam)-1.

Options

- **stat** *[integer ,out]* :: Status code.
- **deriv_vec** (*) *[logical ,in]* :: Derivative flags.

`subroutine interp_D_moment`(*x_vec*, *l*, *lam*, *D*, *stat*, *deriv_vec*)

Interpolate the spectroscopic intensity D-moment.

Parameters

- **x_vec** (*) *[real(RD),in]* :: Atmosphere parameter values.
- **l** *[integer ,in]* :: Harmonic degree of moment.
- **lam** (*) *[real(RD),in]* :: Wavelength abscissa (Å).
- **D** (*) *[real(RD),out]* :: Spectroscopic intensity D-moment (erg/cm^2/s/Å) in bins delineated by lam; length LEN(lam)-1.

Options

- **stat** *[integer ,out]* :: Status code.
- **deriv_vec** (*) *[logical ,in]* :: Derivative flags.

`subroutine interp_flux`(*x_vec*, *lam*, *I*, *stat*, *deriv_vec*)

Interpolate the spectroscopic flux.

Parameters

- **x_vec** (*) *[real(RD),in]* :: Atmosphere parameter values.
- **lam** (*) *[real(RD),in]* :: Wavelength abscissa (Å).
- **F** (*) *[real(RD),out]* :: Spectroscopic flux (erg/cm^2/s/Å) in bins delineated by lam; length LEN(lam)-1.

Options

- **stat** *[integer ,out]* :: Status code.
- **deriv_vec** (*) *[logical ,in]* :: Derivative flags.

photgrid_t

type photgrid_t

The photgrid_t type represents a grid of photometric data.

This grid may be used to interpolate the intensity (or related quantities) for a set of atmosphere parameter values.

subroutine get_rank(*rank*)

Get the rank (dimension) of the grid.

Parameters
rank *[integer ,out]* :: Returned rank.

subroutine get_axis(*i*, *axis*)

Get an axis of the grid.

Parameters

- **i** *[integer ,in]* :: Index of axis (between 1 and *rank*)
- **axis** *[axis_t ,out]* :: Returned axis.

subroutine get_cache_count(*cache_count*)

Get the number of nodes currently held in the grid cache.

Parameters
cache_count *[integer ,out]* :: Returned number of nodes.

subroutine get_cache_limit(*cache_limit*)

Get the grid cache occupancy limit.

Parameters
cache_limit *[integer ,out]* :: Returned occupancy limit.

subroutine set_cache_limit(*cache_limit*, *stat*)

Set the grid cache occupancy limit.

Parameters
cache_limit *[integer ,in]* :: Occupancy limit.

Options
stat *[integer ,out]* :: Status code.

subroutine interp_intensity(*x_vec*, *mu*, *I*, *stat*, *deriv_vec*)

Interpolate the photometric intensity, normalized to the zero-point flux.

Parameters

- **x_vec** (*) *[real(RD),in]* :: Atmosphere parameter values.
- **mu** *[real(RD),in]* :: Cosine of angle of emergence relative to surface normal.
- **I** *[real(RD),out]* :: Photometric intensity (/sr).

Options

- **stat** *[integer ,out]* :: Status code.
- **deriv_vec** (*) *[logical ,in]* :: Derivative flags.

`subroutine interp_E_moment`(*x_vec*, *k*, *E*, *stat*, *deriv_vec*)

Interpolate the photometric E-moment, normalized to the zero-point flux.

Parameters

- **x_vec** (*) *[real(RD),in]* :: Atmosphere parameter values.
- **k** *[integer ,in]* :: Degree of of moment.
- **E** *[real(RD),out]* :: Photometric E-moment.

Options

- **stat** *[integer ,out]* :: Status code.
- **deriv_vec** (*) *[logical ,in]* :: Derivative flags.

`subroutine interp_D_moment`(*x_vec*, *l*, *D*, *stat*, *deriv_vec*)

Interpolate the photometric D-moment, normalized to the zero-point flux.

Parameters

- **x_vec** (*) *[real(RD),in]* :: Atmosphere parameter values.
- **l** *[integer ,in]* :: Harmonic degree of moment.
- **D** *[real(RD),out]* :: Photometric D-moment.

Options

- **stat** *[integer ,out]* :: Status code.
- **deriv_vec** (*) *[logical ,in]* :: Derivative flags.

`subroutine interp_flux`(*x_vec*, *F*, *stat*, *deriv_vec*)

Interpolate the photometric flux, normalized to the zero-point flux.

Parameters

- **x_vec** (*) *[real(RD),in]* :: Atmosphere parameter values.
- **F** *[real(RD),out]* :: Photometric flux.

Options

- **stat** *[integer ,out]* :: Status code.
- **deriv_vec** (*) *[logical ,in]* :: Derivative flags.

axis_t

`type axis_t`

The axis_t type represents a grid axis.

`subroutine get_n`(*n*)

Get the number of points making up the axis.

Parameters

rank *[integer ,out]* :: Returned number of points.

`subroutine get_x_min`(*x_min*)

Get the minimum value of the axis.

Parameters

x_min *[real(RD),out]* :: Returned minimum value.

subroutine get_x_max(*x_max*)

Get the maximum value of the axis.

Parameters
x_max *[real(RD),out]* :: Returned maximum value.

subroutine get_label(*label*)

Get the axis label

Parameters
character (*) :: Returned label.

subroutine fetch(*i*, *x*, *stat*)

Fetch an axis value

Parameters

- **i** *[integer ,in]* :: Index of value (from 1 to *n*).
- **x** *[real(RD),out]* :: Returned value.

Options
stat *[integer ,out]* :: Status code.

subroutine locate(*x*, *i*)

Locate where along the axis a value falls.

Parameters

- **x** *[real(RD),out]* :: Value to locate.
- **i** *[integer ,out]* :: Location index.

9.1.2 Parameters

int LABEL_LEN

Length of *axis_t* labels.

int STAT_OK

Status code indicating call completed without error.

int STAT_OUT_OF_BOUNDS_RANGE_LO

Status code indicating call encountered an out-of-bounds reference, below the range minimum.

int STAT_OUT_OF_BOUNDS_RANGE_HI

Status code indicating call encountered an out-of-bounds reference, above the range maximum.

int STAT_OUT_OF_BOUNDS_AXIS_LO

Status code indicating call encountered an out-of-bounds reference, below the axis minimum.

int STAT_OUT_OF_BOUNDS_AXIS_HI

Status code indicating call encountered an out-of-bounds reference, above the axis maximum.

int STAT_OUT_OF_BOUNDS_LAM_LO

Status code indicating call encountered an out-of-bounds reference, below the wavelength minimum.

int STAT_OUT_OF_BOUNDS_LAM_HI

Status code indicating call encountered an out-of-bounds reference, above the wavelength maximum.

int STAT_OUT_OF_BOUNDS_MU_LO

Status code indicating call encountered an out-of-bounds reference, below the emergence cosine minimum.

int STAT_OUT_OF_BOUNDS_MU_HI

Status code indicating call encountered an out-of-bounds reference, above the emergence cosine maximum.

int STAT_UNAVAILABLE_DATA

Status code indicating call encountered unavailable data.

int STAT_INVALID_ARGUMENT

Status code indicating call encountered an invalid argument.

int STAT_FILE_NOT_FOUND

Status code indicating call encountered a file that could not be found.

int STAT_INVALID_FILE_TYPE

Status code indicating call encountered a file with an invalid type.

int STAT_INVALID_FILE_REVISION

Status code indicating call encountered a file with an invalid revision number.

9.1.3 Procedures

subroutine load_specgrid(*specgrid_file_name*, *specgrid*, *stat*)

Create a new `specgrid_t` by loading data from a *specgrid* file.

Parameters

- **specgrid_file_name** *[character(*),in]* :: Name of the *specgrid* file.
- **specgrid** *[specgrid_t ,out]* :: Returned grid.

Options

stat *[integer ,out]* :: Returned status code.

subroutine load_photgrid(*photgrid_file_name*, *photgrid*, *stat*)

Create a new `photgrid_t` by loading data from a *photgrid* file.

Parameters

- **photgrid_file_name** *[character(*),in]* :: Name of the *photgrid* file.
- **photgrid** *[specgrid_t ,out]* :: Returned grid.

Options

stat *[integer ,out]* :: Returned status code.

subroutine load_photgrid_from_specgrid(*specgrid_file_name*, *passband_file_name*, *photgrid*, *stat*)

Create a new `photgrid_t` by loading data from a *specgrid* file, and convolving on-the-fly with a passband response function also loaded from file.

Parameters

- **specgrid_file_name** *[character(*),in]* :: Name of the *specgrid* file.
- **passband_file_name** *[character(*),in]* :: Name of the passband file.
- **photgrid** *[specgrid_t ,out]* :: Returned grid.

Options

stat *[integer ,out]* :: Returned status code.

subroutine get_version(*version*)

Get the version of the MSG library

Parameters

character (*) :: Returned version string.

9.2 Compiling/Linking

The module file `fmsg_m` for the Fortran interface is provided in the directory `$MSG_DIR/include`, and executables should be linked against `$MSG_DIR/lib/libfmsg.so` (Linux) or `$MSG_DIR/lib/libfmsg.dylib` (MacOS). To simplify this process, a script `$MSG_DIR/scripts/fmsg_link` is provided that writes the appropriate linker commands to standard output. This script can be used to compile/link a program with **gfortran** as follows:

```
$ gfortran -I $MSG_DIR/include -o myprogram myprogram.f90 `$MSG_DIR/scripts/fmsg_link`
```

CHAPTER

TEN

C INTERFACE

The C interface is provided through a set of functions defined in the `cmsg.h` header file. These functions are implemented using the C interoperability capabilities in Fortram 2003/2008/2018, and are based around a pair of *void* * typedefs — *SpecGrid* and *PhotGrid* — that store pointers to corresponding Fortran *specgrid_t* and *photgrid_t* objects.

10.1 API Specification

10.1.1 Typedefs

type **SpecGrid**

A *void* * pointer to a Fortran *specgrid_t* spectroscopic grid object.

type **PhotGrid**

A *void* * pointer to a Fortran *specgrid_t* photometric grid object.

10.1.2 Functions

SpecGrid Functions

void **load_specgrid**(const char *specgrid_file_name, *SpecGrid* *specgrid, *Stat* *stat)

Create a new *SpecGrid* by loading data from a *specgrid* file.

Parameters

- **specgrid_file_name** – Name of the *specgrid* file.
- **specgrid** – Returned grid.
- **stat** – Returned status code.

void **unload_specgrid**(*SpecGrid* specgrid)

Unload a *specgrid* grid, freeing up memory.

Parameters

- **specgrid** – Grid to unload.

void **get_specgrid_rank**(*SpecGrid* specgrid, int *rank)

Get the rank (dimension) of the grid.

Parameters

- **specgrid** – Grid object.
- **rank** – Returned rank.

void **get_specgrid_lam_min**(*SpecGrid* specgrid, double *lam_min)

Get the minimum wavelength of the grid.

Parameters

- **specgrid** – Grid object.
- **lam_min** – Returned minimum wavelength.

void **get_specgrid_lam_max**(*SpecGrid* specgrid, double *lam_max)

Get the maximum wavelength of the grid.

Parameters

- **specgrid** – Grid object.
- **lam_max** – Returned maximum wavelength.

void **get_specgrid_cache_lam_min**(*SpecGrid* specgrid, double *cache_lam_min)

Get the minimum wavelength of the grid cache.

Parameters

- **specgrid** – Grid object.
- **cache_lam_min** – Returned minimum wavelength.

void **get_specgrid_cache_lam_max**(*SpecGrid* specgrid, double *cache_lam_max)

Get the maximum wavelength of the grid cache.

Parameters

- **specgrid** – Grid object.
- **cache_lam_max** – Returned maximum wavelength.

void **get_specgrid_cache_count**(*SpecGrid* specgrid, int *cache_count)

Get the number of nodes currently held in the grid cache.

Parameters

- **specgrid** – Grid object.
- **cache_count** – Returned number of nodes.

void **get_specgrid_cache_limit**(*SpecGrid* specgrid, int *cache_limit)

Get the maximum number of nodes to hold in the grid cache.

Parameters

- **specgrid** – Grid object.
- **cache_limit** – Returned maximum number of nodes.

void **get_specgrid_axis_x_min**(*SpecGrid* specgrid, int i, double *x_min)

Get the minimum value of the i'th grid axis.

Parameters

- **specgrid** – Grid object.
- **i** – Axis index (beginning at 0).

- **x_min** – Returned minimum value.

void **get_specgrid_axis_x_max**(*SpecGrid* specgrid, int i, double *x_max)

Get the maximum value of the i'th grid axis.

Parameters

- **specgrid** – Grid object.
- **i** – Axis index (beginning at 0).
- **x_max** – Returned maximum value.

void **get_specgrid_axis_label**(*SpecGrid* specgrid, int i, char *label)

Get the label of the i'th grid axis.

Parameters

- **photgrid** – Grid object.
- **i** – Index of the label (beginning at 0).
- **axis_label** – Buffer to store axis label buffer (at least 17 bytes, to accomodate label plus null terminator).

void **set_specgrid_cache_lam_min**(*SpecGrid* specgrid, double cache_lam_min, *Stat* *stat)

Set the minimum wavelength of the grid cache.

Parameters

- **specgrid** – Grid object.
- **cache_lam_min** – Minimum wavelength.
- **stat** – Returned status code (set to NULL if not required).

void **set_specgrid_cache_lam_max**(*SpecGrid* specgrid, double cache_lam_max, *Stat* *stat)

Set the maximum wavelength of the grid cache.

Parameters

- **specgrid** – Grid object.
- **cache_lam_max** – Maximum wavelength.
- **stat** – Returned status code (set to NULL if not required).

void **set_specgrid_cache_limit**(*SpecGrid* specgrid, int cache_limit, *Stat* *stat)

Set the maximum number of notes to hold in the grid cache. Set to 0 to disable caching.

Parameters

- **specgrid** – Grid object.
- **cache_limit** – Maximum number of nodes.
- **stat** – Returned status code (set to NULL if not required).

void **interp_specgrid_intensity**(*SpecGrid* specgrid, double x_vec[], double mu, int n, double lam[], double I[], *Stat* *stat, bool deriv_vec[])

Interpolate the spectroscopic intensity.

Parameters

- **specgrid** – Grid object.
- **x_vec** – Atmospheric parameter values.

- **mu** – Cosine of angle of emergence relative to surface normal.
- **n** – Number of points in wavelength abscissa.
- **lam[n]** – Wavelength abscissa (Å).
- **I[n-1]** – Returned spectroscopic intensity (erg/cm^2/s/Å/sr) in bins delineated by lam
- **stat** – Returned status code (set to NULL if not required).
- **deriv_vec** – Derivative flags (set to NULL if not required).

void **interp_specgrid_E_moment**(*SpecGrid* specgrid, double x_vec[], int k, int n, double lam[], double E[], *Stat* *stat, bool deriv_vec[])

Interpolate the spectroscopic intensity E-moment.

Parameters

- **specgrid** – Grid object.
- **x_vec** – Atmospheric parameter values.
- **k** – Degree of moment.
- **n** – Number of points in wavelength abscissa.
- **lam[n]** – Wavelength abscissa (Å).
- **D[n-1]** – Returned spectroscopic intensity E-moment (erg/cm^2/s/Å) in bins delineated by lam
- **stat** – Returned status code (set to NULL if not required).
- **deriv_vec** – Derivative flags (set to NULL if not required).

void **interp_specgrid_D_moment**(*SpecGrid* specgrid, double x_vec[], int l, int n, double lam[], double D[], *Stat* *stat, bool deriv_vec[])

Interpolate the spectroscopic intensity D-moment.

Parameters

- **specgrid** – Grid object.
- **x_vec** – Atmospheric parameter values.
- **l** – Harmonic degree of moment.
- **n** – Number of points in wavelength abscissa.
- **lam[n]** – Wavelength abscissa (Å).
- **D[n-1]** – Returned spectroscopic intensity D-moment (erg/cm^2/s/Å) in bins delineated by lam
- **stat** – Returned status code (set to NULL if not required).
- **deriv_vec** – Derivative flags (set to NULL if not required).

void **interp_specgrid_flux**(*SpecGrid* specgrid, double x_vec[], int n, double lam[], double F[], *Stat* *stat, bool deriv_vec[])

Interpolate the spectroscopic flux.

Parameters

- **specgrid** – Grid object.
- **x_vec** – Atmospheric parameter values.

- **n** – Number of points in wavelength abscissa.
- **lam[n]** – Wavelength abscissa (Å).
- **F[n-1]** – Returned spectroscopic flux (erg/cm^2/s/Å) in bins delineated by lam
- **stat** – Returned status code (set to NULL if not required).
- **deriv_vec** – Derivative flags (set to NULL if not required).

PhotGrid Functions

void **load_photgrid**(const char *photgrid_file_name, *PhotGrid* *photgrid, int *stat)

Create a new *PhotGrid* by loading data from a *photgrid* file.

Parameters

- **photgrid_file_name** – Name of the *photgrid* file.
- **photgrid** – Returned grid object.
- **stat** – Returned status code.

void **load_photgrid_from_specgrid**(const char *specgrid_file_name, const char *passband_file_name, *PhotGrid* *photgrid, int *stat)

Create a new *PhotGrid* by loading data from a *specgrid* file, and convolving on-the-fly with a passband response function also loaded from file.

Parameters

- **specgrid_file_name** – Name of the *specgrid* file.
- **passband_file_name** – Name of the passband file.
- **photgrid** – Returned grid object.
- **stat** – Returned status code.

void **unload_photgrid**(*PhotGrid* photgrid)

Unload a photometric grid, freeing up memory.

Parameters

- **photgrid** – Grid object.

void **get_photgrid_rank**(Photgrid photgrid, int *rank)

Get the rank (dimension) of the grid.

Parameters

- **photgrid** – Grid object.
- **rank** – Returned rank.

void **get_photgrid_cache_count**(Photgrid photgrid, int *cache_count)

Get the number of nodes currently held in the grid cache.

Parameters

- **photgrid** – Grid object.
- **ceche_count** – Returned number of nodes.

void **get_photgrid_cache_limit**(Photgrid photgrid, int *cache_limit)

Get the maximum number of nodes to hold in the grid cache.

Parameters

- **photgrid** – Grid object.
- **cache_limit** – Returned maximum number of nodes.

void **get_photgrid_axis_x_min**(Photgrid photgrid, int i, double *x_min)

Get the minimum value of the i'th grid axis.

Parameters

- **photgrid** – Grid object.
- **i** – Axis index (beginning at 0).
- **x_min** – Returned minimum value.

void **get_photgrid_axis_x_max**(Photgrid photgrid, int i, double *x_max)

Get the maximum value of the i'th grid axis.

Parameters

- **photgrid** – Grid object.
- **i** – Axis index (beginning at 0).
- **x_max** – Returned maximum value.

void **get_photgrid_axis_label**(*SpecGrid* specgrid, int i, char *label)

Get the label of the i'th grid axis.

Parameters

- **photgrid** – Grid object.
- **i** – Index of the label (beginning at 0).
- **axis_label** – Buffer to store axis label buffer (at least 17 bytes, to accomodate label plus null terminator).

void **set_photgrid_cache_limit**(Photgrid photgrid, int cache_limit, int *stat)

Set the maximum number of notes to hold in the grid cache. Set to 0 to disable caching.

Parameters

- **photgrid** – Grid object.
- **cache_limit** – Maximum number of nodes.
- **stat** – Returned status code (set to NULL if not required).

void **interp_photgrid_intensity**(*PhotGrid* photgrid, double x_vec[], double mu, double *I, int *stat, bool deriv_vec[])

Interpolate the photometric intensity, normalized to the zero-point flux.

Parameters

- **photgrid** – Grid object.
- **x_vec** – Atmospheric parameter values.
- **mu** – Cosine of angle of emergence relative to surface normal.
- **I** – Returned photometric intensity (/sr).

- **stat** – Returned status code (set to NULL if not required).
- **deriv_vec** – Derivative flags (set to NULL if not required).

void **interp_photgrid_E_moment**(*PhotGrid* photgrid, double x_vec[], int k, double *E, int *stat, bool deriv_vec[])

Interpolate the photometric intensity E-moment, normalized to the zero-point flux.

Parameters

- **photgrid** – Grid object.
- **x_vec** – Atmospheric parameter values.
- **k** – Degree of moment.
- **D** – Returned photometric intensity E-moment.
- **stat** – Returned status code (set to NULL if not required).
- **deriv_vec** – Derivative flags (set to NULL if not required).

void **interp_photgrid_D_moment**(*PhotGrid* photgrid, double x_vec[], int l, double *D, int *stat, bool deriv_vec[])

Interpolate the photometric intensity D-moment, normalized to the zero-point flux.

Parameters

- **photgrid** – Grid object.
- **x_vec** – Atmospheric parameter values.
- **l** – Harmonic degree of moment.
- **D** – Returned photometric intensity D-moment.
- **stat** – Returned status code (set to NULL if not required).
- **deriv_vec** – Derivative flags (set to NULL if not required).

void **interp_photgrid_flux**(*PhotGrid* photgrid, double x_vec[], double *F, int *stat, bool deriv_vec[])

Interpolate the photometric flux, normalized to the zero-point flux.

Parameters

- **PhotGrid** – Grid object.
- **x_vec** – Atmospheric parameter values.
- **F** – Returned photometric flux.
- **stat** – Returned status code (set to NULL if not required).
- **deriv_vec** – Derivative flags (set to NULL if not required).

10.1.3 Enums

enum **Stat**

enumerator **STAT_OK**

Status code indicating call completed without error.

enumerator **STAT_OUT_OF_BOUNDS_RANGE_LO**

Status code indicating call encountered an out-of-bounds reference, below the range minimum.

enumerator **STAT_OUT_OF_BOUNDS_RANGE_HI**

Status code indicating call encountered an out-of-bounds reference, above the range maximum.

enumerator **STAT_OUT_OF_BOUNDS_AXIS_LO**

Status code indicating call encountered an out-of-bounds reference, below the axis minimum.

enumerator **STAT_OUT_OF_BOUNDS_AXIS_HI**

Status code indicating call encountered an out-of-bounds reference, above the axis maximum.

enumerator **STAT_OUT_OF_BOUNDS_LAM_LO**

Status code indicating call encountered an out-of-bounds reference, below the wavelength minimum.

enumerator **STAT_OUT_OF_BOUNDS_LAM_HI**

Status code indicating call encountered an out-of-bounds reference, above the wavelength maximum.

enumerator **STAT_OUT_OF_BOUNDS_MU_LO**

Status code indicating call encountered an out-of-bounds reference, below the emergence cosine minimum.

enumerator **STAT_OUT_OF_BOUNDS_MU_HI**

Status code indicating call encountered an out-of-bounds reference, above the emergence cosine maximum.

enumerator **STAT_UNAVAILABLE_DATA**

Status code indicating call encountered unavailable data.

enumerator **STAT_INVALID_ARGUMENT**

Status code indicating call encountered an invalid argument.

enumerator **STAT_FILE_NOT_FOUND**

Status code indicating call encountered a file that could not be found.

enumerator **STAT_INVALID_FILE_TYPE**

Status code indicating call encountered a file with an invalid type.

enumerator **STAT_INVALID_FILE_REVISION**

Status code indicating call encountered a file with an invalid revision number.

10.2 Compiling/Linking

Headers for the C interface are provided in the header file `$MSG_DIR/include/cmsg.h`, and executables should be linked against `$MSG_DIR/lib/libcmsg.so` (Linux) or `$MSG_DIR/lib/libcmsg.dylib` (MacOS). To simplify this process, a script `$MSG_DIR/scripts/cmsg_link` is provided that writes the appropriate linker commands to standard output. This script can be used to compile/link a program with **gcc** as follows:

```
$ gcc -I $MSG_DIR/include -o myprogram myprogram.c `$MSG_DIR/scripts/cmsg_link`
```

CHAPTER

ELEVEN

TROUBLESHOOTING

11.1 Missing `forum.inc`

During compilation, if the error `include file 'forum.inc' not found` arises then it's likely you have an incomplete copy of the source code. Verify that when you *checked out* the code from GitHub, you used the `--recurse-submodules` option.

11.2 Other Issues

If you encouter something else that doesn't work as it should, please open an issue.

CHAPTER

TWELVE

GRID FILES

Due to their large size and gradually evolving content (as improvements are made), HDF5 *specgrid* files are not shipped as part of the rhdtownsend/msg git repository; they must be downloaded separately (the sole exception is the demo grid, `$MSG_DIR/data/grids/sg-demo.h5`). This chapter describes the various grids currently available.

12.1 OSTAR2002 Grids

The OSTAR2002 grids are based on the line-blanketed O-star model atmospheres published in Lanz & Hubeny (2003). Spectra are calculated from these atmospheres using SYNSPEC, and their angle dependence parameterized with the *CLARET* limb-darkening law.

File	Resolution	λ Range	Atmospheric Parameters
sg-OSTAR2002-low.h5 (50MB)	$\mathcal{R} = 1000$	$[880\,, 5.0]$	`Teff`, `log(g)`, `Z/Zo`
sg-OSTAR2002-medium.h5 (447MB)	$\mathcal{R} = 10000$	$[880\,, 5.0]$	`Teff`, `log(g)`, `Z/Zo`
sg-OSTAR2002-high.h5 (4.2GB)	$\mathcal{R} = 100000$	$[880\,, 5.0]$	`Teff`, `log(g)`, `Z/Zo`

The definitions and ranges of the atmospheric parameters are as follows:

- effective temperature `Teff` $= / \in [27500, 55000]$
- surface gravity `log(g)` $= \log_{10}(g/\ ^{-2}) \in [3.0, 4.75]$
- metallicity `Z/Zo` $= Z/Z_\odot \in [0.02, 2]$

12.2 BSTAR2006 Grids

The BSTAR2006 grids are based on the line-blanketed B-star model atmospheres published in Lanz & Hubeny (2007). Spectra are calculated from these atmospheres using SYNSPEC, and their angle dependence parameterized with the *CLARET* limb-darkening law.

File	Resolution	λ Range	Atmospheric Parameters
sg-BSTAR2006-low.h5 (77MB)	$\mathcal{R} = 1000$	$[880\,,5\,]$	`Teff`, `log(g)`, `Z/Zo`
sg-BSTAR2006-medium.h5 (693MB)	$\mathcal{R} = 10000$	$[880\,,5\,]$	`Teff`, `log(g)`, `Z/Zo`
sg-BSTAR2006-high.h5 (6.5GB)	$\mathcal{R} = 100000$	$[880\,,5\,]$	`Teff`, `log(g)`, `Z/Zo`

The definitions and ranges of the atmospheric parameters are as follows:

- effective temperature `Teff` $= / \in [15000, 30000]$
- surface gravity `log(g)` $= \log_{10}(g/^{-2}) \in [1.753.00, 4.75]$
- metallicity `Z/Zo` $= Z/Z_{\odot} \in [0, 2]$

12.3 CAP18 Grids

The CAP18 grids are based on the data published in Allende Prieto et al. (2018) (the letters 'CAP' are the initials of the first author). The angle dependence of spectra is parameterized with the *CONST* limb-darkening law.

File	Resolution	λ Range	Atmospheric Parameters
sg-CAP18-large.h5 (73GB)	$\mathcal{R} = 10000$	$[1300\,,6.5\,]$	`Teff`, `log(g)`, `[Fe/H]`, `[alpha/Fe]`, `log(xi)`
sg-CAP18-coarse.h5 (339MB)	$\mathcal{R} = 10000$	$[1300\,,6.5\,]$	`Teff`, `log(g)`, `[Fe/H]`
sg-CAP18-high.h5 (2.9GB)	$\mathcal{R} = 100000$	$[1300\,,6.5\,]$	`Teff`, `log(g)`, `[Fe/H]`
sg-CAP18-ultra.h5 (5.2GB)	$\mathcal{R} = 300000$	$[1300\,,6.5\,]$	`Teff`, `log(g)`, `[Fe/H]`

The definitions and ranges of the atmospheric parameters are as follows:

- effective temperature `Teff` $= / \in [27500, 55000]$
- surface gravity `log(g)` $= \log_{10}(g/^{-2}) \in [3.0, 4.5]$
- metallicity `[Fe/H]` $= \log_{10}[(\mathrm{Fe/H})/(\mathrm{Fe/H})_{\odot}] \in [-5.0, 0.5]$
- alpha enhancement `[alpha/Fe]` $= \log_{10}[(\alpha/\mathrm{Fe})/(\alpha/\mathrm{Fe})_{\odot}] \in [-5.0, 0.5]$
- microturbulent velocity `log(xi)` $= \log_{10}(\xi/^{-1}) \in [-0.301, 0.903]$

12.4 Göttingen Grids

The Göettingen grids are based on the data described in Husser et al. (2013) and available for download from phoenix.astro.physik.uni-goettingen.de. The angle dependence of spectra is parameterized with the *CONST* limb-darkening law.

File	Resolution	λ Range	Atmospheric Parameters
sg-Goettingen-HiRes.h5 (116GB)	variable	$[500\,, 5.5]$	`Teff`, `log(g)`, `[Fe/H]`, `[alpha/Fe]`
sg-Goettingen-MedRes-A.h5 (6.0GB)	$\Delta\lambda = 1$ ($10\times$ over-sampled)	$[3000\,, 1.0]$	`Teff`, `log(g)`, `[Fe/H]`, `[alpha/Fe]`
sg-Goettingen-MedRes-R.h5 (18GB)	$\mathcal{R} = 10,000$ ($10\times$ over-sampled)	$[3000\,, 2.5]$	`Teff`, `log(g)`, `[Fe/H]`, `[alpha/Fe]`

The definitions and ranges of the atmospheric parameters are as follows:

- effective temperature `Teff` $= / \in [2\,300, 12\,000]$
- surface gravity `log(g)` $= \log_{10}(g/\,^{-2}) \in [0.0, 6.0]$
- metallicity `[Fe/H]` $= \log_{10}[(\mathrm{Fe/H})/(\mathrm{Fe/H})_\odot] \in [-4.0, 1.0]$
- alpha enhancement `[alpha/H]` $= \log_{10}[(\alpha/\mathrm{Fe})/(\alpha/\mathrm{Fe})_\odot] \in [-0.2, 1.2]$

CHAPTER

THIRTEEN

PASSBAND FILES

As with the *Grid Files*, the passband files used by MSG are not shipped as part of the rhdtownsend/msg git repository; they must be downloaded separately. This chapter describes the various files available for download.

13.1 SVO Filter Service Passbands

The Spanish Virtual Observatory (SVO) offers a database of filters and calibrations for a wide array of photometric systems. The table below provides links to tar archives of passband files created for selected systems in the SVO database using the **`make_passband`** tool. For each filter of each system, separate passband files for the Vega, AB and ST magnitude systems are provided.

File	Facility	Instrument	Filters
Corot.tar.bz2	Corot	—	AS, PF
GAIA-Gaia2m.tar.bz2	GAIA	Gaia2m	Gbp_faint, Gbp_bright, G, Grp
GAIA-GAIA2r.tar.bz2	GAIA	GAIA2r	Gbp, G, Grp
GAIA-GAIA2.tar.bz2	GAIA	GAIA2	Gbp, G, Grp
GAIA-GAIA3.tar.bz2	GAIA	GAIA3	Gbp, G, Grp
GAIA-GAIA0.tar.bz2	GAIA	GAIA0	Gbp, G, Grp
GALEX.tar.bz2	GALEX	—	FUV, NUV
Generic-Johnson.tar.bz2	Generic	Johnson	U, B, V, R, I, J, M
Geneva.tar.bz2	Geneva	—	U, B1, B, B2, V2, V, G
Hipparcos.tar.bz2	Hipparcos	—	Hp, Hp_bes, Hp_MvB

continues on next page

Table 13.1 – continued from previous page

File	Facility	Instrument	Filters
HST-STIS_FUV.tar.bz2	HST	STIS_FUV	F25LYA, F25LYA_G140L, F25LYA_G140M, 25MAMA_G140M, F25ND5_G140M, 25MAMA_G140L, F25ND3_G140M, F25ND5_G140L, 25MAMA, F25ND3_G140L, F25ND5, F25ND3, F25NDQ2_G140M, F25NDQ3_G140M, F25NDQ2_G140L, F25NDQ1_G140M, F25NDQ3_G140L, F25NDQ1_G140L, F25NDQ2, F25NDQ4_G140M, F25NDQ3, F25NDQ1, F25SRF2_G140M, F25NDQ4_G140L, F25SRF2_G140L, F25NDQ4, F25SRF2, F25QTZ_G140M, F25QTZ_G140L, F25QTZ
HST-ACS_SBC.tar.bz2	HST	ACS_SBC	F122M, F115LP, PR130L, PR110L, F125LP, F140LP, F150LP, F165LP
HST-WFPC1-WF.tar.bz2	HST	WFPC1-WF	G200M2, F157W, F194W, F230W, F122M, F284W, F336W, F368M, F375N, F413M, F439W, F437N, G200, F469N, F487N, F492M, G450, F502N, F517N, F555W, F547M, F569W, F588N, F606W, F8ND, POL0, POL120, POL60, F128LP, F622W, F631N, F648M, F656N, F658N, F664N, F673N, F675W, F702W, F718M, G800, F791W, F814W, F725LP, F875M, F889N, F785LP, F850LP, F1042M, F1083N
HST-WFPC1-PC.tar.bz2	HST	WFPC1-PC	G200M2, F157W, F194W, F230W, F284W, F336W, F368M, F375N, F413M, F439W, F437N, F469N, F122M, F487N, F492M, F502N, G450, G200, F517N, F555W, F547M, F569W, F588N, F606W, F622W, F8ND, POL0, POL120, POL60, F128LP, F631N, F648M, F656N, F658N, F664N, F673N, F675W, F702W, F718M, G800, F791W, F814W, F725LP, F875M, F889N, F785LP, F850LP, F1042M, F1083N

continues on next page

Table 13.1 – continued from previous page

File	Facility	Instrument	Filters
HST-WFPC2-WF.tar.bz2	HST	WFPC2-WF	F122M, F160BW, F157W, F170W, F185W, F218W, F255W, F300W, F336W, F343N, F375N, FQUVN33, FQUVN_B, F390N, FQUVN_C, FQUVN_D, F380W, F410M, F439W, F437N, F450W, F467M, F469N, F487N, F502N, F555W, FQCH4N_D, F547M, F569W, F588N, F606W, FQCH4N33, F622W, F631N, F165LP, F130LP, F656N, F658N, F673N, F675W, F702W, POLQ, POLQ_90, POLQ_45, FQCH4N_B, F814W, F791W, F785LP, FQCH4N_C, F850LP, F953N, F1042M
HST-WFPC2-PC.tar.bz2	HST	WFPC2-PC	F122M, F160BW, F157W, F170W, F185W, F218W, F255W, F300W, F336W, F343N, F375N, FQUVN, F390N, F380W, F410M, F439W, F437N, F450W, F467M, F469N, F487N, F502N, F555W, FQCH4N, F547M, F569W, F588N, F606W, F622W, F631N, F165LP, F130LP, F656N, F658N, F673N, F675W, F702W, POLQ, F814W, F791W, F785LP, F850LP, F953N, F1042M
HST-HSP_UV1.tar.bz2	HST	HSP_UV1	F122M_B, F122M_A, F135W_A, F135W_B, F145M_A, F145M_B, PRISM_BLUE, F152M_A, F152M_B, F184W_A, F184W_B, F218M_A, F218M_B, F220W_A, F220W_B, F140LP_A, F140LP_B, F240W_A, F240W_B, F248M_A, F248M_B, PRISM_RED, F278N_A, F278N_B
HST-HSP_UV2.tar.bz2	HST	HSP_UV2	F122M_B, F122M_A, F145M_A, F145M_B, PRISM_BLUE, F152M_A, F152M_B, F179M_A, F179M_B, F184W_A, F184W_B, F218M_A, F218M_B, F140LP_A, F160LP_A, F140LP_B, F160LP_B, F248M_A, F248M_B, F262M_A, F262M_B, PRISM_RED, F278N_A, F278N_B, F284M_A, F284M_B
HST-FOC_F48.tar.bz2	HST	FOC_F48	F140W, F150W, F175W, F195W, F220W, F275W, PRISM3, F130LP, F342W, PRISM1, F180LP, PRISM2, F305LP, F430W

continues on next page

Table 13.1 – continued from previous page

File	Facility	Instrument	Filters
HST-FOC_F96.tar.bz2	HST	FOC_F96	F120M, F130M, F140M, F140W, F152M, F170M, F175W, F165W, F190M, F195W, F210M, F220W, F231M, F253M, F275W, F278M, F307M, F320W, F6ND, F2ND, F342W, F1ND, PRISM1, F346M, F130LP, POL120, POL0, F4ND, PRISM2, POL60, F8ND, F372M, F410M, F430W, F370LP, F437M, F470M, F486N, F502M, F501N, F480LP, F550M, F600M, F630M
HST-HSP_VIS.tar.bz2	HST	HSP_VIS	F184W_A, F184W_B, PRISM_BLUE, F240W_A, F240W_B, F262M_A, F262M_B, F355M_A, F355M_B, F160LP_A, F160LP_B, F419N_A, F419N_B, F450W_A, F450W_B, F400LP_A, F400LP_B, F551W_B, F551W_A, PRISM_RED, F620W_B, F620W_A

continues on next page

Table 13.1 – continued from previous page

File	Facility	Instrument	Filters
HST-STIS_NUV.tar.bz2	HST	STIS_NUV	F25CN182, F25CN182_PRISM, F25CIII_PRISM, F25CIII, F25CIII_G230L, F25CIII_G230M, F25CN182_G230L, F25CN182_G230M, 25MAMA, F25NDQ1, 25MAMA_PRISM, F25QTZ, F25SRF2, F25QTZ_PRISM, F25SRF2_PRISM, F25NDQ1_PRISM, 25MAMA_G230L, F25QTZ_G230L, F25SRF2_G230L, 25MAMA_G230M, F25NDQ1_G230L, F25QTZ_G230M, F25SRF2_G230M, F25NDQ2, F25NDQ1_G230M, F25NDQ2_PRISM, F25NDQ2_G230L, F25ND3, F25NDQ2_G230M, F25ND3_PRISM, F25ND3_G230L, F25ND3_G230M, F25NDQ3, F25NDQ3_G230L, F25CN270_G230L, F25CN270, F25CN270_PRISM, F25CN270_G230M, F25NDQ3_PRISM, F25NDQ3_G230M, F25NDQ4, F25NDQ4_PRISM, F25MGII, F25MGII_PRISM, F25MGII_G230L, F25MGII_G230M, F25NDQ4_G230L, F25NDQ4_G230M, F25ND5, F25ND5_PRISM, F25ND5_G230L, F25ND5_G230M

continues on next page

Table 13.1 – continued from previous page

File	Facility	Instrument	Filters
HST-WFC3_UVIS2.tar.bz2	HST	WFC3_UVIS2	F218W, FQ232N, F225W, FQ243N, G280, F275W, F300X, F280N, F336W, F343N, F373N, FQ378N, FQ387N, F390M, F390W, F395N, F410M, FQ422M, F438W, FQ436N, FQ437N, F467M, F469N, F475W, F487N, F200LP, F475X, FQ492N, F502N, FQ508N, F555W, F547M, FQ575N, F350LP, F606W, FQ619N, F621M, F625W, F631N, FQ634N, F645N, F656N, F657N, F658N, F665N, FQ672N, FQ674N, F673N, F680N, F689M, F600LP, FQ727N, FQ750N, F763M, F775W, F814W, F845M, FQ889N, FQ906N, F850LP, FQ924N, FQ937N, F953N
HST-WFC3_UVIS1.tar.bz2	HST	WFC3_UVIS1	F218W, FQ232N, F225W, FQ243N, F275W, G280, F300X, F280N, F336W, F343N, F373N, FQ378N, FQ387N, F390M, F390W, F395N, F410M, FQ422M, F438W, FQ436N, FQ437N, F467M, F469N, F475W, F487N, F475X, FQ492N, F502N, FQ508N, F200LP, F555W, F547M, FQ575N, F350LP, F606W, FQ619N, F621M, F625W, F631N, FQ634N, F645N, F656N, F657N, F658N, F665N, FQ672N, FQ674N, F673N, F680N, F689M, F600LP, FQ727N, FQ750N, F763M, F775W, F814W, F845M, FQ889N, FQ906N, F850LP, FQ924N, FQ937N, F953N
HST-ACS_HRC.tar.bz2	HST	ACS_HRC	F220W, F250W, F330W, F344N, FR388N, F435W, FR459M, F475W, F502N, FR505N, F555W, F550M, F606W, PR200L, F625W, FR656N, F658N, F660N, POL_UV, POL_V, G800L, F775W, F814W, F892N, F850LP, FR914M
HST-HSP_POL.tar.bz2	HST	HSP_POL	F216M_0, F237M_0, F277M_0, F327M_0, F160LP_T, F160LP_A

continues on next page

Table 13.1 – continued from previous page

File	Facility	Instrument	Filters
HST-STIS_CCD.tar.bz2	HST	STIS_CCD	F28X50LP_G230LB, 50CCD_G230LB, 50CORON_G230LB, F28X50LP_G230MB, 50CCD_G230MB, 50CORON_G230MB, F28X50OII_G430L, F28X50OII_G430M, F28X50OII, 50CCD_G430M, 50CORON_G430M, 50CCD_G430L, 50CORON_G430L, F28X50OIII, F28X50OIII_G430L, F28X50OIII_G430M, F28X50LP_G430M, F28X50LP_G430L, 50CCD, 50CORON, 50CCD_G750L, 50CORON_G750L, F28X50LP, 50CCD_G750M, 50CORON_G750M, F28X50LP_G750L, F28X50LP_G750M
HST-FOS_BLUE.tar.bz2	HST	FOS_BLUE	G130H, MIRROR, G190H, G160L, G400H, G270H, PRISM
HST-ACS_WFC.tar.bz2	HST	ACS_WFC	FR388N, FR423N, F435W, FR459M, FR462N, F475W, F502N, FR505N, F555W, FR551N, F550M, F606W, FR601N, F625W, FR647M, FR656N, F658N, F660N, POL_UV, POL_V, FR716N, G800L, F775W, FR782N, F814W, FR853N, F850LP, F892N, FR914M, FR931N, FR1016N
HST-FOS_RED.tar.bz2	HST	FOS_RED	G780H, G160L, G190H, MIRROR, G400H, G270H, PRISM, G570H, G650L
HST-FGS.tar.bz2	HST	FGS	ND5, PUPIL, F583W, F605W, F550W, F650W
HST-NICMOS1.tar.bz2	HST	NICMOS1	F090M, F095N, F097N, POL0S, POL240S, POL120S, F108N, F110M, F113N, F110W, F140W, F145M, F160W, F165M, F164N, F166N, F170M, F187N, F190N
HST-WFC3_IR.tar.bz2	HST	WFC3_IR	F098M, G102, F105W, F110W, F125W, F126N, F127M, F128N, F130N, F132N, F139M, G141, F140W, F153M, F160W, F164N, F167N

continues on next page

Table 13.1 – continued from previous page

File	Facility	Instrument	Filters
HST-NICMOS3.tar.bz2	HST	NICMOS3	G096, F108N, F110W, F113N, F150W, G141, F160W, F164N, F166N, F175W, F187N, F190N, F196N, F200N, F212N, F215N, G206, F222M, F240M
HST-NICMOS2.tar.bz2	HST	NICMOS2	F110W, F160W, F165M, F171M, F180M, F187W, F187N, F190N, POL120L, POL240L, POL0L, F204M, F205W, F207M, F212N, F215N, F216N, F222M, F237M
JWST-NIRCam.tar.bz2	JWST	NIRCam	F070W, F090W, F115W, F140M, F150W, F162M, F164N, F150W2, F182M, F187N, F200W, F210M, F212N, F250M, F277W, F300M, F323N, F322W2, F335M, F356W, F360M, F405N, F410M, F430M, F444W, F460M, F466N, F470N, F480M
JWST-NIRISS.tar.bz2	JWST	NIRISS	F090W, F115W, F140M, F150W, F158M, F200W, F277W, F356W, F380M, F430M, F444W, F480M
JWST-MIRI.tar.bz2	JWST	MIRI	F560W, F770W, F1000W, F1065C, F1130W, F1140C, F1280W, F1500W, F1550C, F1800W, F2100W, F2300C, F2550W
Kepler.tar.bz2	Kepler	—	K
LSST.tar.bz2	LSST	—	u_filter, u, g_filter, g, r, r_filter, i, i_filter, z, z_filter, y, y_filter
MOST.tar.bz2	MOST	—	band
PAN-STARRS-PS1.tar.bz2	PAN-STARRS	PS1	g, r, w, open, i, z, y
Spitzer-IRAC.tar.bz2	Spitzer	IRAC	I1, I2, I3, I4
Spitzer-IRS.tar.bz2	Spitzer	IRS	Blue, Red
Spitzer-MIPS.tar.bz2	Spitzer	MIPS	24mu, 70mu, 160mu
Generic-Stromgren.tar.bz2	Generic	Stromgren	u, v, b, y
TESS.tar.bz2	TESS	—	Red
WFIRST-WFI.tar.bz2	WFIRST	WFI	R062, Z087, Y106, Prism, J129, W146, Grism, H158, F184

CHAPTER

FOURTEEN

GRID TOOLS

This appendix outlines the set of tools provided with MSG to assist in creating and managing custom grids. These tools are built during compilation when the `TOOLS` enivoronment variable is set to *yes* (see the *Installation* chapter for further details); once built, they can be found in the `$MSG_DIR/bin` directory.

14.1 Extracting Spectra

MSG's HDF5 *specgrid* files are built from a set of *specint* (spectroscopic intensity) files representing individual spectra at the grid nodes. These files are themselves extracted from pre-calculated grids, with a variety of supported formats as discussed in each of the following sections.

14.1.1 SYNSPEC

The **`synspec_to_specint`** tool extracts an single intensity spectrum from a `fort.18` data file produced by the SYNSPEC spectral synthesis package (Lanz & Hubeny, 2003), and writes it to an *specint* file. This tool accepts the following command-line arguments:

`<synspec_file_name>`
: Name of input file.

`<n_mu>`
: Number of μ values in input file (as specified by the `nmu` parameter in the `fort.55` SYNSPEC control file).

`<mu_0>`
: Minimum μ value in input file (as specified by the `ang0` parameter in the `fort.55` SYNSPEC control file).

`<lam_min>`
: Minimum wavelength of output file.

`<lam_max>`
: Maximum wavelength in output file.

`<R>`
: Resolution $\mathcal{R} = \lambda/\Delta\lambda$ in output file.

`<law_str>`
: Limb-darkening law in output file (see the *Limb-Darkening Laws* section for a list of options).

`<specint_file_name>`
: Name of output file.

<label> (optional)

Label of atmosphere parameter (must be accompanied by a corresponding *<value>* argument).

<value> (optional)

Lalue of atmosphere parameter (must be accompanied by a corresponding *<label>* argument).

Note that *<label>* and *<value>* parameters must be paired; and that there can be multiple of these pairs. For the law selected by the *<law_str>* option, the tool calculates the limb-darkening coefficients at each wavelength via a least-squares fit to the function

$$y(\mu) = 1 - \frac{I_\lambda(\mu; \ldots)}{I_\lambda(1; \ldots)}.$$

14.1.2 FERRE

The **ferre_to_specint** tool extracts a series of flux spectra from a data file in FERRE format (see the FERRE User Guide), and writes them to *specint* files. This tool accepts the following command-line arguments:

<ferre_file_name>

Name of input file.

<ferre_file_type>

Type of input file. This determines the mapping between atmospheric parameters given in the input file, and atmospheric parameters written to the output file. Supported options are: 'CAP18' (for the Allende Prieto et al., 2018 grids).

<specint_prefix>

Prefix of output files; *specint* files will have the name `<specint_prefix>-NNNNNNNN.h5`, wehre `NNNNNNNN` is the zero-padded index of the spectrum (starting at 1).

14.1.3 Goettingen

The **goettingen_to_specint** tool extracts a flux spectrum from a data file in FITS format (with the schema described by Husser et al., 2013), and writes it to a *specint* file. This tool accepts the following command-line arguments:

<fits_file_name>

Name of input file.

<wave_type>

Type of wavelength abscissa. This determines the number and distribution of points to assume for the input file. Supported options, corresponding to the different grids described by Husser et al. (2013), are: 'HiRes' (high-resolution), 'MedRes-A1' (medium-resolution, $\Delta\lambda = 1$) and 'MedRes-R10000' (medium resolution, $\mathcal{R} = 10\,000$). grids),

<specint_file_name>

Name of output file.

Note: In order for **goettingen_to_specint** to build, you must first uncomment/edit the line in `$MSG_DIR/build/Makefile` that defines the *FITS_LDFLAGS* variable. This variable defines the flags used to link against your system's FITS library.

14.2 Modifying Spectra

The spectra contained in *specint* files (as produced by one of the tools above) can be subsetted and/or rebinned using the **specint_to_specint** tool. This tool accepts the following command-line arguments:

<specint_file_name_in>
Name of input file.

<specint_file_name>
Name of output file.

lam_min=<value> (optional)
Subset to have a minimum wavelength of at least <*value*>.

lam_max=<value> (optional)
Subset to have a maximum wavelength of at most <*value*>.

R=<value> (optional)
Rebin to have a uniform resolution $\mathcal{R}$ of <*value*>.

dlam=<value> (optional)
Rebin to have a uniform wavelength spacing $\Delta\lambda$ of <*value*>.

just=<L|R> (optional)
Justify the new wavelength abscissa to the left ('L') or right ('R').

14.3 Creating Spectroscopic Grids

With a set of *specint* files extracted, a *specgrid* file can be created using the **specint_to_specgrid** tool. This tool accepts the following command-line arguments:

<manifest_file_name>
Name of input manifest file (see below).

<specgrid_file_name>
Name of output file.

<allow_dupes> (optional)
Flag governing handling of duplicate grid nodes in the manifest file; set to 'T' to allow duplicates.

The manifest file is a simple text file that lists all the *specint* files (one per line) that should be included in the grid.

14.4 Creating Passband Files

Additional passband files (beyond those already provided in the *Passband Files* appendix) can be created using the **make_passband** tool. This tool accepts the following command-line arguments:

<table_file_name>
Name of input file (see below).

<F_0>
Normalizing flux F_0 in $^{-2\ -1}$.

<passband_file_name>

Name of output file.

The input file is a text file tabulating wavelength λ (in) and passband response function $S'(\lambda)$ (see the *Evaluating Photometric Colors* section).

14.5 Creating Photometric Grids

Given a *specgrid* file, a corresponding *photgrid* file can be built using the *specgrid_to_photgrid* tool. This tool accepts the following command-line arguments:

<specgrid_file_name>

Name of input file.

<passband_file_name>

Name of passband file.

<photgrid_file_name>

Name of output file.

Note that it's not always necessary to create *photgrid* files, as MSG can convolve with passbands on the fly (as discussed in the *Evaluating Photometric Colors* section).

CHAPTER

FIFTEEN

TENSOR PRODUCT INTERPOLATION

Tensor product interpolation is a generalization of low-dimension (typically, 2-d or 3-d) multivariate interpolation schemes to an arbitrary number of dimensions. The literature on this topic is rather specialized (although see this Wikipedia section); therefore, this appendix provides a brief introduction. Interpolation in general is covered by many textbooks, so the focus here is reviewing how simple schemes can be generalized.

15.1 Univariate Interpolation

Suppose we are given a set of values $f(x_1), f(x_2), \ldots, f(x_n)$, representing some function $f(x)$ evaluated at a set of n discrete grid points. Then, a piecewise-linear interpolation scheme approximates $f(x)$ in each subinterval $x_i \le x \le x_{i+1}$ via

$$\tilde{f}(x) = f_i\,\ell_1(u) + f_{i+1}\,\ell_2(u)$$

Here $f_i \equiv f(x_i)$, while

$$u \equiv \frac{x - x_i}{x_{i+1} - x_i}$$

is a normalized coordinate that expresses where in the subinterval we are; $u = 0$ corresponds to $x = x_i$, and $u = 1$ to $x = x_{i+1}$. The linear basis functions are defined by

$$\ell_1(u) = 1 - u, \qquad \ell_2(u) = u.$$

This interpolation scheme is C^0 continuous, and reproduces $f(x)$ exactly at the grid points.

If we want a smoother representation of the function, we generally need more data. Suppose then that we're also provided the derivatives fx at the grid points. Then, a piecewise-cubic interpolation scheme is

$$\tilde{f}(x) = f_i\,c_1(u) + f_{i+1}\,c_2(u) + \partial_x f_i\,c_3(u) + \partial_x f_{i+1}\,c_4(u)$$

where u has the same definition as before, $\partial_x f_i \equiv (fx)_{x=x_i}$, and the cubic basis functions are

$$c_1(u) = 2u^3 - 3u^2 + 1, \quad c_2(u) = u^3 - u^2, \quad c_3(u) = -2u^3 + 3u^2, \quad c_4(u) = u^3 - 2u^2 + u$$

(these can be recognized as the basis functions for cubic Hermite splines). This new definition is C^1 continuous, and reproduces $f(x)$ *and* its first derivative exactly at the grid points.

15.2 Bivariate Interpolation

Bivariate interpolation allows us to approximate a function $f(x, y)$ from its values on a rectilinear (but not necessarily equally spaced) grid described by the axis values $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_m$. A piecewise-bilinear interpolating scheme approximates $f(x, y)$ in each subinterval $x_i \leq x \leq x_{i+1}, y_j \leq y \leq y_{j+1}$ via

$$\tilde{f}(x, y) = f_{i,j}\,\ell_1(u)\ell_1(v) + f_{i+1,j}\,\ell_2(u)\ell_2(v) + f_{i,j+1}\,\ell_1(u)\ell_1(v) + f_{i+1,j+1}\,\ell_2(u)\ell_2(v)$$

Here, u has the same definition as before, while

$$v \equiv \frac{y - y_j}{y_{j+1} - y_j},$$

and $f_{i,j} \equiv f(x_i, y_j)$. We can also write the scheme in the more-compact form

$$\tilde{f}(x, y) = \sum_{p,q=1}^{2} \mathcal{L}^{p,q}\,\ell_p(u)\ell_q(v),$$

where the coefficients $\mathcal{L}^{p,q}$ can be expressed as the matrix

$$\mathcal{L}^{p,q} \equiv \begin{bmatrix} f_{i,j} & f_{i,j+1} \\ f_{i+1,j} & f_{i+1,j+1} \end{bmatrix}.$$

A corresponding piecewise-bicubic interpolating scheme can be written in the form

$$\tilde{f}(x, y) = \sum_{p,q=1}^{4} \mathcal{C}^{p,q}\,c_p(u)c_q(v),$$

where the coefficients $\mathcal{C}^h$ can be expressed as the matrix

$$\mathcal{C}^{p,q} \equiv \begin{bmatrix} f_{i,j} & f_{i,j+1} & \partial_y f_{i,j} & \partial_y f_{i,j+1} \\ f_{i+1,j} & f_{i+1,j+1} & \partial_y f_{i+1,j} & \partial_y f_{i+1,j+1} \\ \partial_x f_{i,j} & \partial_x f_{i,j+1} & \partial_{xy} f_{i,j} & \partial_{xy} f_{i,j+1} \\ \partial_x f_{i+1,j} & \partial_x f_{i+1,j+1} & \partial_{xy} f_{i+1,j} & \partial_{xy} f_{i+1,j+1} \end{bmatrix}.$$

Constructing this matrix requires 16 values: the function at the four corners of the subinterval, the first derivatives with respect to x and with respect to y at the corners, and the cross derivatives $\partial_{xy} f$ at the corners.

15.3 Multivariate Interpolation

Let's now generalize the compact expressions above for piecewise-bilinear and bicubic interpolation, to piecewise interpolation in N dimensions. For linear interpolation, we have

$$\tilde{f}(x_1, x_2, \ldots, x_N) = \sum_{p_1,p_2,\ldots,p_N=1}^{2} \mathcal{L}^{p_1,p_2,\ldots,p_N} \prod_{k=1}^{N} \ell_{p_k}(u_k)$$

Likewise, for cubic interpolation, we have

$$\tilde{f}(x_1, x_2, \ldots, x_N) = \sum_{p_1,p_2,\ldots,p_N=1}^{4} \mathcal{C}^{p_1,p_2,\ldots,p_N} \prod_{k=1}^{N} c_{p_k}(u_k).$$

The coefficients $\mathcal{L}^{p_1,p_2,\ldots,p_N}$ and $\mathcal{C}^{p_1,p_2,\ldots,p_N}$ cannot easily be expressed in closed form, but they are relatively easy to construct algorithmically.

The summations in expressions above can be regarded as the contraction (over all indices) of a pair of rank-N tensors. In the cubic case, the components of the first tensor correspond to the coefficients $\mathcal{C}^{p_1,p_2,\ldots,p_N}$, while the second tensor is formed by taking N outer products between the vectors

$$\mathbf{c}_k(u_k) = \begin{bmatrix} c_1(u_k) \\ c_2(u_k) \\ c_3(u_k) \\ c_4(u_k) \end{bmatrix} \quad (k = 1, \ldots, N)$$

Hence, this kind of multivariate interpolation is also known as tensor product interpolation.

FORTRAN MODULE INDEX

INDEX

Symbols

A

C

D

E

F

G

I

J

L

M

P

R

S

T

U