slug Documentation

Release 2.0

Mark Krumholz, Michele Fumagalli, et al.

Nov 13, 2017

CONTENTS

Contents:

CHAPTER

ONE

LICENSE

SLUG is distributed under the terms of the GNU General Public License version 3.0. The text of the license is included in the main directory of the repository as `GPL-3.0.txt`.

CHAPTER

TWO

GETTING SLUG

SLUG is available at https://bitbucket.org/krumholz/slug2. The easiest way to download a copy is via git. If you have git, you can download SLUG by doing:

```
git clone https://krumholz@bitbucket.org/krumholz/slug2.git
```

In addition to the core SLUG code, the Bayesian inference tools cluster_slug and sfr_slug require large libraries of simulations on which to operate. These are not included in the git repository due to their sizes. You can download these from http://www.slugsps.com/data.

CHAPTER

THREE

INTRODUCTION TO SLUG

This is a guide for users of the SLUG software package. SLUG is distributed under the terms of the GNU General Public License v. 3.0. A copy of the license notification is included in the main SLUG directory. If you use SLUG in any published work, please cite the SLUG method papers, da Silva, R. L., Fumagalli, M., & Krumholz, M. R., 2012, The Astrophysical Journal, 745, 145 and Krumholz, M. R., Fumagalli, M., da Silva, R. L., Rendahl, T., & Parra, J. 2015, Monthly Notices of the Royal Astronomical Society, 452, 1447.

3.1 What Does SLUG Do?

SLUG is a stellar population synthesis (SPS) code, meaning that, for a specified stellar initial mass function (IMF), star formation history (SFH), cluster mass function (CMF), cluster lifetime function (CLF), and (optionally) distribution of extinctions (A_V), it predicts the spectra and photometry of both individual star clusters and the galaxies (or sub-regions of galaxies) that contain them. It also predicts the yields of various isotopes. In this regard, SLUG operates much like any other SPS code. The main difference is that SLUG regards the functions describing the stellar population as probability distributions, and the resulting stellar population as being the result of a draw from them. SLUG performs a Monte Carlo simulation to determine the PDF of the light and yields produced by the stellar populations that are drawn from these distributions. The remainder of this section briefly describes the major conceptual pieces of a SLUG simulation. For a more detailed description, readers are referred to da Silva, Fumagalli, & Krumholz (2012).

3.2 Cluster Simulations and Galaxy Simulations

SLUG can simulate either a simple stellar population (i.e., a group of stars all born at one time) or a composite stellar population, consisting of stars born at a distribution of times. We refer to the former case as a "cluster" simulation, and the latter as a "galaxy" simulation, since one can be thought of as approximating the behavior of a single star cluster, and the other as approximating a whole galaxy.

3.3 Probability Distribution Functions: the IMF, SFH, CMF, CLF, A_V distribution

As mentioned above, SLUG regards the IMF, SFH, CMF, CLF, and extinction A_V as probability distribution functions. These PDFs can be described by a very wide range of possible functional forms; see *Probability Distribution Functions* for details on the exact functional forms allowed, and on how they can be specified in the code. When SLUG runs a cluster simulation, it draws stars from the specified IMF in an attempt to produce a cluster of a user-specified total mass. There are a number of possible methods for performing such mass-limited sampling, and SLUG gives the user a wide menu of options; see *Probability Distribution Functions*. SLUG will also, upon user request, randomly draw a visual extinction A_V to be applied to the light (and either the same or a different visual extinction can be applied to nebular light – see *Nebular Processing*).

For a galaxy simulation, the procedure involves one extra step. In this case, SLUG assumes that some fraction f_c of the stars in the galaxy are born in star clusters, which, for the purposes of SLUG, means that they all share the same birth time. The remaining fraction $1 - f_c$ of stars are field stars. When a galaxy simulation is run, SLUG determines the total mass of stars M_* that should have formed since the start of the simulation (or since the last output, if more than one output is requested) from the star formation history, and then draws field stars and star clusters in an attempt to produce masses $(1 - f_c)M_*$ and $f_c M_*$. For the field stars, the stellar masses are drawn from the IMF, in a process completely analogous to the cluster case, and each star is given its own randomly-generated extinction. For star clusters, the masses of the clusters are drawn from the CMF, and each cluster is then populated from the IMF as in the cluster case. Again, each cluster gets its own extinction. For both the field stars and the star clusters, the time of their birth is drawn from the PDF describing the SFH.

Finally, star clusters can be disrupted independent of the fate of their parent stars. When each cluster is formed, it is assigned a lifetime drawn from the CLF. Once that time has passed, the cluster ceases to be entered in the lists of individual cluster spectra and photometry (see next section), although the individual stars continue to contribute to the integrated light of the galaxy.

3.4 Spectra and Photometry

Once SLUG has drawn a population of stars, its final step is to compute the light they produce. SLUG does this in several steps. First, it computes the physical properties of all the stars present user-specified times using a set of stellar evolutionary tracks. Second, it uses these physical properties to compute the composite spectra produced by the stars, using a user-specified set of stellar atmosphere models. Formally, the quantity computed is the specific luminosity per unit wavelength L_λ. Third, if nebular emission is enabled, the code calculates the spectrum $L_{\lambda,\mathrm{neb}}$ that emerges after the starlight passes through the HII region aruond the star – see *Nebular Processing*. Fourth, if extinction is enabled, SLUG computes the extincted stellar and nebula-processed spectra $L_{\lambda,\mathrm{ex}}$ and $L_{\lambda,\mathrm{neb,ex}}$ – see *Extinction*. Fifth and finally, SLUG computes photometry for the stellar population by integrating all computed spectra over a set of specified photometric filters. Depending on the options specified by the user and the filter under consideration, the photometric value output will be one of the following:

- The frequency-averaged luminosity across the filter, defined as

$$\langle L_\nu \rangle_R = \frac{\int L_\nu R_\nu \, d\ln\nu}{\int R_\nu (\nu/\nu_c)^\beta \, d\ln\nu},$$

where L_ν is the specific luminosity per unit frequency, R_ν is the filter response function per photon at frequency ν, ν_c is the central wavelength of the filter, and β is a constant that is defined by convention for each filter, and is either 0, 1, or 2; usually it is 0 for optical and UV filters.

- The wavelength-averaged luminosity across the filter, defined as

$$\langle L_\lambda \rangle_R = \frac{\int L_\lambda R_\lambda \, d\ln\lambda}{\int R_\lambda (\lambda/\lambda_c)^{-\beta} \, d\ln\lambda},$$

where L_λ is the specific luminosity per unit wavelength, R_λ is the filter response function per photon at wavelength λ, and λ_c is the central wavelength of the filter.

- The AB magnitude, defined by

$$M_{\mathrm{AB}} = -2.5 \log_{10} \left[\frac{\langle L_\nu \rangle_R}{4\pi \, (10\,\mathrm{pc})^2} \right] - 48.6,$$

where $\langle L_\nu \rangle_R$ is in units of $\mathrm{erg\,s^{-1}\,Hz^{-1}}$.

- The ST magnitude, defined by

$$M_{\mathrm{ST}} = -2.5 \log_{10} \left[\frac{\langle L_\lambda \rangle_R}{4\pi \, (10\,\mathrm{pc})^2} \right] - 21.1,$$

where $\langle L_\lambda \rangle_R$ is in units of $\mathrm{erg\,s^{-1}\,Angstrom^{-1}}$.

- The Vega magnitude, defined by

$$M_{\mathrm{Vega}} = M_{\mathrm{AB}} - M_{\mathrm{AB}}(\mathrm{Vega}),$$

where $M_{\mathrm{AB}}(\mathrm{Vega})$ is the AB magnitude of Vega. The latter quantity is computed on the fly, using a stored Kurucz model spectrum for Vega.

- The photon flux above some threshold ν_0, defined as

$$Q(\nu_0) = \int_{\nu_0}^{\infty} \frac{L_\nu}{h\nu}\, d\nu.$$

- The bolometric luminosity,

$$L_{\mathrm{bol}} = \int_0^{\infty} L_\nu\, d\nu.$$

If nebular processing and/or extinction are enabled, photometric quantities are computed separately for each available version of the spectrum, L_λ, $L_{\lambda,\mathrm{neb}}$, $L_{\lambda,\mathrm{ex}}$, and $L_{\lambda,\mathrm{neb,ex}}$.

For a cluster simulation, this procedure is applied to the star cluster being simulated at a user-specified set of output times. For a galaxy simulation, the procedure is much the same, but it can be done both for all the stars in the galaxy taken as a whole, and individually for each star cluster that is still present (i.e., that has not been disrupted).

3.5 Monte Carlo Simulation

The steps described in the previous two section are those required for a single realization of the stellar population. However, the entire point of SLUG is to repeat this procedure many times in order to build up the statistics of the population light output. Thus the entire procedure can be repeated as many times as the user desires.

3.6 Nebular Processing

SLUG includes methods for post-processing the output starlight to compute the light that will emerge from the HII region around star clusters, and to further apply extinction to that light.

Nebular emission is computed by assuming that, for stars / star clusters younger than 10 Myr, all the ionizing photons are absorbed in a uniform-density, uniform-temperature HII region around each star cluster / star, and then computing the resulting emission at non-ionizing energies. The calculation assumes that the HII region is in photoionization equilibrium, and consists of hydrogen that is fully ionized and helium that is singly ionized. Under these assumptions the volume V, electron density n_e, and hydrogen density n_{H} are related to the hydrogen ionizing luminosity $Q(\mathrm{H}^0)$ via

$$\phi Q(\mathrm{H}^0) = \alpha_{\mathrm{B}}(T) n_e n_{\mathrm{H}} V$$

Here ϕ is the fraction of ionizing photons that are absorbed by hydrogen within the observational aperture, and $\alpha_{\mathrm{B}}(T)$ is the temperature-dependent case B recombination rate coefficient. SLUG approximates $\alpha_{\mathrm{B}}(T)$ using the analytic approximation given by equation 14.6 of Draine (2011, Physics of the Interstellar and Intergalactic Medium, Princeton University Press). The temperature used to compute $\alpha_{\mathrm{B}}(T)$ can either be set by the user directly, or can be looked up automatically based on the age of the stellar population. The parameter ϕ must be chosen by the user. It encompasses two distinct effects, both of which serve to reduce nebular emission. First, not all ionizing photons will be absorbed by H; some will be absorbed by dust, and will not yield nebular emission. At Solar metallicity, this effect sets an upper limit $\phi \approx 0.73$ (see McKee & Williams (1997)). Second, some of the ionizing photons may travel far from the stars before being absorbed that the nebular emission they produce is not captured within the observational aperture. The importance of this effect obviously depends on the details of the observation.

The relation above determines $n_e n_{\rm H} V$, and from this SLUG computes the nebular emission including the following processes:

- H^+ and He^+ free-free emission
- H and He bound-free emission
- Hydrogen 2-photon emission
- Hydrogen recombination lines from all lines with upper levels $n_u \leq 25$
- Non-hydrogen line emission based on a tabulation (see below)

Formally, the luminosity per unit wavelength is computed as

$$L_{\lambda,\rm neb} = \left[\gamma_{\rm ff}^{(\rm H)} + \gamma_{\rm bf}^{(\rm H)} + \gamma_{\rm 2p}^{(\rm H)} + \sum_{n,n'\leq 25, n<n'} \alpha_{nn'}^{\rm eff,B,(H)} E_{nn'}^{(\rm H)} + x_{\rm He}\gamma_{\rm ff}^{(\rm He)} + x_{\rm He}\gamma_{\rm bf}^{(\rm He)} + \sum_i \gamma_{i,\rm line}^{(\rm M)}\right] n_e n_{\rm H} V$$

Here $n_e n_{\rm H} V = \phi_{\rm dust} Q({\rm H}^0)/\alpha_{\rm B}(T)$ from photoionization equilibrium, $E_{nn'}$ is the energy difference between hydrogen levels n and n', and the remaining terms and their sources appearing in this equation are:

- $\gamma_{\rm ff}^{(\rm H)}$ and $\gamma_{\rm ff}^{(\rm He)}$: HII and HeII free-free emission coefficients; these are computed from eqution 10.1 of Draine (2011), using the analytic approximation to the Gaunt factor given by equation 10.8 of the same source
- $\gamma_{\rm bf}^{(\rm H)}$ and $\gamma_{\rm bf}^{(\rm He)}$: HI and HeI bound-free emission coefficients; these are computed using the tabulation and interpolation method given in Ercolano & Storey (2006, MNRAS, 372, 1875)
- $\alpha_{nn'}^{\rm eff,B,(H)}$ is the effective emission rate coefficient for the n to n' H recombination line, taken from the tabulation of Storey & Hummer (1995, MNRAS, 272, 41)
- $\gamma_{i,\rm line}^{(\rm M)}$ is the emissivity for the brightest non-hydrogen lines, computed using a set of pre-tabulated values, following the procedure described in the SLUG 2 method paper
- $\gamma_{\rm 2p}^{(\rm H)}$: hydrogen two-photon emissivity, computed as

$$\gamma_{\rm 2p}^{(\rm H)} = \frac{hc}{\lambda^3} I({\rm H}^0) \alpha_{2s}^{\rm eff,(H)} \frac{1}{1 + \frac{n_{\rm H} q_{2s-2p,p} + (1+x_{\rm He}) n_{\rm H} q_{2s-2p,e}}{A_{2s-1s}}} P_\nu$$

Here

- $I({\rm H}^0)$ is the hydrogen ionization potential
- $\alpha_{2s}^{\rm eff,(H)}$ is the effective recombination rate to the 2s state, taken from the tabulation of Storey & Hummer (1995, MNRAS, 272, 41)
- $q_{2s-2p,p}$ and $q_{2s-2p,e}$ are the collisional rate coefficients for transitions from the 2s to the 2p state induced by collisions with protons and electrons, respectively, taken from Osterbrock (1989, University Science Books, table 4.10)
- A_{2s-1s} is the Einstein coefficient for the hydrogen 2s-1s two-photon emission process, taken from Draine (2011, section 14.2.4)
- P_ν is the frequency distribution for two-photon emission, computed from the analytic approximation of Nussbaumer & Schmutz (1984, A&A, 138, 495)

3.7 Extinction

If extinction is enabled, SLUG applies extinction to the stellar spectra and, if nebular processing is enabled as well, to the spectrum that emerges from the nebula.

SLUG computes the extincted spectrum as

$$L_{\lambda,\mathrm{ex}} = L_\lambda e^{-\tau_\lambda}$$

where the optical depth $\tau_\lambda = (\kappa_\lambda/\kappa_V)(A_V/1.086)$, A_V is the visual extinction in mag, the factor 1.086 is the conversion between magnitudes and the true dimensionless optical depth, κ_λ is a user-specified input extinction at wavelength λ, and the V-band mean opacity is defined by

$$\kappa_V = \frac{\int \kappa_\nu R_\nu(V)\, d\nu}{\int R_\nu(V)\, d\nu}$$

where $R_\nu(V)$ is the filter response function as frequency ν for the Johnson V filter. The extinction curve κ_λ can be specified via a user-provided file, or the user may select from a set of pre-defined extinction curves; see *Extinction Keywords* for details.

The computation for the extincted stellar plus nebular spectrum $L_{\lambda,\mathrm{neb,ex}}$ is analogous. SLUG allows the nebular and stellar emission to undergo different amounts of extinction, consistent with observational results indicating that nebular light is usually more extincted than the stellar continuum (Calzetti et al. (2000), Kreckel et al. (2013)). The total extincted stellar plus nebular spectrum is

$$L_{\lambda,\mathrm{neb,ex}} = L_\lambda e^{-\tau_\lambda} + L_{\lambda,\mathrm{neb}} e^{-\tau_\lambda f_{\mathrm{neb-ex}}}$$

where L_λ is the unextincted stellar spectrum, and $f_{\mathrm{neb-ex}}$ is the ratio of nebular to stellar extinction – typically about 2.1 based on observations, but left as a parameter to be set by the user.

3.8 Chemical Yields

In addition to computing the light output by a stellar population, SLUG can also predict the yield of isotopes. SLUG includes yields for core collapse supernovale and AGB stars. The core collapse supernova yields at present are for Solar metallicity only, and come from the yield tables provided by Sukhbold et al. (2016), which provide a finely-spaced set of yields for progenitors of mass 9 - 120 $M_\odot$. The AGB star yields come from Karakas & Lugaro (2016) or Doherty et al. (2014), depending on the progenitor mass; these yields are available at a range of metallicities. AGB and core collapse supernova yields can be turned on and off independently.

The yield tables are all slightly different with regard to what isotopes they include: the tables from Doherty et al. only include 37 stable isotopes up to the iron peak, the core collapse yield tables from Sukhbold et al. contain 302 isotopes, mostly stable but with some selected long-lived unstable ones, and the Karakas tables include an even larger number of stable and unstable isotopes with a wide range of lifetimes from seconds to Myr. It is up to the user how to handle combining the tables; they can be combined additively, so all isotopes are reported and yields are taken to be zero for isotopes that are missing from a given table, or disjunctively, so that only isotopes present in all yield tables are output. See *Parameter Specification* for details.

For unstable isotopes, the code correctly handles radioactive decay, i.e., if a certain amount of an unstable isotope with lifetime λ is produced at time t and none is produced thereafter, the amount reported at time $t + \Delta t$ will be smaller by a factor of $e^{-\Delta t/\lambda}$, and the mass of the daughter isotopes will have been increased accordingly. If desirable this behavior can be disabled, so that the yields reported are the total amounts produced, with no decay taken into account. See *Parameter Specification*.

There are a few caveats and limitations to the current approach, which may or may not be important depending the application:

- No yields from type Ia supernovae are presently included.
- The evolutionary tracks used to compute spectra and photometry do not precisely match the stellar evolution calculations used for the yields, so things like stellar lifetimes do not match up precisely.

- Because of the aforementioned issue, it is not possible to track the injection of yields properly in time over the course of a star's lifetime. Instead, SLUG simply assumes that all yields are produced instantaneously at the time when a star dies, with the lifetime taken from the tracks used for light output rather than from the yield calculations. The error is generally small, since the vast majority of the mass loss occurs in the last few centuries of a star's lifetime.
- When the star formation history is being treated non-stochastically (i.e., when the parameter `clust_frac` is less than 1.0, radioactive decay is not properly handled within integration time steps, so yields of unstable isotopes will be slightly off. The error can be minimized by writing more frequent outputs.
- The Doherty tables have use a different "Solar" metallicity scale than the later Karakas and Sukhbold ones. Solar metallicity corresponds to Z = 0.014 for the latter two, and Z = 0.02 for the former.
- When combining yields from multiple sources, neither the option of including all isotopes found in any table nor the option of including only the isotopes common to all tables gives precisely the correct answer. The former option (called `union` in the *Yield Keywords*) misses the contributions to some isotopes from some stars; for example, the yields of S process nuclei will be missing the contribution from stars from 8 - 9 $M_\odot$ (at Solar metallicity), because the Doherty tables do not include S process yields. The latter option (called `intersection` in the *Yield Keywords*) misses the mass contributed to longer-lived species by the decay of shorter-lived ones that are omitted because they are not in all tables.

CHAPTER

FOUR

COMPILING AND INSTALLING SLUG

4.1 Dependencies

The core SLUG program requires

- The Boost C++ libraries
- The GNU scientific library (version 2.x preferred, code can be compiled with version 1.x – see below)

In addition, the following are required for some functionality, but not for the core code:

- The cfitsio library (required for FITS capabilities)
- An implementation of MPI (required for MPI support; for full functionality, the implementation must support the MPI 3.0 or later standard)

Compilation will be easiest if you install the required libraries such that the header files are included in your `CXX_INCLUDE_PATH` (for Boost) and `C_INCLUDE_PATH` (for GSL, cfitsio, and MPI) and the compiled object files are in your `LIBRARY_PATH`. Alternately, you can manually specify the locations of these files by editing the Makefiles – see below. The cfitsio library is optional, and is only required if you want the ability to write FITS output. To compile without it, use the flag `FITS=DISABLE_FITS` when calling `make` (see below). The MPI libraries are required only for MPI capability, which is not enabled by default; see *Using SLUG as a Library with MPI-Enabled Codes* for an explanation of these capabilities and how to enable them. Note that SLUG uses some Boost libraries that must be built separately (see the Boost documentation on how to build and install Boost libraries).

In addition to the core dependencies, slugpy, the python helper library requires:

- numpy
- scipy
- astropy (optional, only required for FITS capabilities)

Finally, the cloudy coupling capability requires:

- cloudy

This is only required performing cloudy runs, and is not required for any other part of SLUG.

4.2 Compiling

If you have Boost in your `CXX_INCLUDE_PATH`, GSL in your `C_INCLUDE_PATH`, and (if you're using it) cfitsio in your `C_INCLUDE_PATH`, and the compiled libraries for each of these in your `LIBRARY_PATH` environment variables, and your system is running either MacOSX or Linux, you should be able to compile simply by doing:

```
make
```

from the main `slug` directory.

To compile in debug mode, do:

```
make debug
```

instead.

To enable MPI support, do:

```
make MPI=ENABLE_MPI
```

In addition, you may need to specify the names of your preferred MPI C++ compiler by setting the variable `MACH_MPICXX` in your machine-specific makefile – see *Machine-Specific Makefiles*. The Makefiles contain reasonable guesses, but since MPI compiler names are much less standardized than general compiler names, you may need to supply yours rather than relying on the default.

If you are compiling using GSL version 1.x or without cfitsio, you must specify these options when compiling. If you are using version 1.x of the GSL, do:

```
make GSLVERSION=1
```

To compile without FITS support, do:

```
make FITS=DISABLE_FITS
```

Note that SLUG is written in C++11, and requires some C++11 features, so it may not work with older C++ compilers. The following compiler versions are known to work: gcc >= 4.8 (4.7 works on most but not all platforms), clang/llvm >= 3.3, icc >= 14.0. Earlier versions may work as well, but no guarantees.

4.3 Machine-Specific Makefiles

You can manually specify the compiler flags to be used for you machine by creating a file named `Make.mach.MACHINE_NAME` in the `src` directory, and then doing:

```
make MACHINE=MACHINE_NAME
```

An example machine-specific file, `src/Make.mach.ucsc-hyades` is included in the repository. You can also override or reset any compilation flag you want by editing the file `src/Make.config.override`.

4.4 Note on Boost Naming and Linking Issues

The Boost libraries have a somewhat complex history of naming conventions (see this stackoverflow discussion thread). As a result, depending on your platform and where you got your Boost libraries and how you compiled them, the libraries names may or may not have names that end in `-mt` (indicating multithreading support). There is unfortunately no easy way to guess whether this tag will be present or not in the Boost installation on any particular system, so the `slug` makefiles contain defaults that are guesses based on some of the most common boost installations (e.g., the macports version of Boost has the `-mt` tag, so the default on Darwin is to include it). If you find that your attempted compilation fails at the linking stage with an error like:

```
ld: library not found for -lboost_system-mt
```

or:

```
ld: library not found for -lboost_system
```

but you are confident that you have boost installed and the path correctly set, you can try adding or removing the `-mt` flag. To do so, edit the file `src/Make.config.override` and add the line:

```
MACH_BOOST_TAG          = -mt
```

(to turn the `-mt` tag on) or:

```
MACH_BOOST_TAG          =
```

(to turn the `-mt` tag off). Then try compiling again.

CHAPTER

FIVE

RUNNING A SLUG SIMULATION

5.1 Basic Serial Runs

Once SLUG is compiled, running a simulation is extremely simple. The first step, which is not required but makes life a lot simpler, is to set the environment variable `SLUG_DIR` to the directory where you have installed SLUG. If you are using a `bash`-like shell, the syntax for this is:

```
export SLUG_DIR = /path/to/slug
```

while for a `csh`-like shell, it is:

```
setenv SLUG_DIR /path/to/slug
```

This is helpful because SLUG needs a lot of input data, and if you don't set this variable, you will have to manually specify where to find it.

Next, to run on a single processor, just do:

```
./bin/slug param/filename.param
```

where `filename.param` is the name of a parameter file, formatted as specified in *Parameter Specification*. The code will write a series of output files as described in *Output Files and Format*.

5.2 Thread-Based Parallelism

If you have more than one core at your disposal, you can also run SLUG in parallel using threads, via the command line:

```
python ./bin/slug.py param/filename.param
```

This called a python script that automatically divides up the Monte Carlo trials you have requested between the available processors, then consolidates the output so that it looks the same as if you had run a single-processor job. The python script allows fairly fine-grained control of the parallelism. It accepts the following command line arguments (not an exhaustive list – do `python ./bin/slug.py --help` for the full list):

- `-n NPROC, --nproc NPROC`: this parameter specifies the number of simultaneous SLUG processes to run. It defaults to the number of cores present on the machine where the code is running.
- `-b BATCHSIZE, --batchsize BATCHSIZE`: this specifies how to many trials to do per SLUG process. It defaults to the total number of trials requested divided by the total number of processes, rounded up, so that only one SLUG process is run per processor. *Rationale*: The default behavior is optimal from the standpoint of minimizing the overhead associated with reading data from disk, etc. However, if you are doing a very large

number of runs that are going to require hours, days, or weeks to complete, and you probably want the code to checkpoint along the way. In that case it is probably wise to set this to a value smaller than the default in order to force output to be dumped periodically.

- `-nc, --noconsolidate`: by default the `slug.py` script will take all the outputs produced by the parallel runs and consolidate them into single output files, matching what would have been produced had the code been run in serial mode. If set, this flag suppresses that behavior, and instead leaves the output as a series of files whose root names match the model name given in the parameter file, plus the extension `_pPPPPP_nNNNNN`, where the digits `PPPPP` give the number of the processor that produces that file, and the digits `NNNNN` give the run number on that processor. *Rationale*: normally consolidation is convenient. However, if the output is very large, this may produce undesirably bulky files. Furthermore, if one is doing a very large number of simulations over an extended period, and the `slug.py` script is going to be run multiple times (e.g., due to wall clock limits on a cluster), it may be preferable to leave the files unconsolidated until all runs have been completed.

5.3 MPI-Based Parallelism

SLUG can also run in parallel on distributed-memory architectures using MPI. To use MPI, you must first compile the code with MPI support – see *Compiling*. Then to start an MPI-parallel computation, do:

```
mpirun -np N bin/slug param/filename.param
```

where *N* is the number of parallel processes to run. In this mode each MPI process will write its own output files, which will be named as *MODELNAME_XXXX_FILETYPE.EXT* where *MODELNAME* is the model name specified in the parameter file (see *Parameter Specification*), *XXXX* is the process number of the process that wrote the file, *FILETYPE* is the type of output file (see *Output Files and Format*), and *EXT* is the extension specifying the file format (see *Output Files and Format*).

If it is desirable to do so, the output files produced by an MPI run can be combined into a single output file using the `consolidate.py` script in the `tools` subdirectory.

Note that full parallel computation is only available under MPI implementations that support the MPI 3.0 standard or later. Earlier versions of MPI allow MPI functionality for SLUG in library mode (see *Using SLUG as a Library*), but do not allow MPI parallel runs of the slug executable.

5.4 Checkpointing and Restarting

When running a large number of trials, it is often desirable to checkpoint the calculation, i.e., to write intermediate outputs rather than waiting until the entire calculation is done to write. SLUG can checkpoint after a specified number of trials; this number is controlled by the *checkpoint_interval* parameter (see *Parameter Specification*). Checkpoint files are are named as *MODELNAME_chkYYYY_FILETYPE.EXT* (or *MODELNAME_XXXX_chkYYYY_FILETYPE.EXT* for MPI runs) where *YYYY* is the number of the checkpoint, starting at 0. Checkpoints are valid output files with some added information – see *Checkpoint Files* for details.

To restart a run from checkpoints, just give the command line option *–restart*, for example:

```
mpirun -np N bin/slug param/filename.param --restart
```

SLUG will automatically search for checkpoint files (using the file names specified in *filename.param*), determine how many trials they contain, and resume the run to complete any remaining trials neede to reach the target number specified in the parameter file.

As with MPI runs, the output checkpoint files run can be combined into a single output file using the `consolidate.py` script in the `tools` subdirectory.

CHAPTER

SIX

PARAMETER SPECIFICATION

6.1 Automated Parameter File Generation

The remainder of this section contains information on how parameter files are formatted, and exactly how parameter choices specify code behavior. However, as a convenience slug comes with a python script that provides a simple menu-driven interface to write parameter files automatically. The script can be started by doing:

```
python tools/python/write_param.py
```

Once started, the script provides a series of menus that allow the user to set all the keywords specified below. The script can then write a validly-formatted parameter file based on the options chosen.

6.2 File Format

An example parameter file is included as `param/example.param` in the source tree. Parameter files for slug are generically formatted as a series of entries of the form:

```
keyword    value
```

Any line starting with # is considered to be a comment and is ignored, and anything on a line after a # is similarly treated as a comment and ignored. Some general rules on keywords are:

- Keywords may appear in any order.
- Many keywords have default values, indicated in parentheses in the list below. These keywords are optional and need not appear in the parameter file; missing keywords are set to their default values. Keywords that do not have a default are required.
- Keywords and values are case-insensitive.
- Unless explicitly stated otherwise, units for mass are always $M_\odot$, units for time are always yr, and units for extinction are always magnitudes.
- Any time a file or directory is specified, the path is resolved by the following rules:
  1. Absolute paths are always treated exactly as written.
  2. For relative paths, the code first checks for a file or directory of that name in the directory where slug is running. If no file of that name is found, and the environment variable `$SLUG_DIR` is set, teh code will search for the file at `$SLUG_DIR/user_specified_path`. This is to allow users to use default options that ship with the code simply by setting their `$SLUG_DIR` environment variable to wherever the code is installed.

3. Notwithstanding rule 2, the output directory (`out_dir`) is always assumed to be relative to the location where slug is running; it will never be prepended with `$SLUG_DIR`.

The keywords recognized by slug can be categorized as described in the remainder of this section.

6.3 Basic Keywords

These specify basic data for the run.

- `model_name` (default: `SLUG_DEF`): name of the model. This will become the base filename for the output files.
- `out_dir` (default: current working direcory): name of the directory into which output should be written. If not specified, output is written into the directory from which the slug executable is called.
- `verbosity` (default: `1`): level of verbosity when running, with 0 indicating no output, 1 indicating some output, and 2 indicating a great deal of output.

6.4 Simulation Control Keywords

These control the operation of the simulation.

- `sim_type` (default: `galaxy`): set to `galaxy` to run a galaxy simulation (a composite stellar population), or to `cluster` to run a cluster simulation (a simple stellar population)
- `n_trials` (default: `1`): number of trials to run
- `checkpoint_interval` (default: checkpointing off): output a checkpoint every `checkpoint_interval` trials
- `log_time` (default: `0`): set to 1 for logarithmic time step, 0 for linear time steps
- `time_step`: size of the time step. If `log_time` is set to 0, this is in yr. If `log_time` is set to 1, this is in dex (i.e., a value of 0.2 indicates that every 5 time steps correspond to a factor of 10 increase in time). Alternately, if `time_step` is set to any value that cannot be converted to a real number, then this is interpreted as giving the name of a PDF file, which must be formatted as described in *Probability Distribution Functions*. In this case one output time will be selected randomly for each trial from the specified PDF. This option is useful, for example, for generating a library of simulations that are randomly sampled in stellar population age. For the PDF option, the options `log_time`, `start_time` and `end_time` will all be ignored, as the relevant parameters will be taken from the specified PDF file. This keyword may be omitted, and will be ignored, if `output_times` is set.
- `start_time`: first output time. This may be omitted if `log_time` is set to 0, in which case it defaults to a value equal to `time_step`. It may also be omitted if `output_times` is set.
- `end_time`: last output time, in yr. This may be omitted of `output_times` is set. Note that not all the tracks include entries going out to times >1 Gyr, and the results will become inaccurate if the final time is larger than the tracks allow.
- `output_times`: an optional parameter giving an exact list of times in yr at which to write output. This must be specified as a comma-separated list, i.e., time0, time1, time2, ..., where all times are positive and are in strictly increasing order. There is no limit on the number of output times that may be included. This parameter may be omitted if `end_time` and `time_step` are set. If this parameter is set, it overrides `start_time`, `end_time`, and `time_step`.
- `sfr`: star formation rate. Only used if `sim_type` is `galaxy`; for `cluster`, it will be ignored, and can be omitted. This parameter can be set in three ways. If the value given is a number, it will be interpreted as

specifying a constant star formation rate. If it is the string `sfh`, the code will interpret this as a flag that a star formation history should be read from the file specified by the `sfh` keyword. If the parameter value is any other string that cannot be converted to a numerical value, it will be interpreted as the name of a PDF file (see *Probability Distribution Functions*); a (constant) value of the star formation rate for each trial will be drawn from this PDF.

- `sfh`: name of star formation history file. This file is a PDF file, formatted as described in *Probability Distribution Functions*. This is ignored, and can be omitted, if `sim_type` is `cluster`, or if `sfr` is not set to `sfh`.
- `cluster_mass`: mass of the star cluster for simulations with `sim_type` set to `cluster`. This can be omitted, and will be ignored, if `sim_type` is `galaxy`. This parameter can be set to either a positive number or to the string `cmf`. If it is set to a numerical value, that value will be used as the cluster mass, in $M_\odot$ for each trial. If it is set to `cmf`, then a new cluster mass will be drawn from the CMF for each trial.
- `redshift` (default: `0`): place the system at the specified redshift. The computed spectra and photometry will then be computed in the observed rather than the rest frame of the system.

6.5 Output Control Keywords

These control what quantities are computed and written to disk. Full a full description of the output files and how they are formatted, see *Output Files and Format*.

- `out_cluster` (default: `1`): write out the physical properties of star clusters? Set to 1 for yes, 0 for no.
- `out_cluster_phot` (default: `1`): write out the photometry of star clusters? Set to 1 for yes, 0 for no.
- `out_cluster_spec` (default: `1`): write out the spectra of star clusters? Set to 1 for yes, 0 for no.
- `out_cluster_yield` (default: `1`): write out the yield of star clusters? Set to 1 for yes, 0 for no.
- `out_integrated` (default: `1`): write out the integrated physical properties of the whole galaxy? Set to 1 for yes, 0 for no. This keyword is ignored if `sim_type` is `cluster`.
- `out_integrated_phot` (default: `1`): write out the integrated photometry of the entire galaxy? Set to 1 for yes, 0 for no. This keyword is ignored if `sim_type` is `cluster`.
- `out_integrated_spec` (default: `1`): write out the integrated spectra of the entire galaxy? Set to 1 for yes, 0 for no. This keyword is ignored if `sim_type` is `cluster`.
- `out_integrated_yield` (default: `1`): write out the integrated yield of the entire galaxy? Set to 1 for yes, 0 for no. This keyword is ignored if `sim_type` is `cluster`.
- `output_mode` (default: `ascii`): set to `ascii`, `binary`, or `fits`. Selecting `ascii` causes the output to be written in ASCII text, which is human-readable, but produces much larger files. Selecting `binary` causes the output to be written in raw binary. Selecting `fits` causes the output to be written FITS format. This will be somewhat larger than raw binary output, but the resulting files will be portable between machines, which the raw binary files are not guaranteed to be. All three output modes can be read by the python library, though with varying speed – ASCII output is slowest, FITS is intermediate, and binary is fastest.

6.6 Stellar Model Keywords

These specify the physical models to be used for stellar evolution, atmospheres, the IMF, extinction, etc.

- **`imf` (default: `lib/imf/chabrier.imf`): name of the IMF descriptor file; this is a PDF file, formatted as described in *I***

  - `chabrier.imf` (single-star IMF from Chabrier, 2005, in "The Initial Mass Function 50 Years Later", eds. E. Corbelli, F. Palla, & H. Zinnecker, Springer: Dordrecht, p. 41)
  - `chabrier03.imf` (single-star IMF from Chabrier, 2003, PASP, 115, 763-795)
  - `kroupa.imf` (IMF from Kroupa, 2002, Science, 295, 82-91)
  - `kroupa_sb99.imf` (simplified version of the Kroupa, 2002 IMF used by default by starburst99)
  - `salpeter.imf` (single-component power law IMF from Salpeter, 1955, ApJ, 121, 161)

- `cmf` (default: `lib/cmf/slug_default.cmf`): name of the CMF descriptor file; this is a PDF file, formatted as described in *Probability Distribution Functions*. The default selection is a power law $dN/dM \propto M^{-2}$ from $M = 10^2 - 10^7\ M_\odot$. This is ignored, and may be omitted, if `sim_type` is set to `cluster` and `cluster_mass` is set to a numerical value.
- `clf` (default: `lib/clf/slug_default.clf`): name of the CLF descriptor file; this is a PDF file, formatted as described in *Probability Distribution Functions*. The default gives a power law distribution of lifetimes t with $dN/dt \propto t^{-1.9}$ from 1 Myr to 1 Gyr. Note that this corresponds to a cluster age distribution of slope -0.9. The slug source also ships with an alternative CLF file, `lib/clf/nodisrupt.clf`, which disables cluster disruption entirely (by setting the lifetime distribution to a δ function at 10^{300} yr).
- **`tracks` (default: `geneva_2013_vvcrit_00`): stellar evolution tracks to use. This can be specified either by giving the**

  - `geneva_2013_vvcrit_00` and `geneva_2013_vvcrit_00`: Geneva (2013) track set, rotating at 0% and 40% of breakup, respectively. These tracks are available at metallicities of Solar and 1/7 Solar.
  - `geneva_mdot_std` and `geneva_mdot_enhanced`: pre-2013 Geneva track set, no rotation, with standard and 2 times standard mass loss rates, respectively. These models are available in metallicities of (relative to Solar) $Z = 0.05, 0.2, 0.4, 1.0, 2.0$.
  - `padova_tpagb_yes` and `padova_tpagb_no`: Padova tracks, no rotation, with and withouth thermally-pulsing AGB stars. These models are available in metallicities of (relative to Solar) $Z = 0.02, 0.2, 0.4, 1.0, 2.5$.
  - `mist_2016_vvcrit_00` and `mist_2016_vvcrit_40`: MIST v1.0 models, rotating and 0% and 40% of breakup, respectively; these models are available at Solar-normalised metallcities of $\log Z = -4$ to 0.5, in steps of 0.5 dex from $\log Z = -4$ to $\log Z = -2$, and 0.25 dex from $\log Z = -2$ to $\log Z = 0.5$.
  - `lib/tracks/sb99/ZXXXXvYY.txt`: individual files for Geneva (2013) tracks; metallicities are Solar (`XXXX = 0140`) and 1/7 Solar (`XXXX = 0020`), and rotation rates are 0 (`YY = 00`) and 40% of breakup (`YY = 40`).
  - `lib/tracks/sb99/modcXXX.dat`: individual files Geneva tracks with standard mass loss, for metallicities of $2\times$ Solar (`040`), Solar (`020`), $0.4\times$ Solar (`008`), $0.2\times$ Solar (`004`), and $0.05\times$ Solar (`001`).
  - `lib/tracks/sb99/modeXXX.dat`: same as `modcXXX.dat`, but twice standard mass loss rates.
  - `lib/tracks/sb99/modpXXX.dat`: individual files for Padova tracks with thermally pulsing AGB stars; metallicities use the same scale as `modcXXX.dat` files (i.e., `020` is Solar).
  - `lib/tracks/sb99/modsXXX.dat`: same as `modpXXX.dat`, but without thermally pulsing AGB stars
  - `lib/tracks/mist/vvcrit000/MIST_v1.0_feh_XXXXX_afe_p0.0_vvcrit0.0_EEPS.fits.gz`: individual files for MIST (2016, v1.0) non-rotating tracks. The `XXXXX` specifies the metallicity; the first letter is `p` or `m` for plus or minus, and the following letters give the

numerical value of the log metallicity in Solar-scaled units (e.g., `p0.00` is Solar, `m1.00` is 1/10 solar, `m2.00` is 1/100th Solar, etc.).

  - `lib/tracks/mist/vvcrit040//MIST_v1.0_feh_XXXXX_afe_p0.0_vvcrit0.4_EEPS.fits.gz`: same as `/MIST_v1.0_feh_XXXXX_afe_p0.0_vvcrit0.0_EEPS.fits.gz`, but rotating at 40% of breakup

- `atmospheres` (default: `lib/atmospheres`): directory where the stellar atmosphere library is located. Note that file names are hard-coded, so if you want to use different atmosphere models with a different format, you will have to write new source code to do so.
- **`specsyn_mode` (default: `sb99`): spectral synthesis mode. Allowed values are:**
  - `planck`: treat all stars as black bodies
  - `Kurucz`: use Kurucz atmospheres, as compiled by Lejeune et al. (1997, A&AS, 125, 229), for all stars
  - `Kurucz+Hillier`: use Kurucz atmospheres for all stars except Wolf-Rayet stars; WR stars use Hillier model atmospheres (Hillier & Miller, 1998, ApJ, 496, 407)
  - `Kurucz+Pauldrach`: use Kurucz atmospheres for all stars except OB stars; OB stars use Pauldrach model atmospheres (Pauldrach et al., 2001, A&A, 375, 161)
  - `SB99`: emulate the behavior of `starburst99`: use Pauldrach for OB stars, Hillier for WR stars, and Kurucz for all other stars
- `clust_frac` (default: `1.0`): fraction of stars formed in clusters
- `min_stoch_mass` (default: `0.0`): minimum stellar mass to be treated stochastically. All stars with masses below this value are assumed to be sampled continuously from the IMF.
- `metallicity` (default: `1.0`): metallicity of the stellar population, relative to Solar. If the tracks are specified by giving a track set, this value must be within the metallicity range covered by the chosen track set. If the tracks are set by specifying a particular track file, this keyword will be ignored in favor of the metallicity used for that track file, and a warning will be issued if it is set.

6.7 Extinction Keywords

- `A_V` (default: no extinction): extinction distribution. This parameter has three possible behaviors. If the parameter `A_V` is omitted entirely, then the code will not compute extinction-corrected spectra or photometry at all; only unextincted values will be reported. If this parameter is specified as a real number, it will be interepreted as specifying a uniform extinction value A_V, in mag, and this extinction will be applied to all predicted light output. Finally, if this parameter is a string that cannot be converted to a real number, it will be interpreted as the name of a PDF file, formatted as described in *Probability Distribution Functions*, specifying the probability distribution of A_V values, in mag.
- **`extinction_curve` (default: `lib/extinct/SB_ATT_SLUG.dat`) file specifying the extinction curve; the file format**
  - `LMC_EXT_SLUG.dat` : LMC extinction curve; optical-UV from Fitzpatrick, E. L., 1999, PASP, 111, 63, IR from Landini, M., et al., 1984, A&A, 134, 284; parts combined by D. Calzetti
  - `MW_EXT_SLUG.dat` : MW extinction curve; optical-UV from Fitzpatrick, E. L., 1999 PASP, 111, 63, IR from Landini, M., et al., 1984, A&A, 134, 284; parts combined by D. Calzetti
  - `SB_ATT_SLUG.dat` : "starburst" extinction curve from Calzetti, D., et al., 2000, ApJ, 533, 682
  - `SMC_EXT_SLUG.dat` : SMC extinction curve from Bouchet, P., et al., 1985, A&A, 149, 330

  - `MW_draine_RV3.1.dat` : MW extinction curve for reddening $R_V = 3.1$, taken from the model of Draine, 2003, ARA&A, 41, 241, and retrieved from B. Draine's personal web page

- `nebular_extinction_factor` (default: 1.0): nebular extinction excess factor. This parameter specifies the ratio of the extinction applied to the nebular light to that applied to the starlight, i.e., it gives $f_{\rm neb,ex} = A_{V,\rm neb}/A_{V,*}$, as defined in *Extinction*. As with `A_V`, this parameter can be set either to a real number, in which case this ratio is treated as constant and equal to the input number, or to the name of a PDF file that specified the distribution of this ratio, formatted as described in *Probability Distribution Functions*. If this keyword is omitted entirely, the nebular and stellar extinctions are set equal to one another.

6.8 Nebular Keywords

- `compute_nebular` (default: `1`): compute the spectrum that results after starlight is processed through the nebula surrounding each cluster or star? Set to 1 for yes, 0 for no.
- `atomic_data` (default: `lib/atomic/`): directory where the atomic data used for nebular emission calculations is located
- `nebular_no_metals` (default: 0): if set to 1, metal lines are not used when computing nebular emission
- `nebular_den` (default: `1e2`): hydrogen number density in $\rm cm^{-3}$ to use in nebular emission computations
- `nebular_temp` (default: `-1`): gas kinetic temperature in K to use in nebular emission computations; if set to non-positive value, the temperature will be determined via the lookup table of cloudy runs for fully sampled IMFs
- `nebular_logU` (default: `-3`): log of dimensionless volume-weighted ionization parameter to assume when computing metal line emission and HII region temperatures from the tabulated cloudy data. At present the allowed values are -3, -2.5, and -2.
- `nebular_phi` (default: `0.73`): fraction of ionizing photons absorbed by H atoms rather than being absorbed by dust grains or rescaping; the default value of `0.73`, taken from McKee & Williams (1997, ApJ, 476, 144) means that 73% of ionizing photons are absorbed by H

6.9 Photometric Filter Keywords

These describe the photometry to be computed. Note that none of these keywords have any effect unless `out_integrated_phot` or `out_cluster_phot` is set to 1.

- **`phot_bands`: photometric bands for which photometry is to be computed. The values listed here can be comma- or whit**
  - `QH0`: the $\rm H^0$ ionizing luminosity, in photons/sec
  - `QHe0`: the $\rm He^0$ ionizing luminosity, in photons/sec
  - `QHe1`: the $\rm He^+$ ionizing luminosity, in photons/sec
  - `Lbol`: the bolometric luminosity, in $L_\odot$
- `filters` (default: `lib/filters`): directory containing photometric filter data
- **`phot_mode` (default: `L_nu`): photometric system to be used when writing photometric outputs. Full definitions of the qu**
  - `L_nu`: report frequency-averaged luminosity in the band, in units of erg/s/Hz
  - `L_lambda`: report wavelength-averaged luminosity in the band, in units of erg/s/Angstrom

  - `AB`: report AB magnitude
  - `STMAG`: report ST magnitude
  - `VEGA`: report Vega magnitude

6.10 Yield Keywords

These keywords control the calculation of chemical yields. See *Chemical Yields* for explanations of the physical models corresponding to these choices.

- `yield_dir` (default: `lib/yields`): directory where the stellar yield tables are located. Note that the file name and format is hardcoded, so if you want to use a different format, you will have to write source code to do so.
- `yield_mode` (default: `sukhbold16+karakas16+doherty14`): sources for yields information. Valid options are:
  - `sukhbold16+karakas16+doherty14`: core collapse supernova yields from Sukhbold et al. (2016) plus AGB star yields from Karakas & Lugaro (2016) plus Doherty et al. (2014)
  - `karakas16+doherty14`: AGB star yields as in the first option, no core collapse supernova yields
  - `sukhbold16`: core collapse superonva yields as in the first option, no AGB star yields
- `no_decay_isotopes` (default: `0`): if set to a non-zero value, this option disables radioactive decay of unstable isotopes
- `isotopes_included` (default: `intersection`): controls how to handle isotopes that are present in some yield tables but not others. Valid options are:
  - `intersection`: only include isotopes present in all yield tables
  - `union`: include all isotopes found in any of the yield tables

CHAPTER

SEVEN

PROBABILITY DISTRIBUTION FUNCTIONS

The SLUG code regards the IMF, the CMF, the CLF, the SFH, and the extinction A_V as probability distribution functions – see *Probability Distribution Functions: the IMF, SFH, CMF, CLF, A_V distribution*. The code provides a generic file format through which PDFs can be specified. Examples can be found in the `lib/imf`, `lib/cmf`, `lib/clf`, and `lib/sfh` directories of the SLUG distribution.

PDFs in SLUG are generically written as functions

$$\frac{dp}{dx} = n_1 f_1(x; x_{1,a}, x_{1,b}) + n_2 f_2(x; x_{2,a}, x_{2,b}) + n_3 f_3(x; x_{3,a}, x_{3,b}) + \cdots ,$$

where $f_i(x; x_{i,a}, x_{i,b})$ is non-zero only for $x \in [x_{i,a}, x_{i,b}]$. The functions f_i are simple continuous functional forms, which we refer to as *segments*. Functions in this form can be specified in SLUG in two ways.

7.1 Basic Mode

The most common way of specifying a PDF is in basic mode. Basic mode describes a PDF that has the properties that

1. the segments are contiguous with one another, i.e., $x_{i,b} = x_{i+1,a}$
2. $n_i f_i(x_{i,b}; x_{i,a}, x_{i,b}) = n_{i+1} f_{i+1}(x_{i+1,a}; x_{i+1,a}, x_{i+1,b})$
3. the overall PDF is normalized such that $\int (dp/dx)\, dx = 1$

Given these constraints, the PDF can be specified fully simply by giving the x values that define the edges of the segments and the functional forms f of each segment; the normalizations can be computed from the constraint equations. Note that SFH PDFs cannot be described using basic mode, because they are not normalized to unity. Specifying a non-constant SFH requires advanced mode.

An example of a basic mode PDF file is as follows:

```
################################################################
# This is an IMF definition file for SLUG v2.
# This file defines the Chabrier (2005) IMF
################################################################

# Breakpoints: mass values where the functional form changes
# The first and last breakpoint will define the minimum and
# maximum mass
breakpoints 0.08 1 120

# Definitions of segments between the breakpoints

# This segment is a lognormal with a mean of log_10 (0.2 Msun)
# and dispersion 0.55; the dispersion is in log base 10, not
```

```
# log base e
segment
type lognormal
mean 0.2
disp 0.55

# This segment is a powerlaw of slope -2.35
segment
type powerlaw
slope -2.35
```

This example represents a Chabrier (2005) IMF from $0.08 - 120\ M_\odot$, which is of the functional form

$$\frac{dp}{dm} \propto \begin{cases} \exp[-\log(m/m_0)^2/(2\sigma^2)](m/m_b)^{-1}, & m < m_b \\ \exp[-\log(m_b/m_0)^2/(2\sigma^2)](m/m_b)^{-2.35}, & m \geq m_b \end{cases},$$

where $m_0 = 0.2\ M_\odot$, $\sigma = 0.55$, and $m_b = 1\ M_\odot$.

Formally, the format of a basic mode file is as follows. Any line beginning with # is a comment and is ignored. The first non-empty, non-comment line in a basic mode PDF file must be of the form:

```
breakpoints x1 x2 x3 ...
```

where `x1`, `x2`, `x3`, `...` are a non-decreasing series of real numbers. These represent the breakpoints that define the edges of the segment, in units of $M_\odot$. In the example given above, the breakpoints are are 0.08, 1, and 120, indicating that the first segment goes from $0.08 - 1\ M_\odot$, and the second from $1 - 120\ M_\odot$.

After the `breakpoints` line, there must be a series of entries of the form:

```
segment
type TYPE
key1 VAL1
key2 VAL2
...
```

where `TYPE` specifies what functional form describes the segment, and `key1 VAL1`, `key2 VAL2`, etc. are a series of (key, value) pairs the define the free parameters for that segment. In the example above, the first segment is described as having a `lognormal` functional form, and the keywords `mean` and `disp` specify that the lognormal has a mean of 0.2 $M_\odot$ and a dispersion of 0.55 in $\log_{10}$. The second segment is of type `powerlaw`, and it has a slope of -2.35. The full list of allowed segment types and the keywords that must be specified with them are listed in the *Segment Types* Table. Keywords and segment types are case-insensitive. Where more than one keyword is required, the order is arbitrary.

The total number of segments must be equal to one less than the number of breakpoints, so that each segment is described. Note that it is not necessary to specify a normalization for each segment, as the segments will be normalized relative to one another automatically so as to guarantee that the overall function is continuous.

Table 7.1: Segment Types

Name	Functional form	Key-word	Meaning	Key-word	Meaning
`delta`	$\delta(x - x_a)$				
`exponential`	$\exp(-x/x_*)$	`scale`	Scale length, x_*		
`lognormal`	$x^{-1} \exp\{-[\log_{10}(x/x_0)]^2/2\sigma^2\}$	`mean`	Mean, x_0	`disp`	Dispersion in $\log_{10}$, σ
`normal`	$\exp[-(x - x_0)^2/2\sigma^2]$	`mean`	Mean, x_0	`disp`	Dispersion, σ
`powerlaw`	x^p	`slope`	Slope, p		
`schechter`	$x^p \exp(-x/x_*)$	`slope`	Slope, p	`xstar`	Cutoff, x_*

7.2 Variable Mode

Variable Mode works as an extension to Basic Mode. Instead of assigning a value to a parameter, you can define a PDF and then draw values for the parameter from it.

Formally, the format of a Variable Mode PDF file follows that of a Basic Mode PDF file, but with one addition. To specify a parameter as variable, the entry must be of the form:

```
key1 V path/to/pdf.vpar
```

with the `V` instructing the code that the parameter is variable. The `.vpar` files are formatted as if they are standard Basic Mode PDF files. Variable Mode is an extension of Basic Mode, and it is not supported in Advanced Mode PDF files.

Any number of parameters belonging to a PDF can be made to vary, and the distributions from which their values are drawn can be constructed from any combination of the PDF segment types specified for Basic Mode.

An example of a Variable Mode PDF file for the IMF is as follows:

```
################################################################
# This is an IMF definition file for SLUG v2.
# This file defines a power law PDF with variable slope
################################################################

# Breakpoints: mass values where the functional form changes
# The first and last breakpoint will define the minimum and
# maximum mass
breakpoints 0.08 120

# Definitions of segments between the breakpoints

# This segment is a powerlaw with slopes drawn from slope_pdf
segment
type powerlaw
slope V lib/imf/slope_pdf.vpar
```

An example of a parameter's PDF file is as follows:

```
################################################################
# This is a parameter definition file for SLUG v2.
# lib/imf/slope_pdf.vpar
################################################################
```

```
# Breakpoints
breakpoints -3.0 -1.0

# Draw parameters from a normal distribution
segment
type normal
mean -2.35
disp 0.1
```

The above examples correspond to a powerlaw IMF with a slope varying between -3.0 and -1.0, with the value drawn from a normal distribution.

While the Variable Mode implementation is very general, it is currently only enabled for the IMF. The new parameter values are drawn at the start of each galaxy or cluster realisation.

7.3 Advanced Mode

In advanced mode, one has complete freedom to set all the parameters describing the PDF: the endpoints of each segment $x_{i,a}$ and $x_{i,b}$, the normalization of each segment n_i, and the functional forms of each segment f_i. This can be used to defined PDFs that are non-continuous, or that are overlapping; the latter option can be used to construct segments with nearly arbitrary functional forms, by constructing a Taylor series approximation to the desired functional form and then using a series of overlapping `powerlaw` segments to implement that series.

An example of an advanced mode PDF file is as follows:

```
################################################################
# This is a SFH definition file for SLUG v2.
# This defines a SF history consisting of a series of
# exponentially-decaying bursts with a period of 100 Myr and
# a decay timescale of 10 Myr, with an amplitude chosen to
# give a mean SFR of 10^-3 Msun/yr.
################################################################

# Declare that this is an advanced mode file
advanced

# First exponential burst
segment
type exponential
min      0.0
max      1.0e8          # Go to 100 Myr
weight   1.0e5          # Form 10^5 Msun of stars over 100 Myr
scale    1.0e7          # Decay time 10 Myr

# Next 4 bursts
segment
type exponential
min      1.0e8
max      2.0e8
weight   1.0e5
scale    1.0e7

segment
type exponential
min      2.0e8
max      3.0e8
```

```
weight   1.0e5
scale    1.0e7

segment
type exponential
min      3.0e8
max      4.0e8
weight   1.0e5
scale    1.0e7

segment
type exponential
min      4.0e8
max      5.0e8
weight   1.0e5
scale    1.0e7
```

This represents a star formation history that is a series of exponential bursts, separated by 100 Myr, with decay times of 10 Myr. Formally, this SFH follows the functional form

$$\dot{M}_* = ne^{-(t \bmod P)/t_{\rm dec}},$$

where $P = 100$ Myr is the period and $t_{\rm dec} = 10$ Myr is the decay time, from times $0 - 500$ Myr. The normalization constant n is set by the condition that $(1/P)\int_0^P \dot{M}_*\, dt = 0.001\ M_\odot\ {\rm yr}^{-1}$, i.e., that the mean SFR averaged over a single burst period is 0.001 $M_\odot$ yr^{-1}.

Formally, the format of an advanced mode file is as follows. First, all advanced mode files must start with the line:

```
advanced
```

to declare that the file is in advanced mode. After that, there must be a series of entries of the form:

```
segment
type TYPE
min MIN
max MAX
weight WEIGHT
key1 VAL1
key2 VAL2
...
```

The `type` keyword is exactly the same as in basic mode, as are the segment-specific parameter keywords `key1`, `key2`, ... The same functional forms, listed in the *Segment Types* Table, are available as in basic mode. The additional keywords that must be supplied in advanced mode are `min`, `max`, and `weight`. The `min` and `max` keywords give the upper and lower limits $x_{i,a}$ and $x_{i,b}$ for the segment; the probability is zero outside these limits. The keyword `weight` specifies the integral under the segment, i.e., the weight w_i given for segment i is used to set the normalization n_i via the equation

$$w_i = n_i \int_{x_{i,a}}^{x_{i,b}} f_i(x)\, dx.$$

In the case of a star formation history, as in the example above, the weight w_i of a segment is simply the total mass of stars formed in that segment. In the example given above, the first segment declaration sets up a PDF that with a minimum at 0 Myr, a maximum at 100 Myr, following an exponential functional form with a decay time of 10^7 yr. During this time, a total mass of $10^5\ M_\odot$ of stars is formed.

Note that, for the IMF, CMF, and CLF, the absolute values of the weights to not matter, only their relative values. On the other hand, for the SFH, the absolute weight does matter.

7.4 Sampling Methods

A final option allowed in both basic and advanced mode is a specification of the sampling method. The sampling method is a description of how to draw a population of objects from the PDF, when the population is specified as having a total sum $M_{\rm target}$ (usually but not necessarily a total mass) rather than a total number of members N; there are a number of ways to do this, which do not necessarily yield identical distributions, even for the same underlying PDF. To specify a sampling method, simply add the line:

```
method METHOD
```

to the PDF file. This line can appear anywhere except inside a `segment` specification, or before the `breakpoints` or `advanced` line that begins the file. The following values are allowed for `METHOD` (case-insensitive, as always):

- `stop_nearest`: this is the default option: draw until the total mass of the population exceeds $M_{\rm target}$. Either keep or exclude the final star drawn depending on which choice brings the total mass closer to the target value.
- `stop_before`: same as `stop_nearest`, but the final object drawn is always excluded.
- `stop_after`: same as `stop_nearest`, but the final object drawn is always kept.
- `stop_50`: same as `stop_nearest`, but keep or exclude the final object with 50% probability regardless of which choice gets closer to the target.
- `number`: draw exactly $N = M_{\rm target}/\langle M\rangle$ object, where $\langle M\rangle$ is the expectation value for a single draw.
- `poisson`: draw exactly N objects, where the value of N is chosen from a Poisson distribution with expectation value $\langle N\rangle = M_{\rm target}/\langle M\rangle$
- `sorted_sampling`: this method was introduced by Weidner & Kroupa (2006, MNRAS. 365, 1333), and proceeds in steps. One first draws exactly $N = M_{\rm target}/\langle M\rangle$ as in the `number` method. If the resulting total mass $M_{\rm pop}$ is less than $M_{\rm target}$, the procedure is repeated recursively using a target mass $M_{\rm target} - M_{\rm pop}$ until $M_{\rm pop} > M_{\rm target}$. Finally, one sorts the resulting stellar list from least to most massive, and then keeps or removes the final, most massive star using a `stop_nearest` policy.

See the file `lib/imf/wk06.imf` for an example of a PDF file with a `method` specification.

CHAPTER

EIGHT

OUTPUT FILES AND FORMAT

SLUG can produce 7 output files, though the actual number produced depends on the setting for the `out_*` keywords in the parameter file. (Additional output files can be produced by *cloudy_slug: An Automated Interface to cloudy*, and are documented in that section rather than here.)

The only file that is always produced is the summary file, which is named `MODEL_NAME_summary.txt`, where `MODEL_NAME` is the value given by the `model_name` keyword in the parameter file. This file contains some basic summary information for the run, and is always formatted as ASCII text regardless of the output format requested.

The other eight output files all have names of the form `MODEL_NAME_xxx.ext`, where the extension `.ext` is one of `.txt`, `.bin`, or `.fits` depending on the `output_mode` specified in the parameter file, and `xxx` is `integrated_prop`, `integrated_spec`, `integrated_phot`, `integrated_yield`, `cluster_prop`, `cluster_spec`, `cluster_phot`, or `cluster_yield`. The production of these output files is controlled by the parameters `out_integrated`, `out_integrated_spec`, `out_integrated_phot`, `out_integrated_yield`, `out_cluster`, `out_cluster_spec`, `out_cluster_phot`, and `out_cluster_yield` in the parameter file.

The easiest way to read these output files is with *slugpy – The Python Helper Library*, which can parse them and store the information in python structures. However, for users who wish to write their own parsers or examine the data directly, the format is documented below. The following conventions are used throughout, unless noted otherwise:

- Masses are in $M_\odot$
- Times in year
- Wavelengths are in Angstrom
- Specific luminosities are in erg/s/Angstrom
- For `binary` outputs, variable types refer to C++ types

8.1 The `integrated_prop` File

This file contains data on the bulk physical properties of the galaxy as a whole. It consists of a series of entries containing the following fields:

- `Trial`: which trial these data are from
- `Time`: evolution time at which the output is produced
- `TargetMass`: target mass of stars in the galaxy up that time, if the IMF and SFH were perfectly sampled
- `ActualMass`: actual mass of stars produced in the galaxy up to that time; generally not exactly equal to `TargetMass` due to finite sampling of the IMF and SFH
- `LiveMass`: current mass of all stars in the galaxy, accounting for the effects of stellar evolution (mass loss, supernovae); does not include the mass of stellar remnants (black holes, neutron stars, white dwarfs)

- `StellarMass`: same as `LiveMass`, but including the mass of stellar remnants; at present, remnant mass calculation is hard-coded to use the initial-final mass relation of Kruijssen (2009, A&A, 507, 1409)
- `ClusterMass`: current mass of all stars in the galaxy that are presently in clusters; same as `LiveMass`, but only including the stars in clusters
- `NumClusters`: number of non-disrupted clusters present in the galaxy at this time
- `NumDisClust`: number of disrupted clusters present in the galaxy at this time
- `NumFldStars`: number of field stars present in the galaxy at this time; this count only includes those stars being treated stochastically (see the parameter `min_stoch_mass` in *Stellar Model Keywords*)
- `VPx`: values drawn for variable parameter `x` (0,1,2 etc...); present only if SLUG was run with a variable mode IMF

If `output_mode` is `ascii`, these data are output in a series of columns, with different trials separated by lines of dashes. If `output_mode` is `fits`, the data are stored as a FITS binary table extension, with one column for each of the variables above, plus an additional column giving the trial number for that entry. Both the ASCII- and FITS-formatted output should be fairly self-documenting.

For `binary` output, the file consists of a series of records containing the following variables

- `Trial` (`unsigned int`)
- `Time` (`double`)
- `TargetMass` (`double`)
- `ActualMass` (`double`)
- `LiveMass` (`double`)
- `StellarMass` (`double`)
- `ClusterMass` (`double`)
- `NumClusters` (`std::vector<double>::size_type`, usually `unsigned long long`)
- `NumDisClust` (`std::vector<double>::size_type`, usually `unsigned long long`)
- `NumFldStars` (`std::vector<double>::size_type`, usually `unsigned long long`)
- `VPx` (`double`); present only if `nvps` is greater than 0, with one entry present for each variable parameter `x`

The first record in the binary file indicates the number of variable parameters used.

- `nvps` (`integer`): the number of variable parameters for the IMF.

There is one record of this form for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial.

8.2 The `integrated_spec` File

This file contains data on the spectra of the entire galaxy, and consists of a series of entries containing the following fields:

- `Trial`: which trial these data are from
- `Time`: evolution time at which the output is produced
- `Wavelength`: observed frame wavelength at which the stellar spectrum is evaluated
- `L_lambda`: specific luminosity at the specified wavelength, before extinction or nebular effects are applied

- `Wavelength_neb`: observed frame wavelength at which the stellar plus nebular spectrum is evaluated (present only if SLUG was run with nebular emission enabled)
- `L_lambda_neb`: specific luminosity at the specified wavelength, after the light has been processed through the nebula (only present if SLUG was run with nebular emission enabled)
- `Wavelength_ex`: observed frame wavelength at which the extincted stellar spectrum is evaluated (present only if SLUG was run with extinction enabled)
- `L_lambda_ex`: specific luminosity at the specified wavelength after extinction is applied, but without the effects of the nebula (only present if SLUG was run with extinction enabled)
- `Wavelength_neb_ex`: observed frame wavelength at which the extincted stellar plus nebular spectrum is evaluated (present only if SLUG was run with nebular processing and extinction enabled)
- `L_lambda_neb_ex`: specific luminosity at the specified wavelength, after the light is first processed by the nebular and then subjected to dust extinction (only present if SLUG was run with both extinction and nebular emission enabled)

If `output_mode` is `ascii`, these data are output in a series of columns, with different trials separated by lines of dashes. In `ascii` mode, only a single `Wavelength` column is present, and for those wavelengths that are not included in one of the grids, some entries may be blank.

If `output_mode` is `fits`, the output FITS file has two binary table extensions. The first table contains a field `Wavelength` listing the wavelengths at which the stellar spectra are given; if nebular emission was enabled in the SLUG calculation, there is also a field `Wavelength_neb` giving the nebular wavelength grid, and if extinction was enabled the table has a field `Wavelength_ex` listing the wavelengths at which the extincted spectrum is computed. If both nebular emission and extinction were included, the field `Wavelength_neb_ex` exists as well, giving the wavelength grid for that spectrum. The second table has three fields, `Trial`, `Time`, and `L_lambda` giving the trial number, time, and stellar spectrum. It may also contain fields `L_lambda_neb`, `L_lambda_ex`, and `L_lambda_neb_ex` giving the stellar plus nebular spectrum, extincted stellar spectrum, and extincted stellar plus nebular spectrum. Both the ASCII- and FITS-formatted output should be fairly self-documenting.

For binary output, the file is formatted as follows. The file starts with

- `Nebular` (`byte`): a single byte, with a value of 0 indicating that nebular processing was not enabled for this run, and a value of 1 indicating that it was enabled
- `Extinct` (`byte`): a single byte, with a value of 0 indicating that extinction was not enabled for this run, and a value of 1 indicating that it was enabled
- `NWavelength` (`std::vector<double>::size_type`, usually `unsigned long long`): the number of wavelength entries in the stellar spectra
- `Wavelength` (`NWavelength` entries of type `double`)
- `NWavelength_neb` (`std::vector<double>::size_type`, usually `unsigned long long`): the number of wavelength entries in the stellar plus nebular spectra; only present if `Nebular` is 1
- `Wavelength_neb` (`NWavelength_neb` entries of type `double`)
- `NWavelength_ex` (`std::vector<double>::size_type`, usually `unsigned long long`): the number of wavelength entries in the extincted spectra; only present if `Extinct` is 1
- `Wavelength_ex` (`NWavelength_ex` entries of type `double`); only present if `Extinct` is 1
- `NWavelength_neb_ex` (`std::vector<double>::size_type`, usually `unsigned long long`): the number of wavelength entries in the extincted nebular plus stellar spectra; only present if `Nebular` and `Extinct` are both 1
- `Wavelength_ex` (`NWavelength_neb_ex` entries of type `double`); only present if `Nebular` and `Extinct` are both 1

and then contains a series of records in the format

- `Trial` (`unsigned int`)
- `Time` (`double`)
- `L_lambda` (`NWavelength` entries of type `double`)
- `L_lambda_neb` (`NWavelength_neb` entries of type `double`); only present if `Nebular` is 1
- `L_lambda_ex` (`NWavelength_ex` entries of type `double`); only present if `Extinct` is 1
- `L_lambda_neb_ex` (`NWavelength_neb_ex` entries of type `double`); only present if `Nebular` and `Extinct` are both 1

There is one such record for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial.

8.3 The `integrated_phot` File

This file contains data on the photometric properties of the entire galaxy, and consists of a series of entries containing the following fields:

- `Trial`: which trial these data are from
- `Time`: evolution time at which the output is produced
- `PhotFilter1`: photometric value through filter 1, where filters follow the order in which they are specified by the `phot_bands` keyword; units depend on the value of `phot_mode` (see *Photometric Filter Keywords*)
- `PhotFilter2`
- `PhotFilter3`
- `...`
- `PhotFilter1_neb`: photometric value through filter 1 for the spectrum after nebular processing, in the same units as `PhotFilter1`; only present if SLUG was run with nebular processing enabled
- `PhotFilter2_neb`
- `PhotFilter3_neb`
- `...`
- `PhotFilter1_ex`: photometric value through filter 1 for the extincted spectrum, in the same units as `PhotFilter1`; only present if SLUG was run with extinction enabled
- `PhotFilter2_ex`
- `PhotFilter3_ex`
- `...`
- `PhotFilter1_neb_ex`: photometric value through filter 1 for the spectrum after nebular processing and extinction, in the same units as `PhotFilter1`; only present if SLUG was run with both nebular processing and extinction enabled
- `PhotFilter2_neb_ex`
- `PhotFilter3_neb_ex`
- `...`

If `output_mode` is `ascii`, these data are output in a series of columns, with different trials separated by lines of dashes. The columns for photometry of the extincted spectrum are present only if extinction was enabled when SLUG was run. Entries for some filters may be left blank. If so, this indicates that the photon response function provided for that filter extends beyond the wavelength range covered by the provided extinction curve. Since the extincted spectrum cannot be computed over the full range of the filter in this case, photometry for that filter cannot be computed either.

If `output_mode` is `fits`, the data are stored as a series of columns in a binary table extension to the FITS file; the filter names and units are included in the header information for the columns. If SLUG was run with nebular emission enabled, for each filter `FILTERNAME` there is a corresponding column `FILTERNAME_neb` giving the photometric value for the nebular-processed spectrum. Similarly, the columns `FILTERNAME_ex` and `FILTERNAME_neb_ex` give the photometric values for the extincted stellar and stellar + nebular spectra, respectively. Some of the extincted values may be `NaN`; this indicates that the photon response function provided for that filter extends beyond the wavelength range covered by the provided extinction curve. In addition to the time and photometric filter values, the FITS file contains a column specifying the trial number for that entry. Both the ASCII- and FITS-formatted output should be fairly self-documenting.

For binary output, the file is formatted as follows. The file starts with

- `NFilter` (stored as `ASCII text`): number of filters used
- `FilterName FilterUnit` (`NFilter` entries stored as `ASCII text`): the name and units for each filter are listed in ASCII, one filter-unit pair per line
- `Nebular` (`byte`): a single byte, with a value of 0 indicating that nebular processing was not enabled for this run, and a value of 1 indicating that it was enabled
- `Extinct` (`byte`): a single byte, with a value of 0 indicating that extinction was not enabled for this run, and a value of 1 indicating that it was enabled

This is followed by a series of entries of the form

- `Trial` (`unsigned int`)
- `Time` (`double`)
- `PhotFilter` (`NFilter` entries of type `double`)
- `PhotFilter_neb` (`NFilter` entries of type `double`); only present if `Nebular` is 1.
- `PhotFilter_ex` (`NFilter` entries of type `double`); only present if `Extinct` is 1. Note that some values may be `NaN` if photometry could not be computed for that filter (see above).
- `PhotFilter_neb_ex` (`NFilter` entries of type `double`); only present if `Nebular` and `Extinct` are both 1. Note that some values may be `NaN` if photometry could not be computed for that filter (see above).

There is one such record for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial.

8.4 The `integrated_yield` File

This file contains data on the integrated chemical yield of the entire galaxy, and consists of a series of entries containing the following fields:

- `Trial`: which trial these data are from
- `Time`: evolution time at which the output is produced
- `Name`: name (i.e., atomic symbol) of an isotope being produced
- `Z`: atomic number of an isotope being produced
- `A`: mass number of an isotope being produced

- `Yield`: mass of a particular isotope produced up to the specified time; for unstable isotopes, this includes the effects of radioactive decay, so yield can decrease with time under some circumstances

If `output_mode` is `ascii`, these data are output in a series of columns, with different trials separated by lines of dashes. If `output_mode` is `fits`, the data are stored as a series of columns in a binary table extension to the FITS file; the `Name`, `Z`, and `A` fields are placed in the first binary table extension, and are the same for every output. The `Time` and `Yields` fields are in the second binary table extension. In addition to these two fields, the second binary table contains a column specifying the trial number for each entry.

For binary output, the file is formatted as follows. It starts with

- `NIso` (`std::vector<double>::size_type`, usually `unsigned long long`): number of isotopes in the output

This is followed by `NIso` entries of the form

- `Name` (`char[4]`): isotope name (i.e., element symbol)
- `Z` (`unsigned int`)
- `A` (`unsigned int`)

The remainder of the file contains records of the from

- `Trial` (`unsigned int`)
- `Time` (`double`)
- `Yield` (`double[NIso]`)

There is one such record for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial.

8.5 The `cluster_prop` File

This file contains data on the bulk physical properties of the non-disrupted star clusters in the galaxy, with one entry per cluster per time at which that cluster exists. Each entry contains the following fields

- `UniqueID`: a unique identifier number for each cluster that is preserved across times and output files
- `Time`: evolution time at which the output is produced
- `FormTime`: time at which that cluster formed
- `Lifetime`: amount of time from birth to when the cluster will disrupt
- `TargetMass`: target mass of stars in the cluster, if the IMF were perfectly sampled
- `BirthMass`: actual mass of stars present in the cluster at formation
- `LiveMass`: current mass of all stars in the cluster, accounting for the effects of stellar evolution (mass loss, supernovae); does not include the mass of stellar remnants (black holes, neutron stars, white dwarfs)
- `StellarMass`: same as `LiveMass`, but including the mass of stellar remnants; at present, remnant mass calculation is hard-coded to use the initial-final mass relation of Kruijssen (2009, A&A, 507, 1409)
- `NumStar`: number of living stars in the cluster at this time; this count only includes those stars being treated stochastically (see the parameter `min_stoch_mass` in *Stellar Model Keywords*)
- `MaxStarMass`: mass of most massive star still living in the cluster; this only includes those stars being treated stochastically (see the parameter `min_stoch_mass` in *Stellar Model Keywords*)
- `A_V`: visual extinction for that cluster, in mag; present only if SLUG was run with extinction enabled

- VPx: values drawn for variable parameter x (0,1,2 etc. . .); present only if SLUG was run with a variable mode IMF

If output_mode is ascii, these data are output in a series of columns, with different trials separated by lines of dashes. If output_mode is fits, the data are stored as a FITS binary table extension, with one column for each of the variables above, plus an additional column giving the trial number for that entry. Both the ASCII- and FITS-formatted output should be fairly self-documenting.

For binary output, the first entry in the file is a header containing

- Extinct (byte): a single byte, with a value of 0 indicating that extinction was not enabled for this run, and a value of 1 indicating that it was enabled
- nvps (integer): the number of variable parameters for the IMF.

Thereafter, the file consists of a series of records, one for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial. Each record consists of a header containing

- Time (double)
- NCluster (std::vector<double>::size_type, usually unsigned long long): number of non-disrupted clusters present at this time

This is followed by NCluster entries of the following form:

- UniqueID (unsigned long)
- FormationTime (double)
- Lifetime (double)
- TargetMass (double)
- BirthMass (double)
- LiveMass (double)
- StellarMass (double)
- NumStar (std::vector<double>::size_type, usually unsigned long long)
- MaxStarMass (double)
- A_V (double); present only if Extinct is 1
- VPx (double); present only if nvps is greater than 0, with one entry present for each variable parameter x

8.6 The cluster_spec File

This file contains the spectra of the individual clusters, and each entry contains the following fields:

- UniqueID: a unique identifier number for each cluster that is preserved across times and output files
- Time: evolution time at which the output is produced
- Wavelength: observed frame wavelength at which the stellar spectrum is evaluated
- L_lambda: specific luminosity at the specified wavelength, before extinction or nebular effects are applied
- Wavelength_neb: observed frame wavelength at which the stellar plus nebular spectrum is evaluated (present only if SLUG was run with nebular emission enabled)
- L_lambda_neb: specific luminosity at the specified wavelength, after the light has been processed through the nebula (only present if SLUG was run with nebular emission enabled)

- `Wavelength_ex`: observed frame wavelength at which the extincted stellar spectrum is evaluated (present only if SLUG was run with extinction enabled)
- `L_lambda_ex`: specific luminosity at the specified wavelength after extinction is applied, but without the effects of the nebula (only present if SLUG was run with extinction enabled)
- `Wavelength_neb_ex`: observed frame wavelength at which the extincted stellar plus nebular spectrum is evaluated (present only if SLUG was run with nebular processing and extinction enabled)
- `L_lambda_neb_ex`: specific luminosity at the specified wavelength, after the light is first processed by the nebular and then subjected to dust extinction (only present if SLUG was run with both extinction and nebular emission enabled)

If `output_mode` is `ascii`, these data are output in a series of columns, with different trials separated by lines of dashes. The columns `L_lambda_neb`, `L_lambda_ex`, and `L_lambda_neb_ex` are present only if SLUG was run with the appropriate options enabled. Some entries in these fields may be empty; see *The integrated_spec File*.

If `output_mode` is `fits`, the output FITS file has two binary table extensions. The first table contains a field listing the wavelengths at which the spectra are given, in the same format as for *The integrated_spec File*. The second table has always contains the fields `UniqueId`, `Time`, `Trial`, and `L_lambda` giving the cluster unique ID, time, trial number, and stellar spectrum. Depending on whether nebular processing and/or extinction were enabled when SLUG was run, it may also contain the fields `L_lambda_neb`, `L_lambda_ex`, and `L_lambda_neb_ex` giving the nebular-processed, extincted, and nebular-processed plus extincted spectra. Both the ASCII- and FITS-formatted output should be fairly self-documenting.

Output in `binary` mode is formatted as follows. The file starts with

- `Nebular` (`byte`): a single byte, with a value of 0 indicating that nebular processing was not enabled for this run, and a value of 1 indicating that it was enabled
- `Extinct` (`byte`): a single byte, with a value of 0 indicating that extinction was not enabled for this run, and a value of 1 indicating that it was enabled
- `NWavelength` (`std::vector<double>::size_type`, usually `unsigned long long`): the number of wavelength entries in the stellar spectra
- `Wavelength` (`NWavelength` entries of type `double`)
- `NWavelength_neb` (`std::vector<double>::size_type`, usually `unsigned long long`): the number of wavelength entries in the stellar plus nebular spectra; only present if `Nebular` is 1
- `Wavelength_neb` (`NWavelength_neb` entries of type `double`)
- `NWavelength_ex` (`std::vector<double>::size_type`, usually `unsigned long long`): the number of wavelength entries in the extincted spectra; only present if `Extinct` is 1
- `Wavelength_ex` (`NWavelength_ex` entries of type `double`); only present if `Extinct` is 1
- `NWavelength_neb_ex` (`std::vector<double>::size_type`, usually `unsigned long long`): the number of wavelength entries in the extincted nebular plus stellar spectra; only present if `Nebular` and `Extinct` are both 1
- `Wavelength_ex` (`NWavelength_neb_ex` entries of type `double`); only present if `Nebular` and `Extinct` are both 1

and then contains a series of records, one for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial. Each record consists of a header containing

- `Time` (`double`)
- `NCluster` (`std::vector<double>::size_type`, usually `unsigned long long`): number of non-disrupted clusters present at this time

This is followed by `NCluster` entries of the following form:

- `UniqueID` (`unsigned long`)
- `L_lambda` (`NWavelength` entries of type `double`)
- `L_lambda_neb` (`NWavelength_neb` entries of type `double`); only present if `Nebular` is 1
- `L_lambda_ex` (`NWavelength_ex` entries of type `double`); only present if `Extinct` is 1
- `L_lambda_neb_ex` (`NWavelength_neb_ex` entries of type `double`); only present if `Nebular` and `Extinct` are both 1

8.7 The `cluster_phot` File

This file contains the photometric values for the individual clusters. Each entry contains the following fields:

- `UniqueID`: a unique identifier number for each cluster that is preserved across times and output files
- `Time`: evolution time at which the output is produced
- `PhotFilter1`: photometric value through filter 1, where filters follow the order in which they are specified by the `phot_bands` keyword; units depend on the value of `phot_mode` (see *Photometric Filter Keywords*)
- `PhotFilter2`
- `PhotFilter3`
- `...`
- `PhotFilter1_neb`: photometric value through filter 1 for the spectrum after nebular processing, in the same units as `PhotFilter1`; only present if SLUG was run with nebular processing enabled
- `PhotFilter2_neb`
- `PhotFilter3_neb`
- `...`
- `PhotFilter1_ex`: photometric value through filter 1 for the extincted spectrum, in the same units as `PhotFilter1`; only present if SLUG was run with extinction enabled
- `PhotFilter2_ex`
- `PhotFilter3_ex`
- `...`
- `PhotFilter1_neb_ex`: photometric value through filter 1 for the spectrum after nebular processing and extinction, in the same units as `PhotFilter1`; only present if SLUG was run with both nebular processing and extinction enabled
- `PhotFilter2_neb_ex`
- `PhotFilter3_neb_ex`
- `...`

If `output_mode` is `ascii`, these data are output in a series of columns, with different trials separated by lines of dashes. Some of the extincted photometry columns may be blank; see *The integrated_phot File*.

If `output_mode` is `fits`, the data are stored as a series of columns in a binary table extension to the FITS file; the filter names and units are included in the header information for the columns. If SLUG was run with nebular emission enabled, for each filter `FILTERNAME` there is a corresponding column `FILTERNAME_neb` giving the photometric value for the nebular-processed spectrum. Similarly, the columns `FILTERNAME_ex` and `FILTERNAME_neb_ex` give the photometric values for the extincted stellar and stellar + nebular spectra, respectively. Some of the extincted

values may be NaN; this indicates that the photon response function provided for that filter extends beyond the wavelength range covered by the provided extinction curve. In addition to the time and photometric filter values, the FITS file contains a column specifying the trial number for that entry. Both the ASCII- and FITS-formatted output should be fairly self-documenting.

In binary output mode, the binary data file starts with

- NFilter (stored as ASCII text): number of filters used
- FilterName FilterUnit (NFilter entries stored as ASCII text): the name and units for each filter are listed in ASCII, one filter-unit pair per line
- Nebular (byte): a single byte, with a value of 0 indicating that nebular processing was not enabled for this run, and a value of 1 indicating that it was enabled
- Extinct (byte): a single byte, with a value of 0 indicating that extinction was not enabled for this run, and a value of 1 indicating that it was enabled

and then contains a series of records, one for each output time , with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial. Each record consists of a header containing

- Time (double)
- NCluster (std::vector<double>::size_type, usually unsigned long long): number of non-disrupted clusters present at this time

This is followed by NCluster entries of the following form:

- UniqueID (unsigned long)
- PhotFilter (NFilter entries of type double)
- PhotFilter_neb (NFilter entries of type double); only present if Nebular is 1.
- PhotFilter_ex (NFilter entries of type double); only present if Extinct is 1. Note that some values may be NaN if photometry could not be computed for that filter (see above).
- PhotFilter_neb_ex (NFilter entries of type double); only present if Nebular and Extinct are both 1. Note that some values may be NaN if photometry could not be computed for that filter (see above).

8.8 The cluster_yield File

This file contains data on the chemical yield of individual star clusters, and consists of a series of entries containing the following fields:

- UniqueID: a unique identifier number for each cluster that is preserved across times and output files
- Time: evolution time at which the output is produced
- Name: name (i.e., atomic symbol) of an isotope being produced
- Z: atomic number of an isotope being produced
- A: mass number of an isotope being produced
- Yield: mass of a particular isotope produced up to the specified time; for unstable isotopes, this includes the effects of radioactive decay, so yield can decrease with time under some circumstances

If output_mode is ascii, these data are output in a series of columns, with different trials separated by lines of dashes. If output_mode is fits, the data are stored as a series of columns in a binary table extension to the FITS file; the Name, Z, and A fields are placed in the first binary table extension, and are the same for every output. The Time and Yields fields are in the second binary table extension. In addition to these two fields, the second binary table contains a column specifying the trial number for each entry.

For binary output, the file is formatted as follows. It starts with

- `NIso` (`std::vector<double>::size_type`, usually `unsigned long long`): number of isotopes in the output

This is followed by `NIso` entries of the form

- `Name` (`char[4]`): isotope name (i.e., element symbol)
- `Z` (`unsigned int`)
- `A` (`unsigned int`)

Thereafter, the file consists of a series of records, one for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial. Each record consists of a header containing

- `Time` (`double`)
- `NCluster` (`std::vector<double>::size_type`, usually `unsigned long long`): number of non-disrupted clusters present at this time

This is followed by `NCluster` entries of the following form:

- `UniqueID` (`unsigned long`)
- `Yield` (`double[NIso]`)

There is one such record for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial.

8.9 Checkpoint Files

Checkpoint files are identical to regular output files, except that they begin with a statement of the number of trials they contain. For ASCII files, this is indicated by a first line of the form `N_Trials = N`. For binary files, the file begins with an unsigned integer that gives the number of trials. For FITS files, the file has a keyword `N_Trials` in the first binary table HDU that gives the number of trials.

The slugpy library can read checkpoint files as well as regular output files (see *slugpy – The Python Helper Library*).

CHAPTER

NINE

FILTERS AND FILTER DATA

SLUG comes with a fairly extensive list of filters, adapted from the list maintained by Charlie Conroy as part of fsps. However, users may wish to add additional filters, and so the format of the filter list is documented here for convenience.

Filter data is stored in two ASCII text files, `FILTER_LIST` and `allfilters.dat`, which are stored in the `lib/filters` directory. The `FILTER_LIST` file is an index listing the available filters. In consists of five whitespace-separated columns. The first column is just an numerical index. The second is the name of the filter; this is the name that should be entered in the `phot_bands` keyword (see *Photometric Filter Keywords*) to request photometry in that filter. The third and fourth columns the value of β and λ_c (the central wavelength) for that filter – see *Spectra and Photometry* for definitions. Anything after the fourth column is regarded as a comment, and can be used freely for a description of that filter.

The `allfilters.dat` file contains the filter responses. The file contains a series of entires for different filters, each delineated by a header line that begins with `#`. The order in which filters appear in this file matches that in which they appear in the `FILTER_LIST`. After the header line, are a series of lines each containing two numbers. The first is the wavelength in Angstrom, and the second is the filter response function at that wavelength.

CHAPTER

TEN

SLUGPY – THE PYTHON HELPER LIBRARY

10.1 Installing slugpy

SLUG comes with the python module slugpy, which contains an extensive set of routines for reading, writing, and manipulating SLUG outputs. You can install slugpy one of two ways.

1. Using `Make`. If you compile the main slug code by doing `Make` in the main slug directory, the c slugpy extensions will be build automatically. Once that is done, you will be able to use slugpy just by importing it, provided that the slugpy directory is in your python import path.
2. Using `setup.py`. The slug distribution comes with a `setup.py` script that follows the standard python package convensions. Just do:

```
python setup.py build
```

to build the c extensions in place, which will let you import slugpy from the directory where it is located. Alternately, do:

```
python setup.py install
```

to install as a site package. Installing as a site package often requires root permissions, in which case you should do:

```
sudo python setup.py install
```

or:

```
python setup.py install --user
```

instead.

A note on compiling slugpy with setup.py: the slugpy c extensions require the GNU Scientific Library; to build or install slugpy, the appropriate headers must be in your default include path, and the appropriate libraries in your default link path. If they are not, you can tell setup where they are located by creating a file named `setup.cfg` in the slug2 directory, which contains the lines:

```
[build_ext]
include_dirs=/PATH/TO/GSL/HEADER
library_dirs=/PATH/TO/GSL/LIBRARIES
```

Then you should be able to build and install slugpy with `setup.py`.

10.2 Basic Usage

The most common task is to read a set of SLUG outputs into memory so that they can be processed. To read the data from a SLUG run using slugpy, one can simply do the following:

```
from slugpy import *
idata = read_integrated('SLUG_MODEL_NAME')
cdata = read_cluster('SLUG_MODEL_NAME')
```

The `read_integrated` function reads all the integrated-light data (i.e., the data stored in the `_integrated_*` files – see *Output Files and Format*) for a SLUG output whose name is given as the argument. This is the base name specified by the `model_name` keyword (see *Basic Keywords*), without any extensions; the slugpy library will automatically determine which outputs are available and in what format, and read the appropriate files. It returns a `namedtuple` containing all the output data available for that simulation. Note that some of these fields will only be present if the cloudy-slug interface (see *cloudy_slug: An Automated Interface to cloudy*) was used to process the SLUG output through cloudy to predict nebular emission, and some will be present only if extinction was enabled when SLUG was run. The fields returned are as follows:

- time: output times
- target_mass: target stellar mass at each time
- actual_mass: actual stellar mass at each time
- live_mass: mass of currently-alive stars
- cluster_mass: mass of living stars in non-disrupted clusters
- num_clusters: number of non-disrupted clusters
- num_dis_clusters: number of disrupted clusters
- num_fld_stars: number of still-living stars that formed in the field
- wl: wavelengths of output stellar spectra (in Angstrom)
- spec: integrated spectrum of all stars, expressed as a specific luminosity (erg/s/Angstrom)
- filter_names: list of photometric filter names
- filter_units: list of units for photometric outputs
- filter_wl_eff: effective wavelength for each photometric filter
- filter_wl: list of wavelengths for each filter at which the response function is given (in Angstrom)
- filter_response: photon response function for each filter at each wavelength (dimensionless)
- filter_beta: index β used to set the normalization for each filter – see *Spectra and Photometry*
- filter_wl_c: pivot wavelength used to set the normalization for each filter for which $\beta \neq 0$ – see *Spectra and Photometry*
- phot: photometry of the stars in each filter
- isotope_name: element symbols for the isotopes whose yields are reported
- isotope_Z: atomic numbers for the isotopes whose yields are reported
- isotope_A: atomic numbers for the isotopes whose yields are reported
- yld: yield of each isotope at each time

The following fields are present only if SLUG was run with nebular processing enabled:

- wl_neb: same as wl, but for the spectrum that emerges after the starlight has passed through the nebulae around the emitting clusters and field stars. The nebular grid is finer than the stellar grid, because it contains wavelength extra entries around prominent lines so that the lines are resolved on the grid
- spec_neb: same as spec, but for the nebular-processed spectrum
- phot_neb: same as phot, but for the nebular-processed spectrum

The following fields are present only if SLUG was run with extinction enabled:

- wl_ex: wavelengths of output stellar spectra after extinction has been applied(in Angstrom). Note that wl_ex will generally cover a smaller wavelength range than wl, because the extinction curve used may not cover the full wavelength range of the stellar spectra. Extincted spectra are computed only over the range covered by the extinction curve.
- spec_ex: same as spec, but for the extincted spectrum
- phot_ex: same as phot, but for the extincted spectrum. Note that some values may be `NaN`. This indicates that photometry of the extincted spectrum could not be computed for that filter, because the filter response curve extends to wavelengths outside the range covered by the extinction curve.

The following fields are present only if SLUG was run with both nebular processing and extinction enabled:

- wl_neb_ex: same as wl_neb, but for the extincted, nebular-processed spectrum. Will be limited to the same wavelength range as wl_ex.
- spec_neb_ex: same as spec_neb, but with extinction applied
- phot_neb_ex: same as phot_neb, but wtih extinction applied. Note that some values may be `NaN`. This indicates that photometry of the extincted spectrum could not be computed for that filter, because the filter response curve extends to wavelengths outside the range covered by the extinction curve.

The following fields are present only for runs that have been processed through the cloudy_slug interface (see *cloudy_slug: An Automated Interface to cloudy*):

- cloudy_wl: wavelengths of the output nebular spectra (in Angstrom)
- cloudy_inc: incident stellar radiation field, expressed as a specific luminosity (erg/s/Angstrom) – should be the same as spec, but binned onto cloudy's wavelength grid; provided mainly as a bug-checking diagnostic
- cloudy_trans: the transmitted stellar radiation field computed by cloudy, expressed as a specific luminosity (erg/s/Angstrom) – this is the radiation field of the stars after it has passed through the HII region, and is what one would see in an observational aperture centered on the stars with negligible contribution from the nebula
- cloudy_emit: the emitted nebular radiation field computed by cloudy, expressed as a specific luminosity (erg/s/Angstrom) – this is the radiation emitted by the nebula excluding the stars, and is what one would see in an observational aperture that included the nebula but masked out the stars
- cloudy_trans_emit: the sum of the transmitted stellar and emitted nebular radiation, expressed as a specific luminosity (erg/s/Angstrom) – this is what one would see in an observational aperture covering the both the stars and the nebula
- cloudy_linelabel: list of emitting species for the line luminosities computed by cloudy, following cloudy's 4-letter notation
- cloudy_linewl: wavelengths of all the lines computed by cloudy (in Angstrom)
- cloudy_linelum: luminosities of the lines computed by cloudy (in erg/s)
- cloudy_filter_names, cloudy_filter_units, cloudy_filter_wl_eff, cloudy_filter_wl, cloudy_filter_response, cloudy_filter_beta, cloudy_filter_wl_c: exactly the same as the corresponding fields without the cloudy prefix, but for the photometric filters applied to the cloudy output

- cloudy_phot_trans, cloudy_phot_emit, and cloudy_phot_trans_emit: photometry of the transmitted, emitted, and transmitted+emitted radiation field provided by cloudy_trans, cloudy_emit, and cloudy_trans_emit

For the above fields, quantities that are different for each trial and each time are stored as numpy arrays with a shape (N_times, N_trials) for scalar quantities (e.g., actual_mass), or a shape (N, N_times, N_trials) for quantities that are vectors of length N (e.g., the spectrum).

The `read_cluster` function is analogous, except that instead of reading the whole-galaxy data, it reads data on the individual star clusters, as stored in the `_cluster_*` output files. It returns the following fields:

- id: a unique identifier number for each cluster; this is guaranteed to be unique across both times and trials, so that if two clusters in the list have the same id number, that means that the data given are for the same cluster at two different times in its evolution
- trial: the trial number in which that cluster appeared
- time: the time at which the data for that cluster are computed
- form_time: the time at which that cluster formed
- lifetime: the between when the cluster formed and when it will disrupt
- target_mass: the target stellar mass of the cluster
- actual_mass: the actual stellar mass of the cluter
- live_mass: the mass of all still-living stars in the cluster
- num_star: the number of stars in the cluster
- max_star_mass: the mass of the single most massive still-living star in the cluster
- A_V: the visual extinction for this cluster, in mag; present only if SLUG was run with extinction enabled
- All the remaining fields are identical to those listed above for integrated quantities, starting with wl

For all these fields, scalar quantities that are different for each cluster (e.g., actual_mass) will be stored as arrays of shape (N_cluster); vector quantities that are different for each cluster (e.g., spec) will be stored as arrays of shape (N_cluster, N).

The following fields are present only if SLUG was run with a Variable Mode IMF:

- VPx: The value drawn for variable parameter x (0,1,2...) in each trial. The parameters are numbered in the order they are defined in the IMF definition file.

These fields are present in both the cluster and integrated outputs if a simulation has been run using the variable mode IMF.

10.3 Full Documentation of slugpy

`slugpy.`**`combine_cluster`**(*data*)

Function to combine cluster data from multiple SLUG2 runs, treating each input run as a separate set of trials. Trial and cluster unique ID numbers are altered as necessary to avoid duplication between the merged data sets.

Parameters:

data [list_like] A list containing the cluster data for each run, as returned by read_cluster

Returns:

combined_data [namedtuple] The combined data, in the same format as each object in data

slugpy.**combine_integrated**(*data*)

Function to combine integrated data from multiple SLUG2 runs, treating each input run as a separate set of trials.

Parameters

data [list_like] A list containing the integrated data for each run, as returned by read_integrated

Returns

combined_data [namedtuple] The combined data, in the same format as each object in data

slugpy.**compute_photometry**(*wl*, *spec*, *filtername*, *photsystem='L_nu'*, *filter_wl=None*, *filter_response=None*, *filter_beta=None*, *filter_wl_c=None*, *filter_dir=None*)

This function takes an input spectrum and a set of response functions for photometric filters, and returns the photometry through those filters.

Parameters

wl [array] Wavelength of input spectrum in Angstrom

spec [array] Specific luminosity per unit wavelength for input spectrum, in erg/s/A

filtername [string or iterable of strings] Name or list of names of the filters to be used. Filter names can also include the special filters Lbol, QH0, QHe0, and QHe1; the values returned for these will be the bolometric luminosity (in erg/s) and the photon luminosities (in photons/s) in the H, He, and He+ ionizing-continua, respectively.

photsystem [string] The photometric system to use for the output. Allowable values are 'L_nu', 'L_lambda', 'AB', 'STMAG', and 'Vega', corresponding to the options defined in the SLUG code.

filter_wl [array or iterable of arrays] Array giving the wavelengths in Angstrom at which the filter is response function is given. If this object is an iterable of arrays rather than a single array, it is assumed to represent the wavelengths for a set of filters. If this is set, no data is read from disk. Default behavior is to read the filter information from disk.

filter_response [array or iterable of arrays] Array giving the filter response function at each wavelenght and for each filter in filter_wl. Must be set if filter_wl is set, ignored otherwise.

filter_beta [iterable] Array-like object containing the index beta for each filter. Must be set if filter_wl is set, ignored otherwise.

filter_wl_c [iterable] Array-like object containing the pivot wavelength for each filter. Must be set if filter_wl is set, ignored otherwise.

filter_dir [string] Directory where the filter data files can be found. If left as None, filters will be looked for in the $SLUG_DIR/lib/filters directory. This parameter is used only if filtername is not None.

Returns

phot [array] Photometric values in the requested filters. Units depend on the choice of photometric system: L_nu –> erg/s/Hz; L_lambda –> erg/s/A; AB –> absolute AB magnitude; STMAG –> absolute ST magnitude; Vega –> absolute Vega magnitude;

slugpy.**photometry_convert**(*photsystem*, *phot*, *units*, *wl_cen=None*, *filter_last=False*, *filter_names=None*, *filter_dir=None*)

Function to convert photometric data between photometric systems.

Parameters

photsystem [string] The photometric system to which to convert. Allowable values are 'L_nu', 'L_lambda', 'AB', 'STMAG', and 'Vega', corresponding to the options defined in the SLUG code. If

this is set and the conversion requested involves a conversion from a wavelength-based system to a frequency-based one, wl_cen must not be None.

phot [array] array of photometric data; if the array has more than one dimension, the first dimension is assumed to represent the different photometric filters (unless filter_last is True, in which case the last dimension is represents the array of filters)

units [iterable of strings] iterable listing the units of the input photometric data. On return, strings will be changed to the units of the new system.

wl_cen [array] central wavelengths of the filters, in Angstrom; can be left as None if the requested conversion doesn't require going between wavelength- and frequency-based systems.

filter_last [bool] If the input data have more than one dimension, by default it is assumed that the first dimension contains values for the different photometric filters. If this keyword is set to True, it will instead be assumed that the last dimension contains the values for the different filters.

filter_names [iterable of strings] Names of all filters, used to read the filter response functions from disk; only needed for conversions to and from Vega magnitudes, and ignored otherwise

filter_dir [string] Directory where the filter data files can be found. If left as None, filters will be looked for in the $SLUG_DIR/lib/filters directory. This parameter is used only for conversions to and from Vega magnitudes.

Returns Nothing

Raises ValueError, if wl_cen is None but the requested conversion requires going between wavelength- and frequency-based systems

`slugpy.`**`read_cluster`**(*model_name*, *output_dir=None*, *fmt=None*, *nofilterdata=False*, *photsystem=None*, *verbose=False*, *read_filters=None*, *read_nebular=None*, *read_extinct=None*, *read_info=None*, *read_lines=None*, *no_stellar_mass=False*)

Function to read all cluster data for a SLUG2 run.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt [string] Format for the file to be read. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'. If one of these is set, the code will only attempt to open ASCII-, binary-, or FITS-formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

nofilterdata [bool] If True, the routine does not attempt to read the filter response data from the standard location

photsystem [None or string] If photsystem is None, the data will be returned in the same photometric system in which they were read. Alternately, if it is a string, the data will be converted to the specified photometric system. Allowable values are 'L_nu', 'L_lambda', 'AB', 'STMAG', and 'Vega', corresponding to the options defined in the SLUG code. If this is set and the conversion requested involves a conversion from a wavelength-based system to a frequency-based one, nofilterdata must be False so that the central wavelength of the photometric filters is available.

verbose [bool] If True, verbose output is printed as code runs

read_filters [None | string | listlike containing strings] If this is None, photometric data on all filters is read. Otherwise only filters whose name(s) match the input filter names ar read.

read_nebular [None | bool] If True, only photometric data with the nebular contribution is read; if False, only data without it is read. Default behavior is to read all data.

read_extinct [None | bool] If True, only photometric data with extinction applied is read; if False, only data without it is read. Default behavior is to read all data.

read_info [dict] On return, this dict will contain the keys 'prop_name', 'phot_name', 'spec_name', 'cloudyspec_name', 'cloudylines_name' and 'format', giving the names of the files read and the format they were in; 'format' will be one of 'ascii', 'binary', or 'fits'. If one of the files is not present, the corresponding _name key will be omitted from the dict.

no_stellar_mass [bool] Prior to 7/15, output files did not contain the stellar_mass field; this can be detected automatically for ASCII and FITS formats, but not for binary format; if True, this specifies that the binary file being read does not contain a stellar_mass field; it has no effect for ASCII or FITS files

no_neb_extinct [bool] Prior to 2/17, SLUG did not support differential nebular extinction, and thus there was no output field for it; this is detected and handled automatically for ASCII and FITS files; for binary outputs, this flag must be set for pre 2/17 output files to be read correctly

read_lines [None | string | listlike containing strings] If this is None, data for all the available lines will be read. Default is to read all data.

Returns A namedtuple containing the following fields:

(Always present)

id [array, dtype uint] unique ID of cluster

trial: array, dtype uint which trial was this cluster part of

time [array] time at which cluster's properties are being evaluated

A_V [array] A_V value for each cluster, in mag (present only if SLUG was run with extinction enabled)

A_Vneb [array] value of A_V applied to the nebular light for each cluster (present only if SLUG was run with both nebular emission and extinction enabled)

(Present if the run being read contains a cluster_prop file)

form_time [array] time when cluster formed

lifetime [array] time at which cluster will disrupt

target_mass [array] target cluster mass

actual_mass [array] actual mass at formation

live_mass [array] mass of currently living stars

stellar_mass [array] mass of all stars, living and stellar remnants

num_star [array, dtype ulonglong] number of living stars in cluster being treated stochastically

max_star_mass [array] mass of most massive living star in cluster

vpn_tuple [tuple] tuple containing arrays for any variable parameters we have (eg: VP0, VP1,VP2...) in the IMF. Each element of the tuple is an array. Present only if variable parameters were enables when SLUG was run.

(Present if the run being read contains a cluster_spec file)

wl [array] wavelength, in Angstrom

spec [array, shape (N_cluster, N_wavelength)] specific luminosity of each cluster at each wavelength, in erg/s/A

wl_neb [array] wavelength for the nebular spectrum, in Angstrom (present only if SLUG was run with nebular emission enabled)

spec_neb [array, shape (N_cluster, N_wavelength)] specific luminosity at each wavelength and each time for each trial, including emission and absorption by the HII region, in erg/s/A (present only if SLUG was run with nebular emission enabled)

wl_ex [array] wavelength for the extincted spectrum, in Angstrom (present only if SLUG was run with extinction enabled)

spec_ex [array, shape (N_cluster, N_wavelength)] specific luminosity at each wavelength in wl_ex and each time for each trial after extinction has been applied, in erg/s/A (present only if SLUG was run with extinction enabled)

wl_neb_ex [array] wavelength for the extincted spectrum with nebular emission, in Angstrom (present only if SLUG was run with both nebular emission and extinction enabled)

spec_neb_ex [array, shape (N_cluster, N_wavelength)] specific luminosity at each wavelength in wl_ex and each time for each trial including emission and absorption by the HII region, after extinction has been applied, in erg/s/A (present only if SLUG was run with nebular emission and extinction both enabled)

(Present if the run being read contains a cluster_phot file)

filter_names [list of string] a list giving the name for each filter

filter_units [list of string] a list giving the units for each filter

filter_wl_cen [list] central wavelength of each filter; this is set to None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

filter_wl [list of arrays] a list giving the wavelength table for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

filter_response [list of arrays] a list giving the photon response function for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

phot [array, shape (N_cluster, N_filter)] photometric value in each filter for each cluster; units are as indicated in the units field

If extinction is enabled, phot_ex will contain photometry after extinction has been applied.

(Present if the run being read contains a cluster_yield file)

isotope_name [array of strings] Atomic symbols of the isotopes included in the yield table

isotope_Z [array of int] Atomic numbers of the isotopes included in the yield table

isotope_A [array of int] Atomic mass number of the isotopes included in the yield table

yld [array] Yield of each isotope, defined as the instantaneous amount produced up to that time; for unstable isotopes, this includes the effects of decay since production

(Present if the run being read contains a cluster_cloudyspec file)

cloudy_wl [array] wavelength, in Angstrom

cloudy_inc [array, shape (N_cluster, N_wavelength)] specific luminosity of the cluster's stellar radiation field at each wavelength, in erg/s/A

cloudy_trans [array, shape (N_cluster, N_wavelength)] specific luminosity of the stellar radiation field after it has passed through the HII region, at each wavelength, in erg/s/A

cloudy_emit [array, shape (N_cluster, N_wavelength)] specific luminosity of the radiation field emitted by the HII region, at each wavelength, in erg/s/A

cloudy_trans_emit [array, shape (N_cluster, N_wavelength)] the sum of the emitted and transmitted fields; this is what would be seen by an observer looking at both the star cluster and its nebula

(Present if the run being read contains a cluster_cloudylines file)

cloudy_linelabel [array, dtype='S4', shape (N_lines)] labels for the lines, following cloudy's 4 character line label notation

cloudy_linewl [array, shape (N_lines)] rest wavelength for each line, in Angstrom

cloudy_linelum [array, shape (N_cluster, N_lines)] luminosity of each line at each time for each trial, in erg/s

(Present if the run being read contains a cluster_cloudyphot file)

cloudy_filter_names [list of string] a list giving the name for each filter

cloudy_filter_units [list of string] a list giving the units for each filter

cloudy_filter_wl_eff [list] effective wavelength of each filter; this is set to None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

cloudy_filter_wl [list of arrays] a list giving the wavelength table for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

cloudy_filter_response [list of arrays] a list giving the photon response function for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

cloudy_filter_beta [list] powerlaw index beta for each filter; used to normalize the photometry

cloudy_filter_wl_c [list] pivot wavelength for each filter; used to normalize the photometry

cloudy_phot_trans [array, shape (N_cluster, N_filter)] photometric value for each cluster in each filter for the transmitted light (i.e., the starlight remaining after it has passed through the HII region); units are as indicated in the units field

cloudy_phot_emit [array, shape (N_cluster, N_filter)] photometric value for each cluster in each filter for the emitted light (i.e., the diffuse light emitted by the HII region); units are as indicated in the units field

cloudy_phot_trans_emit [array, shape (N_cluster, N_filter)] photometric value in each filter for each cluster for the transmitted plus emitted light (i.e., the light coming directly from the stars after absorption by the HII region, plus the diffuse light emitted by the HII region); units are as indicated in the units field

(Present if the run being read contains a cluster_cloudyparams file)

cloudy_hden [array] number density of H nuclei at the inner edge of the ionized region simulated by cloudy

cloudy_r0 [array] inner radius of the ionized region simulated by cloudy

cloudy_rS [array] outer radius of the ionized region simulated by cloudy (approximate!)

cloudy_QH0 [array] ionizing luminosity used in the cloudy computation

cloudy_covFac [array] covering factor assumed in the cloudy computation; only a fraction covFac of the ionizing photons are assumed to produce emission within the HII region, while the remainder are assumed to escape

cloudy_U [array] volume-averaged ionization parameter of the HII region simulated by cloudy; note that this value is approximate, not exact, and the approximation can be very poor if radiation pressure effects are significant

cloudy_Omega [array] Yeh & Matzner (2012) wind parameter for the HII region simulated by cloudy; as with U, this value is approximate, and the approximation is valid only if radiation pressure effects are small

(Present if the run being read contains a cluster_ew file)

line_names [list of string] a list giving the name for each line

line_units [list of string] a list giving the units for the equivalent width of each line

ew [array, shape (N_cluster, N_lines)] equivalent width value of each line for each cluster; units are as indicated in the units field

Raises IOError, if no photometry file can be opened ValueError, if photsystem is set to an unknown values

slugpy.**read_cluster_phot**(*model_name*, *output_dir=None*, *fmt=None*, *nofilterdata=False*, *photsystem=None*, *verbose=False*, *read_info=None*, *filters_only=False*, *read_filters=None*, *read_nebular=None*, *read_extinct=None*, *phot_only=False*)

Function to read a SLUG2 cluster_phot file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt ['txt' | 'ascii' | 'bin' | 'binary' | 'fits' | 'fits2'] Format for the file to be read. If one of these is set, the function will only attempt to open ASCII-('txt' or 'ascii'), binary ('bin' or 'binary'), or FITS ('fits' or 'fits2') formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

nofilterdata [bool] If True, the routine does not attempt to read the filter response data from the standard location

photsystem [None or string] If photsystem is None, the data will be returned in the same photometric system in which they were read. Alternately, if it is a string, the data will be converted to the specified photometric system. Allowable values are 'L_nu', 'L_lambda', 'AB', 'STMAG', and 'Vega', corresponding to the options defined in the SLUG code. If this is set and the conversion requested involves a conversion from a wavelength-based system to a frequency-based one, nofilterdata must be False so that the central wavelength of the photometric filters is available.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

filters_only [bool] If True, the code only reads the data on the filters, not any of the actual photometry. If combined with nofilterdata, this can be used to return the list of available filters and nothing else.

read_filters [None | string | listlike containing strings] If this is None, data on all filters is read. Otherwise only filters whose name(s) match the input filter names ar read.

read_nebular [None | bool] If True, only data with the nebular contribution is read; if False, only data without it is read. Default behavior is to read all data.

read_extinct [None | bool] If True, only data with extinction applied is read; if False, only data without it is read. Default behavior is to read all data.

phot_only [bool] If true, id, trial, time, and filter information are not read, only photometry

Returns A namedtuple, which can contain the following fields depending on the input options, and depending on which fields are present in the file being read:

id [array, dtype uint] unique ID of cluster

trial: array, dtype uint which trial was this cluster part of

time [array] times at which cluster spectra are output, in yr

filter_names [list of string] a list giving the name for each filter

filter_units [list of string] a list giving the units for each filter

filter_wl_eff [list] effective wavelength of each filter; this is set to None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

filter_wl [list of arrays] a list giving the wavelength table for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1

filter_response [list of arrays] a list giving the photon response function for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1

filter_beta [list] powerlaw index beta for each filter; used to normalize the photometry

filter_wl_c [list] pivot wavelength for each filter; used to normalize the photometry

phot [array, shape (N_cluster, N_filter)] photometric value in each filter for each cluster; units are as indicated in the units field

phot_neb [array, shape (N_filter, N_times, N_trials)] same as phot, but for the light after it has passed through the HII region

phot_ex [array, shape (N_filter, N_times, N_trials)] same as phot, but after extinction has been applied

phot_neb_ex [array, shape (N_filter, N_times, N_trials)] same as phot, but for the light after it has passed through the HII region and then had extinction applied

Raises IOError, if no photometry file can be opened ValueError, if photsystem is set to an unknown value

`slugpy.`**`read_cluster_prop`**(*model_name*, *output_dir=None*, *fmt=None*, *verbose=False*, *read_info=None*, *no_stellar_mass=False*, *no_neb_extinct=False*)

Function to read a SLUG2 cluster_prop file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt ['txt' | 'ascii' | 'bin' | 'binary' | 'fits' | 'fits2'] Format for the file to be read. If one of these is set, the function will only attempt to open ASCII-('txt' or 'ascii'), binary ('bin' or 'binary'), or FITS ('fits' or 'fits2') formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

no_stellar_mass [bool] Prior to 7/15, output files did not contain the stellar_mass field; this can be detected automatically for ASCII and FITS formats, but not for binary format; if True, this specifies that the binary file being read does not contain a stellar_mass field; it has no effect for ASCII or FITS files

no_neb_extinct [bool] Prior to 2/17, SLUG did not support differential nebular extinction, and thus there was no output field for it; this is detected and handled automatically for ASCII and FITS files; for binary outputs, this flag must be set for pre 2/17 output files to be read correctly

Returns A namedtuple containing the following fields:

id [array, dtype uint] unique ID of cluster

trial: array, dtype uint which trial was this cluster part of

time [array] time at which cluster's properties are being evaluated

form_time [array] time when cluster formed

lifetime [array] time at which cluster will disrupt

target_mass [array] target cluster mass

actual_mass [array] actual mass at formation

live_mass [array] mass of currently living stars

stellar_mass [array] mass of all stars, living and stellar remnants

num_star [array, dtype ulonglong] number of living stars in cluster being treated stochastically

max_star_mass [array] mass of most massive living star in cluster

A_V [array] A_V value for each cluster, in mag (present only if SLUG was run with extinction enabled)

A_Vneb [array] value of A_V applied to the nebular light for each cluster (present only if SLUG was run with both nebular emission and extinction enabled)

vpn_tuple [tuple] tuple containing arrays for any variable parameters we have (eg: VP0, VP1,VP2...) in the IMF. Each element of the tuple is an array. Present only if variable parameters were enables when SLUG was run.

`slugpy.`**`read_cluster_spec`**(*model_name*, *output_dir=None*, *fmt=None*, *verbose=False*, *read_info=None*)

Function to read a SLUG2 cluster_spec file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt ['txt' | 'ascii' | 'bin' | 'binary' | 'fits' | 'fits2'] Format for the file to be read. If one of these is set, the function will only attempt to open ASCII-('txt' or 'ascii'), binary ('bin' or 'binary'), or FITS ('fits' or 'fits2') formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

Returns A namedtuple containing the following fields:

id [array, dtype uint] unique ID of cluster

trial: array, dtype uint which trial was this cluster part of

time [array] times at which cluster spectra are output, in yr

wl [array] wavelength, in Angstrom

spec [array, shape (N_cluster, N_wavelength)] specific luminosity of each cluster at each wavelength, in erg/s/A

wl_neb [array] wavelength for the nebular spectrum, in Angstrom (present only if SLUG was run with nebular emission enabled)

spec_neb [array, shape (N_cluster, N_wavelength)] specific luminosity at each wavelength and each time for each trial, including emission and absorption by the HII region, in erg/s/A (present only if SLUG was run with nebular emission enabled)

wl_ex [array] wavelength for the extincted spectrum, in Angstrom (present only if SLUG was run with extinction enabled)

spec_ex [array, shape (N_cluster, N_wavelength)] specific luminosity at each wavelength in wl_ex and each time for each trial after extinction has been applied, in erg/s/A (present only if SLUG was run with extinction enabled)

wl_neb_ex [array] wavelength for the extincted spectrum with nebular emission, in Angstrom (present only if SLUG was run with both nebular emission and extinction enabled)

spec_neb_ex [array, shape (N_cluster, N_wavelength)] specific luminosity at each wavelength in wl_ex and each time for each trial including emission and absorption by the HII region, after extinction has been applied, in erg/s/A (present only if SLUG was run with nebular emission and extinction both enabled)

Raises IOError, if no spectrum file can be opened

`slugpy.`**`read_cluster_yield`**(*model_name*, *output_dir=None*, *fmt=None*, *verbose=False*, *read_info=None*)

Function to read a SLUG2 cluster_spec file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt ['txt' | 'ascii' | 'bin' | 'binary' | 'fits' | 'fits2'] Format for the file to be read. If one of these is set, the function will only attempt to open ASCII-('txt' or 'ascii'), binary ('bin' or 'binary'), or FITS ('fits' or 'fits2') formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

Returns A namedtuple containing the following fields:

isotope_name [array of strings, shape (N_iso)] Atomic symbols of the isotopes included in the yield table

isotope_Z [array of int, shape (N_iso)] Atomic numbers of the isotopes included in the yield table

isotope_A [array of int, shape (N_iso)] Atomic mass number of the isotopes included in the yield table

id [array, dtype uint] unique ID of cluster

trial: array, dtype uint which trial was this cluster part of

time [array] times at which cluster spectra are output, in yr

yld [array, shape (N_cluster, N_iso)] Yield of each isotope, defined as the instantaneous amount produced up to that time; for unstable isotopes, this includes the effects of decay since production

`slugpy.`**`read_filter`**(*filtername*, *filter_dir=None*)

Function to read a filter or set of filters for SLUG2. By default this function searches the SLUG_DIR/lib/filter directory, followed by the current working directory. This can be overridden by the filter_dir keyword.

Parameters

filtername [string or iterable containing strings] Name or names of filters to be read; for the special filters Lbol, QH0, QHe0, and QHe1, the return value will be None

filter_dir [string] Directory where the filter data files can be found

Returns A namedtuple containing the following fields:

wl_eff [float or array] Central wavelength of the filter, defined by wl_eff = exp(int R ln lambda dln lambda / int R dln lambda)

wl [array or list of arrays] Wavelength table for each filter, in Ang

response [array or list of arrays] Response function per photon for each filter

beta [float or array] Index beta for the filter

wl_c [float or array] Pivot wavelength for the filter; used when beta != 0 to normalize the photometry

Raises IOError, if the filter data files cannot be opened, or if the requested filter cannot be found

slugpy.**read_integrated**(*model_name*, *output_dir=None*, *fmt=None*, *nofilterdata=False*, *photsystem=None*, *verbose=False*, *read_info=None*, *no_stellar_mass=False*)

Function to read all integrated light data for a SLUG2 run.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt [string] Format for the file to be read. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'. If one of these is set, the code will only attempt to open ASCII-, binary-, or FITS-formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

nofilterdata [bool] If True, the routine does not attempt to read the filter response data from the standard location

photsystem [None or string] If photsystem is None, the data will be returned in the same photometric system in which they were read. Alternately, if it is a string, the data will be converted to the specified photometric system. Allowable values are 'L_nu', 'L_lambda', 'AB', 'STMAG', and 'Vega', corresponding to the options defined in the SLUG code. If this is set and the conversion requested involves a conversion from a wavelength-based system to a frequency-based one, nofilterdata must be False so that the central wavelength of the photometric filters is available.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'prop_name', 'phot_name', 'spec_name', 'cloudyspec_name', 'cloudylines_name' and 'format', giving the names of the files read and the format they were in; 'format' will be one of 'ascii', 'binary', or 'fits'. If one of the files is not present, the corresponding _name key will be omitted from the dict.

no_stellar_mass [bool] Prior to 7/15, output files did not contain the stellar_mass field; this can be detected automatically for ASCII and FITS formats, but not for binary format; if True, this specifies that the binary file being read does not contain a stellar_mass field; it has no effect for ASCII or FITS files

Returns A namedtuple containing the following fields:

(Always present)

time: array Times at which data are output

(Only present if an integrated_prop file is found)

target_mass [array, shape (N_times, N_trials)] Target stellar mass at each time in each trial

actual_mass [array, shape (N_times, N_trials)] Actual mass of stars created up to each time in each trial

live_mass [array, shape (N_times, N_trials)] Mass of currently-alive stars at each time in each trial

stellar_mass [array] mass of all stars, living and stellar remnants

cluster_mass [array, shape (N_times, N_trials)] Mass of living stars in non-disrupted clusters at each time in each trial

num_clusters [array, shape (N_times, N_trials), dtype ulonglong] Number of non-disrupted clusters present at each time in each trial

num_dis_clusters [array, shape (N_times, N_trials), dtype ulonglong] Number of disrupted clusters present at each time in each trial

num_fld_stars [array, shape (N_times, N_trials), dtype ulonglong] Number of living field stars (excluding those in disrupted clusters and those being treated non-stochastically) present at each time in each trial

(Only present if an integrated_spec file is found)

wl [array] wavelengths, in Angstrom

spec [array, shape (N_wavelength, N_times, N_trials)] specific luminosity at each wavelength and each time for each trial, in erg/s/A

wl_neb [array] wavelength for the nebular spectrum, in Angstrom (present only if SLUG was run with nebular emission enabled)

spec_neb [array, shape (N_wavelength, N_times, N_trials)] specific luminosity at each wavelength and each time for each trial, including emission and absorption by the HII region, in erg/s/A (present only if SLUG was run with nebular emission enabled)

wl_ex [array] wavelength for the extincted spectrum, in Angstrom (present only if SLUG was run with extinction enabled)

spec_ex [array, shape (N_wavelength, N_times, N_trials)] specific luminosity at each wavelength in wl_ex and each time for each trial after extinction has been applied, in erg/s/A (present only if SLUG was run with extinction enabled)

wl_neb_ex [array] wavelength for the extincted spectrum with nebular emission, in Angstrom (present only if SLUG was run with both nebular emission and extinction enabled)

spec_neb_ex [array, shape (N_wavelength, N_times, N_trials)] specific luminosity at each wavelength in wl_ex and each time for each trial including emission and absorption by the HII region, after extinction has been applied, in erg/s/A (present only if SLUG was run with nebular emission and extinction both enabled)

(Only present if an integrated_phot file is found)

filter_names [list of string] a list giving the name for each filter

filter_units [list of string] a list giving the units for each filter

filter_wl_cen [list] central wavelength of each filter; this is set to None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

filter_wl [list of arrays] a list giving the wavelength table for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

filter_response [list of arrays] a list giving the photon response function for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

phot [array, shape (N_filter, N_times, N_trials)] photometric value in each filter at each time in each trial; units are as indicated in the units field

If extinction is enabled, phot_ex will contain photometry after extinction has been applied.

(Only present if an integrate_yield file is found)

isotope_name [array of strings] Atomic symbols of the isotopes included in the yield table

isotope_Z [array of int] Atomic numbers of the isotopes included in the yield table

isotope_A [array of int] Atomic mass number of the isotopes included in the yield table

yld [array] Yield of each isotope, defined as the instantaneous amount produced up to that time; for unstable isotopes, this includes the effects of decay since production

(Only present if an integrated_cloudyspec file is found)

cloudy_wl [array] wavelength, in Angstrom

cloudy_inc [array, shape (N_wavelength, N_times, N_trials)] specific luminosity of the stellar radiation field at each wavelength and each time for each trial, in erg/s/A

cloudy_trans [array, shape (N_wavelength, N_times, N_trials)] specific luminosity of the stellar radiation field after it has passed through the HII region, at each wavelength and each time for each trial, in erg/s/A

cloudy_emit [array, shape (N_wavelength, N_times, N_trials)] specific luminosity of the radiation field emitted by the HII region, at each wavelength and each time for each trial, in erg/s/A

cloudy_trans_emit [array, shape (N_wavelength, N_times, N_trials)] the sum of emitted and transmitted; this is what would be seen by an observer looking at both the star cluster and its nebula

(Only present if an integrated_cloudylines file is found)

cloudy_linelabel [array, dtype='S4', shape (N_lines)] labels for the lines, following cloudy's 4 character line label notation

cloudy_linewl [array, shape (N_lines)] rest wavelength for each line, in Angstrom

cloudy_linelum [array, shape (N_lines, N_times, N_trials)] luminosity of each line at each time for each trial, in erg/s

(Only present if an integrated_cloudyphot file is found)

cloudy_filter_names [list of string] a list giving the name for each filter

cloudy_filter_units [list of string] a list giving the units for each filter

cloudy_filter_wl_eff [list] effective wavelength of each filter; this is set to None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

cloudy_filter_wl [list of arrays] a list giving the wavelength table for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

cloudy_filter_response [list of arrays] a list giving the photon response function for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

cloudy_filter_beta [list] powerlaw index beta for each filter; used to normalize the photometry

cloudy_filter_wl_c [list] pivot wavelength for each filter; used to normalize the photometry

cloudy_phot_trans [array, shape (N_filter, N_times, N_trials)] photometric value in each filter at each time in each trial for the transmitted light (i.e., the starlight remaining after it has passed through the HII region); units are as indicated in the units field

cloudy_phot_emit [array, shape (N_filter, N_times, N_trials)] photometric value in each filter at each time in each trial for the emitted light (i.e., the diffuse light emitted by the HII region); units are as indicated in the units field

cloudy_phot_trans_emit [array, shape (N_filter, N_times, N_trials)] photometric value in each filter at each time in each trial for the transmitted plus emitted light (i.e., the light coming directly from the stars after absorption by the HII region, plus the diffuse light emitted by the HII region); units are as indicated in the units field

(Only present if an integrated_cloudyparams file is found)

cloudy_hden [array] number density of H nuclei at the inner edge of the ionized region simulated by cloudy

cloudy_r0 [array] inner radius of the ionized region simulated by cloudy

cloudy_rS [array] outer radius of the ionized region simulated by cloudy (approximate!)

cloudy_QH0 [array] ionizing luminosity used in the cloudy computation

cloudy_covFac [array] covering factor assumed in the cloudy computation; only a fraction covFac of the ionizing photons are assumed to produce emission within the HII region, while the remainder are assumed to escape

cloudy_U [array] volume-averaged ionization parameter of the HII region simulated by cloudy; note that this value is approximate, not exact, and the approximation can be very poor if radiation pressure effects are significant

cloudy_Omega [array] Yeh & Matzner (2012) wind parameter for the HII region simulated by cloudy; as with U, this value is approximate, and the approximation is valid only if radiation pressure effects are small

slugpy.**read_integrated_phot**(*model_name*, *output_dir=None*, *fmt=None*, *nofilterdata=False*, *photsystem=None*, *verbose=False*, *read_info=None*, *filters_only=False*, *read_filters=None*, *read_nebular=None*, *read_extinct=None*)

Function to read a SLUG2 integrated_phot file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt ['txt' | 'ascii' | 'bin' | 'binary' | 'fits' | 'fits2'] Format for the file to be read. If one of these is set, the function will only attempt to open ASCII-('txt' or 'ascii'), binary ('bin' or 'binary'), or FITS ('fits' or 'fits2') formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

nofilterdata [bool] If True, the routine does not attempt to read the filter response data from the standard location

photsystem [None or string] If photsystem is None, the data will be returned in the same photometric system in which they were read. Alternately, if it is a string, the data will be converted to the specified photometric system. Allowable values are 'L_nu', 'L_lambda', 'AB', 'STMAG', and 'Vega', corresponding to the options defined in the SLUG code. If this is set and the conversion requested involves a conversion from a wavelength-based system to a frequency-based one, nofilterdata must be False so that the central wavelength of the photometric filters is available.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

filters_only [bool] If True, the code only reads the data on the filters, not any of the actual photometry. If combined with nofilterdata, this can be used to return the list of available filters and nothing else.

read_filters [None | string | listlike containing strings] If this is None, data on all filters is read. Otherwise only filters whose name(s) match the input filter names ar read.

read_nebular [None | bool] If True, only data with the nebular contribution is read; if False, only data without it is read. Default behavior is to read all data.

read_extinct [None | bool] If True, only data with extinction applied is read; if False, only data without it is read. Default behavior is to read all data.

Returns A namedtuple , which can contain the following fields depending on the input options, and depending on which fields are present in the file being read:

time [array, shape (N_times) or shape (N_trials)] Times at which data are output; shape is either N_times (if the run was done with fixed output times) or N_trials (if the run was done with random output times)

filter_names [list of string] a list giving the name for each filter

filter_units [list of string] a list giving the units for each filter

filter_wl_eff [list] effective wavelength of each filter; this is set to None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

filter_wl [list of arrays] a list giving the wavelength table for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1

filter_response [list of arrays] a list giving the photon response function for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1

filter_beta [list] powerlaw index beta for each filter; used to normalize the photometry

filter_wl_c [list] pivot wavelength for each filter; used to normalize the photometry

phot [array, shape (N_filter, N_times, N_trials)] photometric value in each filter at each time in each trial; units are as indicated in the units field

phot_neb [array, shape (N_filter, N_times, N_trials)] same as phot, but for the light after it has passed through the HII region

phot_ex [array, shape (N_filter, N_times, N_trials)] same as phot, but after extinction has been applied

phot_neb_ex [array, shape (N_filter, N_times, N_trials)] same as phot, but for the light after it has passed through the HII region and then had extinction applied

Raises IOError, if no photometry file can be opened ValueError, if photsystem is set to an unknown value

`slugpy.`**`read_integrated_prop`**(*model_name*, *output_dir=None*, *fmt=None*, *verbose=False*, *read_info=None*, *no_stellar_mass=False*)

Function to read a SLUG2 integrated_prop file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt ['txt' | 'ascii' | 'bin' | 'binary' | 'fits' | 'fits2'] Format for the file to be read. If one of these is set, the function will only attempt to open ASCII-('txt' or 'ascii'), binary ('bin' or 'binary'), or FITS ('fits' or 'fits2') formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

no_stellar_mass [bool] Prior to 7/15, output files did not contain the stellar_mass field; this can be detected automatically for ASCII and FITS formats, but not for binary format; if True, this specifies that the binary file being read does not contain a stellar_mass field; it has no effect for ASCII or FITS files

Returns A namedtuple containing the following fields:

time [array, shape (N_times) or shape (N_trials)] Times at which data are output; shape is either N_times (if the run was done with fixed output times) or N_trials (if the run was done with random output times)

target_mass [array, shape (N_times, N_trials)] Target stellar mass at each time

actual_mass [array, shape (N_times, N_trials)] Actual mass of stars created up to each time in each trial

live_mass [array, shape (N_times, N_trials)] Mass of currently-alive stars at each time in each trial

stellar_mass [array] mass of all stars, living and stellar remnants

cluster_mass [array, shape (N_times, N_trials)] Stellar mass in non-disrupted clusters at each time in each trial

num_clusters [array, shape (N_times, N_trials), dtype ulonglong] Number of non-disrupted clusters present at each time in each trial

num_dis_clusters [array, shape (N_times, N_trials), dtype ulonglong] Number of disrupted clusters present at each time in each trial

num_fld_stars [array, shape (N_times, N_trials), dtype ulonglong] Number of living field stars (excluding those in disrupted clusters and those being treated non-stochastically) present at each time in each trial

`slugpy.`**`read_integrated_spec`**(*model_name*, *output_dir=None*, *fmt=None*, *verbose=False*, *read_info=None*)

Function to read a SLUG2 integrated_spec file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt ['txt' | 'ascii' | 'bin' | 'binary' | 'fits' | 'fits2'] Format for the file to be read. If one of these is set, the function will only attempt to open ASCII-('txt' or 'ascii'), binary ('bin' or 'binary'), or FITS ('fits' or 'fits2') formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

Returns A namedtuple containing the following fields:

time [array, shape (N_times) or shape (N_trials)] Times at which data are output; shape is either N_times (if the run was done with fixed output times) or N_trials (if the run was done with random output times)

wl [array] wavelength, in Angstrom

spec [array, shape (N_wavelength, N_times, N_trials)] specific luminosity at each wavelength and each time for each trial, in erg/s/A

wl_neb [array] wavelength for the nebular spectrum, in Angstrom (present only if SLUG was run with nebular emission enabled)

spec_neb [array, shape (N_wavelength, N_times, N_trials)] specific luminosity at each wavelength and each time for each trial, including emission and absorption by the HII region, in erg/s/A (present only if SLUG was run with nebular emission enabled)

wl_ex [array] wavelength for the extincted spectrum, in Angstrom (present only if SLUG was run with extinction enabled)

spec_ex [array, shape (N_wavelength, N_times, N_trials)] specific luminosity at each wavelength in wl_ex and each time for each trial after extinction has been applied, in erg/s/A (present only if SLUG was run with extinction enabled)

wl_neb_ex [array] wavelength for the extincted spectrum with nebular emission, in Angstrom (present only if SLUG was run with both nebular emission and extinction enabled)

spec_neb_ex [array, shape (N_wavelength, N_times, N_trials)] specific luminosity at each wavelength in wl_ex and each time for each trial including emission and absorption by the HII region, after extinction has been applied, in erg/s/A (present only if SLUG was run with nebular emission and extinction both enabled)

`slugpy.`**`read_integrated_yield`**(*model_name*, *output_dir=None*, *fmt=None*, *read_info=None*, *verbose=False*)

Function to read a SLUG2 integrated_yield file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt ['txt' | 'ascii' | 'bin' | 'binary' | 'fits' | 'fits2'] Format for the file to be read. If one of these is set, the function will only attempt to open ASCII-('txt' or 'ascii'), binary ('bin' or 'binary'), or FITS ('fits' or 'fits2') formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

Returns A namedtuple containing the following fields:

time [array, shape (N_times) or shape (N_trials)] Times at which data are output; shape is either N_times (if the run was done with fixed output times) or N_trials (if the run was done with random output times)

isotope_name [array of strings, shape (N_iso)] Atomic symbols of the isotopes included in the yield table

isotope_Z [array of int, shape (N_iso)] Atomic numbers of the isotopes included in the yield table

isotope_A [array of int, shape (N_iso)] Atomic mass number of the isotopes included in the yield table

yld [array, shape (N_iso, N_times) or (N_iso, N_trials)] Yield of each isotope, defined as the instantaneous amount produced up to that time; for unstable isotopes, this includes the effects of decay since production

`slugpy.`**`read_summary`**(*model_name*, *output_dir=None*)

Function to open a SLUG output summary file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

Returns

summary [dict] A dict containing all the keywords stored in the output file

Raises IOError, if a summary file for the specified model cannot be found

slugpy.**slug_open**(*filename*, *output_dir=None*, *fmt=None*)

Function to open a SLUG2 output file.

Parameters

filename [string] Name of the file to open, without any extension. The following extensions are tried, in order: .txt, .bin, .fits

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR/output directory if the SLUG_DIR environment variable is set

fmt ['txt' | 'ascii' | 'bin' | 'binary' | 'fits' | 'fits2'] Format for the file to be read. If one of these is set, the function will only attempt to open ASCII-('txt' or 'ascii'), binary ('bin' or 'binary'), or FITS ('fits' or 'fits2') formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

Returns

fp [file or astropy.io.fits.hdu.hdulist.HDUList] A file object pointing the file that has been opened

fname [string] Name of the file that was opened

Raises IOError, if a file of the specified name cannot be found

class slugpy.**slug_pdf**(*pdffile=None*)

A class that implements the SLUG PDF drawing method. This class contains a method to parse slug-formatted PDF files and then draw values from the PDFs they specify. This class is thread-safe, in the sense that if multiple slug_pdf instances are instantiated in different threads, the random streams they generate will not be identical.

draw(**d*)

Draw from the PDF

Parameters:

d0, d1, . . . , dn [int, optional] Dimensions of the returned array; if left unspecified, a single python float is returned

Returns:

x [float or array] One or more numbers drawn from the PDF

slugpy.**write_cluster**(*data*, *model_name*, *fmt*)

Function to write a set of output cluster files in SLUG2 format, starting from a cluster data set as returned by read_cluster.

Parameters

data [namedtuple] Cluster data to be written, in the namedtuple format returned by read_cluster

model_name [string] Base file name to give the model to be written. Can include a directory specification if desired.

fmt ['txt' | 'ascii' | 'bin' | 'binary' | 'fits' | 'fits2'] Format for the output file; 'txt' and 'ascii' produce ASCII text files, 'bin' or 'binary' produce binary files, and 'fits' or 'fits2' product FITS files; 'fits2' uses an ordering that allows for more efficient querying of outputs too large to fit in memory

Returns Nothing

slugpy.**write_integrated**(*data*, *model_name*, *fmt*)

Function to write a set of output integrated files in SLUG2 format, starting from an integrated data set as returned by read_integrated.

Parameters

data [namedtuple] Integrated data to be written, in the namedtuple format returned by read_integrated

model_name [string] Base file name to give the model to be written. Can include a directory specification if desired.

fmt [string] Format for the output file. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'.

Returns Nothing

CHAPTER

ELEVEN

CLOUDY_SLUG: AN AUTOMATED INTERFACE TO CLOUDY

SLUG stochastically generates stellar spectra, and it includes an approximate computation of the nebular lines produced when those photons interact with the interstellar medium. However, this approximation ignores a number of potentially important effects, and does not properly account for the stochastic nature of the stellar spectra. To perform a much more accurate calculation, SLUG includes an automated interface to cloudy (Ferland et al., 2013, RMxAA, 49, 137). This can be used to post-process the output of a SLUG run in order to compute nebular emission.

11.1 cloudy_slug Basics

The basic steps (described in greater detail below) are as follows:

1. Get cloudy installed and compiled, following the directions on the cloudy website.
2. Set the environment variable `$CLOUDY_DIR` to the directory where the cloudy executable `cloudy.exe` is located. If you are using a `bash`-like shell, the syntax for this is:

   ```
   export CLOUDY_DIR = /path/to/cloudy
   ```

   while for a `csh`-like shell, it is:

   ```
   setenv CLOUDY_DIR /path/to/cloudy
   ```

3. If you desire, edit the cloudy input template `cloudy_slug/cloudy.in_template` and the line list `cloudy_slug/LineList_HII.dat`. There are the template input files that will be used for all the cloudy runs, and their syntax follows the standard cloudy syntax. They control things like the density and element abundances in the nebula – see *The cloudy_slug Input Template* for more details.
4. Perform the desired SLUG simulation. The SLUG simulation outputs must include spectra and photometry, and one of the photometric bands output must be `QH0` (see *Photometric Filter Keywords*). Depending on whether one is running in integrated or cluster mode (see *Integrated versus Cluster Spectra*), either integrated specta and photometry or cluster spectra and photometry are required.
5. Invoke the cloudy_slug interface script via:

   ```
   python cloudy_slug/cloudy_slug.py SLUG_MODEL_NAME
   ```

   where `SLUG_MODEL_NAME` is the name of the SLUG run to be processed. See *The cloudy_slug Physical Model* for more information on the underlying physical model assumed in the calculation, and *The cloudy_slug Interface Script* for more details on the python script and its options.
6. The output will be stored as a series of additional output files of with names of the form SLUG_MODEL_NAME_*cloudy*.ext, where the extension is .txt, .bin, or .fits, depending on the format in which the orignal SLUG output was stored. These files can be processed automatically by the slugpy helper

routines (see *slugpy – The Python Helper Library*). See *Full Description of cloudy_slug Output* for a description of the outputs.

Note that some care is required in selecting the conditions passed to cloudy to ensure that the results are physically sensible. Users are strongly encouraged to read *The cloudy_slug Physical Model* to understand exactly what physical assumptions are being made, and to ensure that they are reasonable.

11.2 The cloudy_slug Physical Model

The cloudy_slug code computes emission from a spherical HII region surrounding a stellar population. The stellar population comes from SLUG, and the emission calculation is performed with cloudy. Combining the two requires some physical assumptions and inputs, which are explained in this section.

11.2.1 Integrated versus Cluster Spectra

SLUG outputs both integrated spectra for all the stars in a galaxy, and spectra for individual clusters. Both the integrated spectra and the individual cluster spectra can be processed by cloudy. However, it is important to understand the implicit physical assumptions that one is making while doing so. If one has a galaxy where all stars are in clusters (i.e., cluster formation fraction is unity and there is no cluster disruption), then the integrated starlight spectrum is just the sum of the individual cluster spectra. For nebular emission, however, this is not the case: nebular emission does not, in general, add linearly.

For this reason, if one processes the integrated spectrum through cloudy, the implicit physical assumption is that the entire galaxy is a single giant HII region being ionized by the starlight of all the clusters present. If one processes the individual cluster spectra instead, the implicit physical picture is that there is no overlap whatsoever between the HII regions surrounding different star clusters. Reality almost certainly lies somewhere between these two extremes, but it is important to understand physically what assumption one is making by adopting one or the other. We refer to processing the integrated spectrum as integrated mode, and to processing the individual cluster spectra as cluster mode. Note that cluster mode can be very computationally intensive if there are many clusters present, and that in cluster mode there is no processing of nebular emission produced by field stars.

In either mode, the spectrum that is used to compute the nebular emission will be the *unextincted, non-redshifted* spectrum computed by SLUG.

11.2.2 Nebular Properties

Computing the nebular emission requires specifying the physical properties of the interstellar gas into which the photons propagate. Codes like cloudy require that the HII region be described by an inner radius r_0 and a number density n_0 of hydrogen nuclei at that radius. One option for cloudy_slug is that these parameters can be set in the cloudy inputs as they would be for a normal cloudy run. However, these parameters are not necessarily the most convenient or descriptive ones with which to describe HII regions. For this reason, cloudy_slug allows users to specify HII region properties in a number of other more convneient ways.

The basic assumptions made in cloudy_slug's parameterization is that the HII region is isobaric and isothermal, at all points hydrogen is fully ionized and helium is singly ionized, and that radiation pressure is negligible. The HII region occupies a spherical shell bounded by an inner radius r_0 and an outer radius r_1. The inner radius is set by the presence of a bubble of shocked stellar wind material at a temperature $\sim 10^6$ K, which is assumed to be optically thick to ionizing photons. The outer radius is set by the location where all the ionizing photons have been absorbed.

Under these assumptions, the inner density n_0 is simply the (uniform) density $n_{\rm II}$ throughout the ionized region, and

the ionizing photon luminosity passing through a shell of material at a distance r from the stars is

$$Q(r) = Q(\mathrm{H}^0)\left[1 - \left(\frac{r}{r_S}\right)^3 + \left(\frac{r_0}{r_S}\right)^3\right],$$

where $Q(\mathrm{H}^0)$ is the hydrogen-ionizing luminosity of the source and r_S is the Stromgren radius, given by

$$r_S = \left(\frac{3Q(\mathrm{H}^0)}{4\pi\alpha_B f_e n_{\mathrm{II}}^2}\right)^{1/3}.$$

Here α_B is the case B recombination coefficient and f_e is the abundance of electrons per H nucleus. For the purposes of cloudy_slug, we take these two quantities to have the fixed values $\alpha_B = 2.59 \times 10^{-13}$ cm^3 s^{-1}, appropriate for a temperature of 10^4 K, and $f_e = 1.1$, appropriate for a region where He is singly ionized.

From this setup one can define some useful dimensionless numbers. One is the wind parameter Ω introduced by Yeh & Matnzer (2012, ApJ, 757, 108), which under the simple assumptions made in cloudy_slug is given by

$$\Omega = \frac{r_0^3}{r_1^3 - r_0^3}$$

i.e., it is just the ratio of the volume occupied by the wind gas to that occupied by the photoionized gas. The value of Ω determines whether winds are important ($\Omega \gg 1$) or unimportant ($\Omega \ll 1$) for the dynamics of the HII region. The second dimensionless parameter is the volume-averaged ionization parameter

$$\mathcal{U} = \frac{3}{4\pi(r_1^3 - r_0^3)}\int_{r_0}^{r_1}\left(\frac{Q(r)}{4\pi r^2 c f_i n_{\mathrm{II}}}\right)4\pi r^2\, dr.$$

Here f_i is the number of free ions per H nucleus, and is equal to $f_i = 1.1$ under the assumption that He is singly ionized. The quantity in parentheses is the ratio of the ionizing photon to ion number densities at radius r. The value of $\mathcal{U}$ is, together with n_{II}, the most important factor in determining the output spectrum. A third useful dimensionless parameter is the ionization parameter at the inner radius,

$$\mathcal{U}_0 = \frac{Q(\mathrm{H}^0)}{4\pi r_0^2 f_i n_{\mathrm{II}} c}.$$

The various quantities are not unrelated. It is straightforward to show that they are constrained by the following relationships:

$$r_0 = \Omega^{1/3} r_S$$

$$r_1 = (1+\Omega)^{1/3} r_S$$

$$\mathcal{U} = \left[\frac{81\alpha_B^2 n_{\mathrm{II}} Q(\mathrm{H}^0)}{256\pi c^3 f_e}\right]^{1/3}\left[(1+\Omega)^{4/3} - \Omega^{1/3}\left(\frac{4}{3}+\Omega\right)\right]$$

$$= \left[\frac{81\alpha_B Q(\mathrm{H}^0)}{64\pi c^2 f_e r_S}\right]^{1/2}\left[(1+\Omega)^{4/3} - \Omega^{1/3}\left(\frac{4}{3}+\Omega\right)\right]$$

$$\mathcal{U}_0 = \left[\frac{\alpha_B^2 n_{\mathrm{II}} Q(\mathrm{H}^0)}{36\pi c^3 f_e}\right]^{1/3}\frac{1}{\Omega^{2/3}}$$

$$= \frac{4}{9}\Omega^{-2/3}\left[(1+\Omega)^{4/3} - \Omega^{1/3}\left(\frac{4}{3}+\Omega\right)\right]^{-1}\mathcal{U}$$

These relations may be used to compute any four of the quantities $n_{\rm II}$, r_0, r_1, $\mathcal{U}$, $\mathcal{U}_0$ and Ω given the other two. *slugpy – The Python Helper Library* provides a class `hiiregparam` that can be used to perform such a computation.

Given this background, cloudy_slug allows the user to specify the physical properties of the HII region by setting any two of the following six quantities:

1. The photoionized gas density $n_{\rm II}$.
2. The inner radius r_0.
3. The outer radius r_1.
4. The volume-averaged ionization parameter $\mathcal{U}$.
5. The inner radius ionization parameter $\mathcal{U}_0$.
6. The wind parameter Ω.

The two quantities chosen can be specified exactly, or can be drawn from a specified PDF. One final option, which is only available in cluster mode, is to obtain the required quantities from a dynamical model – see *Dynamical Mode*.

A few caveats are in order at this point.

1. Not all combinations of values are realizable. In addition to the obvious constraints (e.g., $r_1 > r_0$), there are some subtle ones. For example, for any given ionizing luminosity $Q({\rm H}^0)$ and density $n_{\rm II}$, the value of $\mathcal{U}$ is bounded from above. Increasing the wind parameter Ω can allow arbitrarily small values of $\mathcal{U}$, but not arbitrarily large ones. If the user requests a physically impossible combination of parameters, cloudy_slug will correct the parameters to the allowed range and run, while issuing a warning.
2. Even for parameters that are not physically impossible, the results may not be sensible, and may cause cloudy to crash in extreme cases. For example, if one sets $\Omega = 0$ and $\mathcal{U} = 10^{-4}$, then for an ionizing lumnosity of $Q({\rm H}^0) = 10^{50}$ photons/s (typical for a cluster of $\sim 10^4 M_\odot$), the corresponding density is $n_{\rm II} \approx 10^{-5}$ cm^{-3}! As this density the gas will be fully ionized by cosmic rays and the extragalactic background, and it makes no sense to think of it as an HII region. Caution is required.
3. The parameter combinations $(r_0, \mathcal{U})$ and $(r_1, \mathcal{U}_0)$ not allowed because they do not define a unique solution for the other parameters (the resulting equations have multiple physically-valid solutions).
4. The relations given above are only valid if radiation pressure is not dynamically significant. If it is, then there are no known analytic relations between the various quantities. The cloudy_slug code will still run, and will use the relations above, but the actual HII region properties may be markedly different from those requested. In cases where radiation pressure is important, it is generally advisable to save the HII region physical conditions output by cloudy to compute quatities from them directly. The cloudy_slug script will issue a warning if radiation pressure is expected to be significant for the HII region being computed. As a rule of thumb, radiation pressure is significant if

$$\zeta \equiv \frac{r_{\rm ch}}{r_1} > 1$$

where $r_{\rm ch}$ is the characteristic radius defined by Krumholz & Matzner (2009, ApJ, 703, 1352) as

$$r_{\rm ch} = \frac{\alpha_B}{12\pi\phi} \left(\frac{\epsilon_0}{2 f_e k_B T_{\rm II}}\right)^2 f_{\rm trap}^2 \frac{\psi^2 Q({\rm H}^0)}{c^2}$$

Here $\phi = 0.73$ is the fraction of ionizing photons absorbed by hydrogen atoms rather than dust, $\epsilon_0 = 13.6$ eV is the hydrogen ionization potential, $T_{\rm II} = 10^4$ K is the temperature inside the HII region, $f_{\rm trap} = 2$ is the trapping factor that accounts for stellar wind and trapped infrared radiation pressure, and $\psi = 3.2$ is the mean photon energy in Rydberg for a fully sampled IMF at zero age.

11.2.3 Dynamical Mode

In cluster mode, cloudy_slug allows an additional option to derive the physical properties of the HII region. They can be computed from a dynamical model of HII region expansion, taken from Krumholz & Matzner (2009, ApJ, 703, 1352). In this model, the radius of an HII region can be computed as a function of the ionizing luminosity $Q(\mathrm{H}^0)$, ambient hydrogen number density n_{H}, and star cluster age t as

$$r_1 = r_{\mathrm{ch}} \left(x_{\mathrm{rad}}^{7/2} + x_{\mathrm{gas}}^{7/2} \right)^{2/7}$$

$$x_{\mathrm{rad}} = (2\tau^2)^{1/4}$$

$$x_{\mathrm{gas}} = (49\tau^2/36)^{2/7}$$

$$\tau = t/t_{\mathrm{ch}}$$

$$t_{\mathrm{ch}} = \left(\frac{4\pi\mu m_{\mathrm{H}} n_{\mathrm{H}} c r_{\mathrm{ch}}^4}{3 f_{\mathrm{trap}} Q(\mathrm{H}^0) \psi \epsilon_0} \right)^{1/2}$$

Definitions of various quantities appearing in these equations are given above. The quantity $\mu = 1.4$ is the mean mass per hydrogen nucleus for gas of the standard cosmic composition.

We refer to this method of computing HII region properties as dynamical mode. In this mode, a user can specify the properties of the nebula in terms of an ambient density n_{H} and a wind parameter Ω. All other quantities are derived from these two and from the ionizing luminosity $Q(\mathrm{H}^0)$ and age t of each cluster. Dynamical mode can only be used in combination with cluster mode, not integrated mode, because composite stellar populations do not have well-defined ages.

11.3 The cloudy_slug Input Template

The cloudy_slug interface operates by reading SLUG output spectra and using them as inputs to a cloudy calculation. However, cloudy obviously requires many input parameters beyond simply the spectrum of the input radiation field. These parameters are normally provided by an input file whose format is as described in the cloudy documentation. The cloudy_slug interface works by reading a *template* input file that specifies all these parameter, and which will be used as a basis for the final cloudy input files that will contain the SLUG spectra.

In general the template input file looks just like an ordinary cloudy input file, subject to the following restrictions:

1. The input file *must not* contain any commands that specify the luminosity, intensity, or the spectral shape. These will be inserted automatically by the cloudy_slug script.
2. The input file *may* contain a radius command specifying the inner radius of the HII region. If it does not, then the user must specify the radius in another way, by setting 2 of the 6 inputs described in *Nebular Properties* (for simulations not done in dynamic mode) or by setting an ambient density and wind parameters in *Dynamical Mode*. If the user does set these quantities, any radius command in the template file will be ignored, and a warning message will be issued if one is found. Finally, note that cluster_slug will only compute derived parameters correctly from a radius in the template file if the radius is specified in cloudy's default format, by giving a log of the radius in cm; the keywords "linear" and "parsecs" are not currently supported.
3. The input file *may* contain a hydrogen density command specifying the starting hydrogen density. The rules for this are the same as for the radius command.

4. Any outputs to be written (specified using the `save` or `punch` keywords) must give file names containing the string `$OUTPUT_FILENAME`. This string will be replaced by the cloudy_slug script to generate a unique file name for each cloudy run, and to read back these outputs for post-processing.
5. The cloudy_slug output will contain output spectra only if the cloudy input file contains a `save last continuum` command. See *Full Description of cloudy_slug Output*.
6. The cloudy_slug output will contain output line luminosities only if the cloudy input file contains a `save last line list emergent absolute column` command. See *Full Description of cloudy_slug Output*.
7. The cloudy_slug output will contain output physical conditions and dimensionless values only if the cloudy input file contains a `save last hydrogen conditions` command. See *Full Description of cloudy_slug Output*.
8. If any other outputs are produced by the input file, they will neither be processed nor moved, deleted, or otherwise changed by the cloudy_slug script.
9. Running cloudy in grid mode is not currently supported.

An example cloudy input file with reasonable parameter choices is provided as `cloudy_slug/cloudy_in.template` in the main directory of the SLUG repository.

In addition to the input file, the default template makes use of a cloudy line list file to specify which line luminosities should be output (see the cloudy documentation for details). The template points to the file `cloudy_slug/LineList_HII.data` (which is identical to cloudy's default line list for HII regions), but any other valid cloudy line list file would work as well.

11.4 The cloudy_slug Interface Script

The `cloudy_slug.py` script provides the interface between SLUG and cloudy. Usage for this script is as follows:

```
cloudy_slug.py [-h] [-a AGEMAX] [--cloudypath CLOUDYPATH]
               [--cloudytemplate CLOUDYTEMPLATE] [-cm]
               [-cf COVERINGFAC] [-d] [-hd HDEN] [-ip IONPARAM]
               [-ip0 IONPARAM0] [-ipm IONPARAMMAX]
               [--ionparammin IONPARAMMIN] [-nl NICELEVEL]
               [-n NPROC] [-ps PARAMSAFETY] [-qm QH0MIN] [-r0 R0]
               [-r1 R1] [-s] [--slugformat SLUGFORMAT]
               [--slugpath SLUGPATH] [-t TMPDIR] [-v] [-wp WINDPARAM]
               [-wr]
               slug_model_name [start_spec] [end_spec]
```

The positional arguments are as follows:

- `slug_model_name`: this is the name of the SLUG output to be used as a basis for the cloudy calculation. This should be the same as the `model_name` parameter used in the SLUG simulation, with the optional addition of a path specification in front.
- `start_spec`: default behavior is to run cloudy on all the integrated spectra or cluster spectra (see *Integrated versus Cluster Spectra*). If this argument is set, cloudy will only be run in spectra starting with the specified trial number or cluster number; numbers are 0-offset, so the first trial/cluster is 0, the next is 1, etc.
- `end_spec`: same as `start_spec`, but specifying the last cluster to be processed. Per standard python convention, the spectra processed will go up to but not include `end_spec`.

The following optional arguments control paths and file locations:

- `--cloudypath CLOUDYPATH`: path to the cloudy executable; default is `$CLOUDY_DIR/cloudy.exe`

- `--cloudytemplate CLOUDYTEMPLATE`: cloudy input file template (see *The cloudy_slug Input Template*); default is `$SLUG_DIR/cloudy_slug/cloudy.in_template`
- `--slugformat SLUGFORMAT`: the format of slug output data to use; valid values are `ascii`, `bin`, `binary`, and `fits`. By default `cloudy_slug` checks for any output whose name and path match the model name and search path, regardless of format.
- `--slugpath SLUGPATH`: path to the SLUG output data. If not set, cloudy_slug searches for an appropriately-named set of output files first in the current working directory, and next in `$SLUG_DIR/output`
- `-t TMPDIR, --tmpdir TMPDIR`: location of the temporary directory where temporary files should be stored; defaults to *./cloudy_tmp_MODEL_NAME*.

The following arguments control how HII regions are processed:

- `-a AGEMAX, --agemax AGEMAX`: maximum cluster age in Myr for cloudy computation. Cloudy will not be run on clusters older than this value, and the predicted nebular emission for such clusters will be recorded as zero. Default value is 10 Myr. This argument only has an effect if running in cluster mode (see *Integrated versus Cluster Spectra*); otherwise it is ignored.
- `-cf COVERINGFRAC, --coveringfrac COVERINGFRAC`: this sets the covering fraction of the HII region, i.e., the fraction of ionizing photons that are assumed to produce nebular emission; the output luminosity is decreased by a factor of the covering fraction
- `-cm, --clustermode`: if this argument is set, then cloudy_slug will process cluster spectra; the default behavior is to process integrated spectra
- `--ionparammax IONPARAMMAX`: maximum value for the inner radius ionization parameter $\mathcal{U}_0$. If the value falls outside this range, the behavior is controlled by the setting of the `paramsafety` option (see below).
- `--ionparammmin IONPARAMMIN`: same as `ionparammax`, but sets a minimum instead of a maximum.
- `-qm QH0MIN, --qH0min QH0MIN`: minimum ionizing luminosity for which to run cloudy (default = 0). As with `--agemax`, for clusters / times where cloudy is not run, that case will still appear in the output, but the nebular spectra and nebular line luminosities will all be set to zero.

The following parameters specify the physical properties of HII regions, as explained in *Nebular Properties*. Parameters can be set to either fixed values, or to the names of PDF files. Any numerical value given is interpreted as a fixed constant, while non-numerical values are interpreted as the names of PDF files that specify a PDF from which the corresponding parameter is to be drawn. See *Probability Distribution Functions* for details on PDF file formats.

- `-hd HDEN, --hden HDEN`: hydrogen density in HII region, $n_{\rm II}$
- `-ip IONPARAM, --ionparam IONPARAM`: volume-averaged ionization parameter, $\mathcal{U}$
- `-ip0 IONPARAM, --ionparam0 IONPARAM0`: ionization parameter at HII region inner edge, $\mathcal{U}_0$
- `-r0 R0`: inner radius of the HII region
- `-r1 R1`: outer radius of the HII region
- `-wp WINDPARAM, --windparam WINDPARAM`: wind parameter, Ω

The following arguments control general code behavior:

- `-h, --help`: prints a help message and then exits
- `-nl NICELEVEL, --nicelevel NICELEVEL`: if this is set, then the cloudy processes launched by the script will be run at this nice level. If it is not set, they will not be nice'd. Note that this option will only work correctly on platforms that support nice.
- `-n NPROC, --nproc NPROC`: number of simultaneous cloudy processes to run; default is the number of cores available on the system

- `--ps PARAMSAFETY, --paramsafety PARAMSAFETY`: specifies how to handle situations where the combination of input HII region parameters is not physically allowed, or falls outside the bounds set by `ionparammin` and `ionparammax`. Available options are:
  - `warn`: one of the input parameters is adjusted to bring it to a physically-allowed value, and a warning is issued; the run continues. This is the default behaviour.
  - `skip`: runs with unphysical parameter choices are skipped; the parameters that were chosen are recorded in the output file, but the output spectrum, all line luminosities, and all other parameters are set to 0, and cloudy is not run.
  - `halt`: if a forbidden parameter combination is found, the entire cloudy_slug run is halted.
  - `redraw`: if a forbidden parameter combination is found, and one or more parameters are being drawn from a PDF, a new set of parameters will be drawn from the PDF. Redrawing will continue up to 100 times until a physically-allowed parameter combination is found. If no valid parameter combination is found after 100 attempts, revert to `skip`.
- `-s, --save`: by default, cloudy_slug will extract line and spectral data from the cloudy outputs and store them as described in *Full Description of cloudy_slug Output*, then delete the cloudy output files. If this option is set, the cloudy output files will NOT be deleted, and will be left in place, in sub-directory of the working directory called `cloudy_tmp_MODEL_NAME` where `MODEL_NAME` is the SLUG model name. WARNING: cloudy's outputs are written in ASCII and are quite voluminous, so choose this option only if you are running cloudy on a small number of SLUG spectra and/or you are prepared to store hundreds of GB or more. The data that `cloudy_slug` extract are much, much smaller, and (if you do not use ASCII format) are stored in a much more compact form.
- `-v, --verbose`: if this option is set, cloudy_slug produces verbose output as it runs
- `-wr, --writeparams`: if set, this option causes `cloudy_slug` to write out a file beginning with `cloudy_slug.param` for each cloudy run. This file is written in the same directory used by the save command, and it contains an ASCII printout of the various parameters. This option is only applied if `--save` is also set.

11.5 Full Description of cloudy_slug Output

The cloudy_slug script will automatically process the cloudy output and produce a series of new output files, which will be written to the same directory where the input SLUG files are located, and using the same output mode (ASCII text, raw binary, or FITS – see *Output Files and Format*). If cloudy_slug is called to process integrated spectra, the four output files will be `MODEL_NAME_integrated_cloudyparams.ext`, `MODEL_NAME_integrated_cloudylines.ext`, `MODEL_NAME_integrated_cloudyphot.ext`, and `MODEL_NAME_integrated_cloudyspec.ext`, where the extension `.ext` is one of `.txt`, `.bin`, or `.fits`, depending on the `output_mode`. If cloudy_slug is run on cluster spectra, the four output files will be `MODEL_NAME_cluster_cloudyparams.ext`, `MODEL_NAME_cluster_cloudylines.ext`, `MODEL_NAME_cluster_cloudyphot.ext`, and `MODEL_NAME_cluster_cloudyspec.ext`. All of these output files will be read and processed automatically if the outputs are read using `read_integrated` or `read_cluster` in the *slugpy – The Python Helper Library* library.

The format of these files is described below.

11.5.1 The `integrated_cloudyparams` File

This file contains the input parameters for the cloudy runs, and quantities derived from them. All parameters are as defined in *Nebular Properties*. The output file consists of a series of entries containin the following fields:

- `Trial`: which trial these data are from

- `Time`: evolution time at which the output is produced
- `Hden`: number density of hydrogen nuclei at the inner edge of the HII region, in H/cm^3
- `R0`: radius of the inner edge of the HII region, in cm
- `R1`: radius of the outer edge of the HII region, in cm
- `QH0`: ionizing luminosity input to cloudy, in photons/s
- `CovFac`: covering factor used
- `U`: volume-averaged ionization parameter $\mathcal{U}$
- `U0`: inner edge ionization parameter $\mathcal{U}_0$
- `Omega`: wind parameter Ω
- `zeta`: radiation pressure parameter ζ

It will also contain the following fields if the cloudy template file includes a `save last hydrogen conditions` command:

- `Hden_out`: mean H number density for the HII region structure computed by cloudy, in H/cm^3; the average is weighted by the ionized volume, i.e., it is weighted by $x\,dV$, where x is the hydrogen ionization fraction.
- `R1_out`: HII region outer radius returned by cloudy
- `Omega_out`: wind parameter Ω, computed using `R1_out` instead of `R1`
- `zeta_out`: radiation pressure parameter ζ, computing using `R1_out` instead of `R1`

If the SLUG data input to cloudy_slug were written in `ascii` mode, these data are output as a text file containing a series of columns, with different trials separated by lines of dashes.

If the SLUG data input to cloudy_slug were written in `fits` mode, the data are written in a FITS file containing a binary table extension. The table contains one column whose name corresponds to the list of fields above.

If the SLUG data input to cloudy_slug were writtin in `binary` mode, these data are written in a raw binary file that is formatted as follows. First, there are four bytes specifying if the optional fields are included:

- `Hden_out_set` (byte): 0 if the data do not include `Hden_out`, 1 if they do include it
- `R1_out_set` (byte): 0 if the data do not include `R1_out`, 1 if they do include it
- `Omega_out_set` (byte): 0 if the data do not include `Omega_out`, 1 if they do include it
- `zeta_out_set` (byte): 0 if the data do not include `zeta_out`, 1 if they do include it

This is followed by a series of records containing the following fields:

- `Trial` (numpy `uint64`)
- `Time` (numpy `float64`)
- `Hden` (numpy `float64`)
- `R0` (numpy `float64`)
- `R1` (numpy `float64`)
- `QH0` (numpy `float64`)
- `covFac` (numpy `float64`)
- `U` (numpy `float64`)
- `U0` (numpy `float64`)
- `Omega` (numpy `float64`)

- `zeta` (numpy `float64`)
- `Hden_out` (numpy `float64`; optional, only if the relevant byte in the header is set to)
- `R1_out` (numpy `float64`; optional, only if the relevant byte in the header is set to)
- `Omega_out` (numpy `float64`; optional, only if the relevant byte in the header is set to)
- `zeta_out` (numpy `float64`; optional, only if the relevant byte in the header is set to 1)

There is one such record for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial.

11.5.2 The `integrated_cloudylines` File

This file contains data on the nebular line emission produced by the interaction of the stellar radiation field with the ISM. It consists of a series of entries containing the following fields:

- `Time`: evolution time at which the output is produced
- `LineLabel`: four letter code labeling each line. These codes are the codes used by cloudy (see the cloudy documentation)
- `Wavelength`: wavelength of the line, in Angstrom. Note that default cloudy behavior is to round wavelengths to the nearest Angstrom.
- `Luminosity`: line luminosity, in erg/s

If the SLUG data input to cloudy_slug were written in `ascii` mode, these data are output as a text file containing a series of columns, with different trials separated by lines of dashes.

If the SLUG data input to cloudy_slug were written in `fits` mode, the data are written in a FITS file containing two binary table extensions. The first extension contains two fields, `Line_label` and `Wavelength`, giving the four-letter cloudy line codes and central wavelengths. The second extension contains three columns, giving the trial number, time, and line luminosity for each line at each time in each trial.

If the SLUG data input to cloudy_slug were written in `binary` mode, the data are written in a raw binary file. The file starts with a header consisting of

- `NLine` (python `int`, equivalent to C `long`): number of lines
- `LineLabel` (`NLine` entries stored as `ASCII text`): line labels listed in ASCII, one label per line

This is followed by a series of entries of the form

- `Trial` (numpy `uint64`)
- `Time` (numpy `float64`)
- `LineLum` (`NLine` entries of type numpy `float64`)

There is one such record for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial.

11.5.3 The `integrated_cloudyspec` File

This file contains data on the spectrum produced by interaction between the stellar radiation field and the nebula. Each entry in the output file contains the folling fields:

- `Trial`: which trial these data are from
- `Time`: evolution time at which the output is produced

- `Wavelength`: the wavelength at which the spectrum is evaluated, in Angstrom
- `Incident`: specific luminosity in erg/s/Angstrom at the specified wavelength. In cloudy's terminology, this is the *incident* spectrum, i.e., the stellar radiation field entering the nebula. It should be the same as the spectrum contained in the SLUG `integrated_spec` file for the corresponding time and trial, except interpolated onto the wavelength grid used by cloudy.
- `Transmitted`: specific luminosity in erg/s/Angstrom at the specified wavelength. In cloudy's terminology, this is the *transmitted* spectrum, i.e., the stellar spectrum exiting the HII region, not including any emission produced within the nebula. This is what would be detected by an observing aperture that included only the stars, and none of the nebula.
- `Emitted`: specific luminosity in erg/s/Angstrom at the specified wavelength. In cloudy's terminology, this is the *emitted* spectrum, i.e., the spectrum emitted by the diffuse gas in the HII region, excluding any light from the stars themselves. This is what would be seen by an observer whose aperture covered the nebula, but masked the stars.
- `Transmitted_plus_emitted`: this is just the sum of `Transmitted` and `Emitted`. It represents what would be observed in an aperture including both the stars and the HII region.

If the SLUG data input to cloudy_slug were written in `ascii` mode, these data are output as a text file containing a series of columns, with different trials separated by lines of dashes.

If the SLUG data input to cloudy_slug were written in `fits` mode, these data are written in a FITS file containing two binary table extensions. The first extension contains one field, `Wavelength`, which gives the wavelengths of the spectra in Angstrom. The second extension contains six fields: `Trial`, `Time`, `Incident_spectrum`, `Transmitted_spectrum`, `Emitted_spectrum`, and `Transmitted_plus_emitted_spectrum`. The first two of these give the trial number and time, and the remaining four give the incident, transmitted, emitted, and transmitted plus emitted spectra for the corresponding time and trial.

If the SLUG data input to cloudy_slug were written in `binary` mode, these data are written in a raw binary file that is formatted as follows. The file begins with a header consisting of

- `NWavelength` (numpy `int64`): number of wavelengths
- `Wavelength` (`NWavelength` entries of numpy `float64`)

and then contains a series of records of the form

- `Trial` (numpy `uint64`)
- `Time` (numpy `float64`)
- `Incident` (`NWavelength` entries of numpy `float64`)
- `Transmitted` (`NWavelength` entries of numpy `float64`)
- `Emitted` (`NWavelength` entries of numpy `float64`)
- `Transmitted_plus_emitted` (`NWavelength` entries of numpy `float64`)

There is one such record for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial.

11.5.4 The `integrated_cloudyphot` File

This file contains photometric data computed for the spectra produced by the interaction between the stellar radiation field and the HII region. The file consists of a series of entries containing the following fields:

- `Trial`: which trial these data are from
- `Time`: evolution time at which the output is computed

- `PhotFilter1_trans`: photometric value for the *Transmitted* radiation field through filter 1, where filter 1 here is the same as filter 1 in *The integrated_phot File*; units are also the same as in that file.
- `PhotFilter1_emit`: photometric value for the *Emitted* radiation field through filter 1
- `PhotFilter1_trans_emit`: photometric value for the *Transmitted_plus_emitted* radiation field through filter 1
- `PhotFilter2_trans`
- `PhotFilter2_emit`
- `PhotFilter2_trans_emit`
- `...`

For distinctions between the *Transmitted*, *Emitted*, and *Transmitted_plus_emitted* radiation fields, see *The integrated_cloudyspec File*, or the cloudy documentaiton. Note that we do not record photometry for the incident spectrum, since that would be, up to the accuracy of the numerical integration, identical to the photometry already recorded in the *The integrated_phot File*.

If the SLUG data input to cloudy_slug were written in `ascii` mode, these data are output as a text file containing a series of columns, with different trials separated by lines of dashes.

If the SLUG data input to cloudy_slug were written in `fits` mode, these data are written in a FITS file containing one binary table extension, consisting of a series of columns. The columns are `Trial`, `Time`, `Filter1_Transmitted`, `Filter1_Emitted`, `Filter1_Transmitted_plus_emitted`, `....` The first two columns give the trial number and the time, and the remainder give the photometric values for the transmitted, emitted, and transmitted plus emitted spectra in each filter.

If the SLUG data input to cloudy_slug were written in `binary` mode, these data are written to a raw binary file that is formatted as follows. The file starts with an ASCII header consisting of the following, each on a separate line:

- `NFilter` (stored as `ASCII text`): number of filters used
- `FilterName FilterUnit` (`NFilter` entries stored as `ASCII text`): the name and units for each filter are listed in ASCII, one filter-unit pair per line

This is followed by a series of entries of the form:

- `PhotFilter_Transmitted` (`NFilter` entries of numpy `float64`), giving the transmitted photometry in each filter
- `PhotFilter_Emitted` (`NFilter` entries of numpy `float64`), giving the emitted photometry in each filter
- `PhotFilter_Transmitted_plus_emitted` (`NFilter` entries of numpy `float64`), giving the transmitted plus emitted photometry in each filter

There is one such record for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial.

11.5.5 The `cluster_cloudyparams` File

This file contains the input parameters for the cloudy runs, and quantities derived from them. It consists of a series of entries containin the following fields:

- `UniqueID`: a unique identifier number for each cluster that is preserved across times and output files
- `Time`: evolution time at which the output is produced
- `Hden`: number density of hydrogen nuclei at the inner edge of the HII region whose structure cloudy computes, in H/cm^3

- `R0`: radius of the inner edge of the HII region, in cm
- `R1`: radius of the outer edge of the HII region, cm
- `QH0`: ionizing luminosity input to cloudy, in photons/s
- `CovFac`: covering factor used
- `U`: volume-averaged ionization parameter $\mathcal{U}$
- `U0`: inner edge ionization parameter $\mathcal{U}_0$
- `Omega`: wind parameter Ω
- `zeta`: radiation pressure parameter ζ

It will also contain the following fields if the cloudy template file includes a `save last hydrogen conditions` command:

- `Hden_out`: mean H number density for the HII region structure computed by cloudy, in H/cm^3; the average is weighted by the ionized volume, i.e., it is weighted by $x\,dV$, where x is the hydrogen ionization fraction.
- `R1_out`: HII region outer radius returned by cloudy
- `Omega_out`: wind parameter Ω, computed using `R1_out` instead of `R1`
- `zeta_out`: radiation pressure parameter ζ, computing using `R1_out` instead of `R1`

If the SLUG data input to cloudy_slug were written in `ascii` mode, these data are output as a text file containing a series of columns, with different trials separated by lines of dashes.

If the SLUG data input to cloudy_slug were written in `fits` mode, the data are written in a FITS file containing a binary table extension. The table contains one column whose name corresponds to the list of fields above.

If the SLUG data input to cloudy_slug were writtin in `binary` mode, these data are written in a raw binary file that is formatted as follows. First, there are four bytes specifying if the optional fields are included:

- `Hden_out_set` (byte): 0 if the data do not include `Hden_out`, 1 if they do include it
- `R1_out_set` (byte): 0 if the data do not include `R1_out`, 1 if they do include it
- `Omega_out_set` (byte): 0 if the data do not include `Omega_out`, 1 if they do include it
- `zeta_out_set` (byte): 0 if the data do not include `zeta_out`, 1 if they do include it

Next there are a series of records, one for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial. Each record consists of a header containing

- `Time` (`double`)
- `NCluster` (`std::vector<double>::size_type`, usually `unsigned long long`): number of non-disrupted clusters present at this time

This is followed by `NCluster` entries of the following form:

- `UniqueID` (numpy `uint64`)
- `Time` (numpy `float64`)
- `Hden` (numpy `float64`)
- `R0` (numpy `float64`)
- `R1` (numpy `float64`)
- `QH0` (numpy `float64`)
- `covFac` (numpy `float64`)

- `U` (numpy `float64`)
- `U0` (numpy `float64`)
- `Omega` (numpy `float64`)
- `zeta` (numpy `float64`)
- `Hden_out` (numpy `float64`; optional, only if the relevant byte in the header is set to)
- `R1_out` (numpy `float64`; optional, only if the relevant byte in the header is set to)
- `Omega_out` (numpy `float64`; optional, only if the relevant byte in the header is set to)
- `zeta_out` (numpy `float64`; optional, only if the relevant byte in the header is set to 1)

11.5.6 The `cluster_cloudylines` File

This file contains data on the nebular line emission produced by the interaction of the stellar radiation field with the ISM around each cluster. It consists of a series of entries containing the following fields:

- `UniqueID`: a unique identifier number for each cluster that is preserved across times and output files
- `Time`: evolution time at which the output is produced
- `LineLabel`: four letter code labeling each line. These codes are the codes used by cloudy (see the cloudy documentation)
- ‘‘ Wavelength‘‘: wavelength of the line, in Angstrom. Note that default cloudy behavior is to round wavelengths to the nearest Angstrom.
- `Luminosity`: line luminosity, in erg/s

If the SLUG data input to cloudy_slug were written in `ascii` mode, these data are output as a text file containing a series of columns, with different trials separated by lines of dashes.

If the SLUG data input to cloudy_slug were written in `fits` mode, the data are written in a FITS file containing two binary table extensions. The first extension contains two fields, `Line_label` and `Wavelength`, giving the four-letter cloudy line codes and central wavelengths. The second extension contains four columns, giving the unique ID, trial number, time, and line luminosity for each line at each time in each trial.

If the SLUG data input to cloudy_slug were written in `binary` mode, the data are written in a raw binary file. The file starts with a header consisting of

- `NLine` (python `int`, equivalent to C `long`): number of lines
- `LineLabel` (`NLine` entries stored as `ASCII text`): line labels listed in ASCII, one label per line

This is followed by a series of records, one for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial. Each record consists of a header containing

- `Time` (`double`)
- `NCluster` (`std::vector<double>::size_type`, usually `unsigned long long`): number of non-disrupted clusters present at this time

This is followed by `NCluster` entries of the following form:

- `UniqueID` (numpy `uint64`)
- `LineLum` (`NLine` entries of numpy `float64`)

11.5.7 The `cluster_cloudyspec` File

This file contains data on the spectra produced by the interaction of the stellar radiation field with the ISM around each cluster. It consists of a series of entries containing the following fields:

- `UniqueID`: a unique identifier number for each cluster that is preserved across times and output files
- `Time`: evolution time at which the output is produced
- `Wavelength`: observed frame wavelength at which the spectrum is evaluated
- `Incident`: specific luminosity in erg/s/Angstrom at the specified wavelength for the *incident* radiation field
- `Transmitted`: specific luminosity in erg/s/Angstrom at the specified wavelength for the *transmitted* radiation field
- `Emitted`: specific luminosity in erg/s/Angstrom at the specified wavelength for the *emitted* radiation field
- `Transmitted_plus_emitted`: specific luminosity in erg/s/Angstrom at the specified wavelength for the *transmitted plus emitted* radiation field

For explanations of the distinction between the incident, transmitted, emitted, and transmitted plus emitted radiation fields, see *The integrated_cloudyspec File*.

If the SLUG data input to cloudy_slug were written in `ascii` mode, these data are output as a text file containing a series of columns, with different trials separated by lines of dashes.

If the SLUG data input to cloudy_slug were written in `fits` mode, these data are written in a FITS file containing two binary table extensions. The first table contains a column `Wavelength` listing the wavelengths at which the spectra are given. The second table consists of seven columns: `Trial`, `UniqueID`, `Time`, `Incident_spectrum`, `Transmitted_spectrum`, `Emitted_spectrum`, and `Transmitted_plus_emitted_spectrum`. The first three of these give the trial number, unique ID of the cluster, and the time. The remaining four give the incident, transmitted, emitted, and transmitted plus emitted spectra for the corresponding cluster.

If the SLUG data input to cloudy_slug were written in `binary` mode, these data are written to a raw binary file formatted as follows. The file starts with

- `NWavelength` (numpy `int64`): the number of wavelength entries in the spectra
- `Wavelength` (`NWavelength` entries of type `double`)

and then contains a series of records, one for each output time, with different trials ordered sequentially, so that all the times for one trial are output before the first time for the next trial. Each record consists of a header containing

- `Time` (`double`)
- `NCluster` (python `int`): number of non-disrupted clusters present at this time

This is followed by `NCluster` entries of the following form:

- `UniqueID` (`unsigned long`)
- `Incident` (`NWavelength` entries of numpy `float64`)
- `Transmitted` (`NWavelength` entries of numpy `float64`)
- `Emitted` (`NWavelength` entries of numpy `float64`)
- `Transmitted_plus_emitted` (`NWavelength` entries of numpy `float64`)

11.5.8 The `cluster_cloudyphot` File

This file contains data on the photometry of the spectra produced by the interaction of the stellar radiation field with the ISM around each cluster. It consists of a series of entries containing the following fields:

- `UniqueID`: a unique identifier number for each cluster that is preserved across times and output files
- `Time`: evolution time at which the output is produced
- `PhotFilter1_trans`: photometric value for the *Transmitted* radiation field through filter 1, where filter 1 here is the same as filter 1 in *The integrated_phot File*; units are also the same as in that file.
- `PhotFilter1_emit`: photometric value for the *Emitted* radiation field through filter 1
- `PhotFilter1_trans_emit`: photometric value for the *Transmitted_plus_emitted* radiation field through filter 1
- `PhotFilter2_trans`
- `PhotFilter2_emit`
- `PhotFilter2_trans_emit`
- `...`

For distinctions between the *Transmitted*, *Emitted*, and *Transmitted_plus_emitted* radiation fields, see *The integrated_cloudyspec File*, or the cloudy documentaiton. Note that we do not record photometry for the incident spectrum, since that would be, up to the accuracy of the numerical integration, identical to the photometry already recorded in the *The cluster_phot File*.

If the SLUG data input to cloudy_slug were written in `ascii` mode, these data are output as a text file containing a series of columns, with different trials separated by lines of dashes.

If the SLUG data input to cloudy_slug were written in `fits` mode, these data are written in a FITS file containing one binary table extension. The columns in this FITS file are `Trial`, `UniqueID`, `Time`, `Filter1_Transmitted`, `Filter1_Emitted`, `Filter1_Transmitted_plus_emitted`, `....` The first three columns give the trial number, cluster unique ID, and the time, and the remainder give the photometric values for the transmitted, emitted, and transmitted plus emitted spectra in each filter.

If the SLUG data input to cloudy_slug were written in `binary` mode, these data are written in a raw binary file that is formatted as follows. The file starts with an ASCII text header consisting of the following, each on a separate line:

- `NFilter` (stored as `ASCII text`): number of filters used
- `FilterName FilterUnit` (`NFilter` entries stored as `ASCII text`): the name and units for each filter are listed in ASCII, one filter-unit pair per line

This is followed by a series of entries of that each begin with a header

- `Time` (`double`)
- `NCluster` (`std::vector<double>::size_type`, usually `unsigned long long`): number of non-disrupted clusters present at this time

This is followed by `NCluster` entries of the following form:

- `UniqueID` (`unsigned long`)
- `PhotFilter_Transmitted` (`NFilter` entries of numpy `float64`), giving the transmitted photometry in each filter
- `PhotFilter_Emitted` (`NFilter` entries of numpy `float64`), giving the emitted photometry in each filter
- `PhotFilter_Transmitted_plus_emitted` (`NFilter` entries of numpy `float64`), giving the transmitted plus emitted photometry in each filter

11.6 Full Documentation of slugpy.cloudy

slugpy.cloudy.**read_cloudy_continuum**(*filename*, *r0=None*)
Reads a cloudy continuum output, produced by save last continuum

Parameters

filename [string] name of the file to be read

r0 [float] inner radius, in cm; if included, the quantities returned will be total energies instead of energy emission rates instead of rates per unit area

Returns A namedtuple containing the following fields:

wl [array] wavelengths in Angstrom

incident [array] incident radiation field intensity

slugpy.cloudy.**read_cloudy_hcon**(*hcon_file*, *r0=0.0*)
Reads cloudy outputs produce by the 'save last hydrogen conditions' and 'save last hydrogen ionization' file, and uses these to return various HII region diagnostic parameters.

Parameters

hcon_file [str] Name of hydrogen conditions file to be read

r0 [float] Inner radius for the calculation

Returns

r1 [float] outer radius, in cm

nII [float] average number density of H nuclei

Omega [float] wind parameter, defined as r0^3 / (r1^3 - r0^3)

Notes All averages are averages over the ionized volume, i.e., the average is taken with a weighting factor 4 pi r^2 (n_H+/n_H) dV.

slugpy.cloudy.**read_cloudy_linelist**(*filename*)
Reads a cloudy line list output, produced by save last line list

Parameters

filename [string] name of the file to be read

Returns A namedtuple containing the following fields:

labels [array, dtype 'S4'] list of line labels

wl [array] array of line wavelengths, in Angstrom

lum [array] array of line luminosities; this will be in whatever units the cloudy output is in

slugpy.cloudy.**read_cluster_cloudyparams**(*model_name*, *output_dir=None*, *fmt=None*, *verbose=False*, *read_info=None*)
Function to read a SLUG2 cluster_cloudyparams file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt ['txt' | 'ascii' | 'bin' | 'binary' | 'fits' | 'fits2'] Format for the file to be read. If one of these is set, the function will only attempt to open ASCII-('txt' or 'ascii'), binary ('bin' or 'binary'), or FITS ('fits' or 'fits2') formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

Returns A namedtuple containing the following fields:

id [array, dtype uint] unique ID of cluster

trial: array, dtype uint which trial was this cluster part of

time [array] time at which cluster's properties are being evaluated

cloudy_hden [array] number density of H nuclei at the inner edge of the ionized region simulated by cloudy

cloudy_r0 [array] inner radius of the ionized region simulated by cloudy

cloudy_r1 [array] outer radius of the ionized region simulated by cloudy (approximate!)

cloudy_QH0 [array] ionizing luminosity used in the cloudy computation

cloudy_covFac [array] covering factor assumed in the cloudy computation; only a fraction covFac of the ionizing photons are assumed to produce emission within the HII region, while the remainder are assumed to escape

cloudy_U [array] volume-averaged ionization parameter of the HII region simulated by cloudy

cloudy_U0 [array] ionization parameter of the HII reegion at its inner edge

cloudy_Omega [array] Yeh & Matzner (2012) wind parameter for the HII region simulated by cloudy

cloudy_zeta [array] Krumholz & Matzner (2009) radiation pressure parameter for the HII region, again approximate; values of zeta >~1 indicate that radiation pressure is dynamically important

The following fields may or may not be present, depending on what is found in the output file:

cloudy_hden_out [array] volume-averaged number density produced by the cloudy calculation

cloudy_r1_out [array] HII region outer radius produced by cloudy

cloudy_Omega_out [array] value of Omega computed using cloudy_r1_out

cloudy_zeta_out [array] value of zeta computed using cloudy_r1_out

Notes

The relationships between U, U0, Omega, r0, r1, hden, and QH0 used in deriving various parameters are valid only in the limit of negligible radiation pressure. They may be significantly off if radiation pressure is significant, i.e., if zeta >~ 1.

slugpy.cloudy.**read_cluster_cloudyphot**(*model_name*, *output_dir=None*, *fmt=None*, *nofilterdata=False*, *photsystem=None*, *verbose=False*, *read_info=None*)

Function to read a SLUG2 cluster_cloudyphot file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt [string] Format for the file to be read. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'. If one of these is set, the code will only attempt to open ASCII-, binary-, or FITS-formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

nofilterdata [bool] If True, the routine does not attempt to read the filter response data from the standard location

photsystem [None or string] If photsystem is None, the data will be returned in the same photometric system in which they were read. Alternately, if it is a string, the data will be converted to the specified photometric system. Allowable values are 'L_nu', 'L_lambda', 'AB', 'STMAG', and 'Vega', corresponding to the options defined in the SLUG code. If this is set and the conversion requested involves a conversion from a wavelength-based system to a frequency-based one, nofilterdata must be False so that the central wavelength of the photometric filters is available.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

Returns A namedtuple containing the following fields:

id [array, dtype uint] unique ID of cluster

trial: array, dtype uint which trial was this cluster part of

time [array] times at which cluster spectra are output, in yr

cloudy_filter_names [list of string] a list giving the name for each filter

cloudy_filter_units [list of string] a list giving the units for each filter

cloudy_filter_wl_eff [list] effective wavelength of each filter; this is set to None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

cloudy_filter_wl [list of arrays] a list giving the wavelength table for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

cloudy_filter_response [list of arrays] a list giving the photon response function for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

cloudy_filter_beta [list] powerlaw index beta for each filter; used to normalize the photometry

cloudy_filter_wl_c [list] pivot wavelength for each filter; used to normalize the photometry

cloudy_phot_trans [array, shape (N_cluster, N_filter)] photometric value for each cluster in each filter for the transmitted light (i.e., the starlight remaining after it has passed through the HII region); units are as indicated in the units field

cloudy_phot_emit [array, shape (N_cluster, N_filter)] photometric value for each cluster in each filter for the emitted light (i.e., the diffuse light emitted by the HII region); units are as indicated in the units field

cloudy_phot_trans_emit [array, shape (N_cluster, N_filter)] photometric value in each filter for each cluster for the transmitted plus emitted light (i.e., the light coming directly from the stars after absorption by the HII region, plus the diffuse light emitted by the HII region); units are as indicated in the units field

Raises IOError, if no photometry file can be opened; ValueError, if photsystem is set to an unknown value

`slugpy.cloudy.`**`read_cluster_cloudylines`**(*model_name*, *output_dir=None*, *fmt=None*, *verbose=False*, *read_info=None*)

Function to read a SLUG2 cluster_cloudylines file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt [string] Format for the file to be read. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'. If one of these is set, the code will only attempt to open ASCII-, binary-, or FITS-formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

Returns A namedtuple containing the following fields:

id [array, dtype uint] unique ID of cluster

trial: array, dtype uint which trial was this cluster part of

time [array] times at which cluster spectra are output, in yr

cloudy_linelabel [array, dtype='S4', shape (N_lines)] labels for the lines, following cloudy's 4 character line label notation

cloudy_linewl [array, shape (N_lines)] rest wavelength for each line, in Angstrom

cloudy_linelum [array, shape (N_cluster, N_lines)] luminosity of each line at each time for each trial, in erg/s

slugpy.cloudy.**read_cluster_cloudyspec**(*model_name*, *output_dir=None*, *fmt=None*, *verbose=False*, *read_info=None*)

Function to read a SLUG2 cluster_cloudyspec file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt [string] Format for the file to be read. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'. If one of these is set, the code will only attempt to open ASCII-, binary-, or FITS-formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

Returns A namedtuple containing the following fields:

id [array, dtype uint] unique ID of cluster

trial: array, dtype uint which trial was this cluster part of

time [array] times at which cluster spectra are output, in yr

cloudy_wl [array] wavelength, in Angstrom

cloudy_inc [array, shape (N_cluster, N_wavelength)] specific luminosity of the cluster's stellar radiation field at each wavelength, in erg/s/A

cloudy_trans [array, shape (N_cluster, N_wavelength)] specific luminosity of the stellar radiation field after it has passed through the HII region, at each wavelength, in erg/s/A

cloudy_emit [array, shape (N_cluster, N_wavelength)] specific luminosity of the radiation field emitted by the HII region, at each wavelength, in erg/s/A

cloudy_trans_emit [array, shape (N_cluster, N_wavelength)] the sum of the emitted and transmitted fields; this is what would be seen by an observer looking at both the star cluster and its nebula

Raises IOError, if no spectrum file can be opened

slugpy.cloudy.**read_integrated_cloudylines**(*model_name*, *output_dir=None*, *fmt=None*, *verbose=False*, *read_info=None*)

Function to read a SLUG2 integrated_cloudylines file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt [string] Format for the file to be read. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'. If one of these is set, the code will only attempt to open ASCII-, binary-, or FITS-formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

Returns A namedtuple containing the following fields:

time [array, shape (N_times) or shape (N_trials)] Times at which data are output; shape is either N_times (if the run was done with fixed output times) or N_trials (if the run was done with random output times)

cloudy_linelabel [array, dtype='S4', shape (N_lines)] labels for the lines, following cloudy's 4 character line label notation

cloudy_linewl [array, shape (N_lines)] rest wavelength for each line, in Angstrom

cloudy_linelum [array, shape (N_lines, N_times, N_trials)] luminosity of each line at each time for each trial, in erg/s

slugpy.cloudy.**read_integrated_cloudyparams**(*model_name*, *output_dir=None*, *fmt=None*, *verbose=False*, *read_info=None*)

Function to read a SLUG2 integrated_cloudyparams file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt [string] Format for the file to be read. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'. If one of these is set, the code will only attempt to open ASCII-, binary-, or FITS-formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

Returns A namedtuple containing the following fields:

time [array, shape (N_times) or shape (N_trials)] Times at which data are output; shape is either N_times (if the run was done with fixed output times) or N_trials (if the run was done with random output times)

cloudy_hden [array] number density of H nuclei at the inner edge of the ionized region simulated by cloudy

cloudy_r0 [array] inner radius of the ionized region simulated by cloudy

cloudy_r1 [array] outer radius of the ionized region simulated by cloudy (approximate!)

cloudy_QH0 [array] ionizing luminosity used in the cloudy computation

cloudy_covFac [array] covering factor assumed in the cloudy computation; only a fraction covFac of the ionizing photons are assumed to produce emission within the HII region, while the remainder are assumed to escape

cloudy_U [array] volume-averaged ionization parameter of the HII region simulated by cloudy

cloudy_U0 [array] ionization parameter of the HII reegion at its inner edge

cloudy_Omega [array] Yeh & Matzner (2012) wind parameter for the HII region simulated by cloudy

cloudy_zeta [array] Krumholz & Matzner (2009) radiation pressure parameter for the HII region, again approximate; values of zeta >~1 indicate that radiation pressure is dynamically important

The following fields may or may not be present, depending on what is found in the output file:

cloudy_hden_out [array] volume-averaged number density produced by the cloudy calculation

cloudy_r1_out [array] HII region outer radius produced by cloudy

cloudy_Omega_out [array] value of Omega computed using cloudy_r1_out

cloudy_zeta_out [array] value of zeta computed using cloudy_r1_out

Notes

The relationships between U, U0, Omega, r0, r1, hden, and QH0 used in deriving various parameters are valid only in the limit of negligible radiation pressure. They may be significantly off if radiation pressure is significant, i.e., if zeta >~ 1.

slugpy.cloudy.**read_integrated_cloudyphot**(*model_name*, *output_dir=None*, *fmt=None*, *nofilterdata=False*, *photsystem=None*, *verbose=False*, *read_info=None*)

Function to read a SLUG2 integrated_cloudyphot file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt [string] Format for the file to be read. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'. If one of these is set, the code will only attempt to open ASCII-, binary-, or FITS-formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

nofilterdata [bool] If True, the routine does not attempt to read the filter response data from the standard location

photsystem [None or string] If photsystem is None, the data will be returned in the same photometric system in which they were read. Alternately, if it is a string, the data will be converted to the specified photometric system. Allowable values are 'L_nu', 'L_lambda', 'AB', 'STMAG', and 'Vega', corresponding to the options defined in the SLUG code. If this is set and the conversion requested involves

a conversion from a wavelength-based system to a frequency-based one, nofilterdata must be False so that the central wavelength of the photometric filters is available.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

Returns A namedtuple containing the following fields:

time [array, shape (N_times) or shape (N_trials)] Times at which data are output; shape is either N_times (if the run was done with fixed output times) or N_trials (if the run was done with random output times)

cloudy_filter_names [list of string] a list giving the name for each filter

cloudy_filter_units [list of string] a list giving the units for each filter

cloudy_filter_wl_eff [list] effective wavelength of each filter; this is set to None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

cloudy_filter_wl [list of arrays] a list giving the wavelength table for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

cloudy_filter_response [list of arrays] a list giving the photon response function for each filter; this is None for the filters Lbol, QH0, QHe0, and QHe1; omitted if nofilterdata is True

cloudy_filter_beta [list] powerlaw index beta for each filter; used to normalize the photometry

cloudy_filter_wl_c [list] pivot wavelength for each filter; used to normalize the photometry

cloudy_phot_trans [array, shape (N_filter, N_times, N_trials)] photometric value in each filter at each time in each trial for the transmitted light (i.e., the starlight remaining after it has passed through the HII region); units are as indicated in the units field

cloudy_phot_emit [array, shape (N_filter, N_times, N_trials)] photometric value in each filter at each time in each trial for the emitted light (i.e., the diffuse light emitted by the HII region); units are as indicated in the units field

cloudy_phot_trans_emit [array, shape (N_filter, N_times, N_trials)] photometric value in each filter at each time in each trial for the transmitted plus emitted light (i.e., the light coming directly from the stars after absorption by the HII region, plus the diffuse light emitted by the HII region); units are as indicated in the units field

Raises IOError, if no photometry file can be opened; ValueError, if photsystem is set to an unknown value

`slugpy.cloudy.`**`read_integrated_cloudyspec`**(*model_name*, *output_dir=None*, *fmt=None*, *verbose=False*, *read_info=None*)

Function to read a SLUG2 integrated_cloudyspec file.

Parameters

model_name [string] The name of the model to be read

output_dir [string] The directory where the SLUG2 output is located; if set to None, the current directory is searched, followed by the SLUG_DIR directory if that environment variable is set

fmt [string] Format for the file to be read. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'. If one of these is set, the code will only attempt to open ASCII-, binary-, or FITS-formatted output, ending in .txt., .bin, or .fits, respectively. If set to None, the code will try to open ASCII files first, then if it fails try binary files, and if it fails again try FITS files.

verbose [bool] If True, verbose output is printed as code runs

read_info [dict] On return, this dict will contain the keys 'fname' and 'format', giving the name of the file read and the format it was in; 'format' will be one of 'ascii', 'binary', or 'fits'

Returns A namedtuple containing the following fields:

time [array, shape (N_times) or shape (N_trials)] Times at which data are output; shape is either N_times (if the run was done with fixed output times) or N_trials (if the run was done with random output times)

cloudy_wl [array] wavelength, in Angstrom

cloudy_inc [array, shape (N_wavelength, N_times, N_trials)] specific luminosity of the stellar radiation field at each wavelength and each time for each trial, in erg/s/A

cloudy_trans [array, shape (N_wavelength, N_times, N_trials)] specific luminosity of the stellar radiation field after it has passed through the HII region, at each wavelength and each time for each trial, in erg/s/A

cloudy_emit [array, shape (N_wavelength, N_times, N_trials)] specific luminosity of the radiation field emitted by the HII region, at each wavelength and each time for each trial, in erg/s/A

cloudy_trans_emit [array, shape (N_wavelength, N_times, N_trials)] the sum of emitted and transmitted; this is what would be seen by an observer looking at both the star cluster and its nebula

slugpy.cloudy.**write_cluster_cloudyparams**(*data*, *model_name*, *fmt*)

Write out photometry for nebular emission computed by cloudy on a slug spectrum for a series of clusters

Parameters

data [namedtuple] Cluster cloudy parameter data; a namedtuple containing the fields id, trial, time, cloudy_hden, cloudy_r0, cloudy_r1, cloudy_QH0, cloudy_covFac, cloudy_U, cloudy_U0, cloudy_Omega, and cloudy_zeta; may also optionally contain the fields cloudy_r1_out, cloudy_hden_out, cloudy_Omega_out, and cloudy_zeta_out

model_name [string] Base file name to give the model to be written. Can include a directory specification if desired.

fmt [string] Format for the output file. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'.

Returns Nothing

slugpy.cloudy.**write_cluster_cloudyphot**(*data*, *model_name*, *fmt*)

Write out photometry for nebular emission computed by cloudy on a slug spectrum for a series of clusters

Parameters

data [namedtuple] Cluster cloudy photometry data to be written; a namedtuple containing the fields id, time, cloudy_filter_names, cloudy_filter_units, cloudy_phot_trans, cloudy_phot_emit, and cloudy_phot_trans_emit

model_name [string] Base file name to give the model to be written. Can include a directory specification if desired.

fmt [string] Format for the output file. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'.

Returns Nothing

slugpy.cloudy.**write_cluster_cloudylines**(*data*, *model_name*, *fmt*)

Write out data computed by cloudy on a slug spectrum

Parameters

data [namedtuple] Cloudy spectral data for clusters to be written; a namedtuple containing the fields time, cloudy_linelist, cloudy_linewl, cloudy_linelum

model_name [string] Base file name to give the model to be written. Can include a directory specification if desired.

fmt [string] Format for the output file. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'.

Returns Nothing

slugpy.cloudy.**write_cluster_cloudyspec**(*data*, *model_name*, *fmt*)
Write out data computed by cloudy on a slug spectrum

Parameters

data [namedtuple] Cloudy spectral data for clusters to be written; a namedtuple containing the fields id, time, cloudy_wl, cloudy_inc, cloudy_trans, cloudy_emit, and cloudy_trans_emit

model_name [string] Base file name to give the model to be written. Can include a directory specification if desired.

fmt [string] Format for the output file. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'.

Returns Nothing

slugpy.cloudy.**write_integrated_cloudyparams**(*data*, *model_name*, *fmt*)
Write out photometry for nebular emission computed by cloudy on a slug spectrum for a series of clusters

Parameters

data [namedtuple] Cluster cloudy parameter data; a namedtuple containing the fields time, cloudy_hden, cloudy_r0, cloudy_r1, cloudy_QH0, cloudy_covFac, cloudy_U, cloudy_U0, cloudy_Omega, and cloudy_zeta; may also optionally contain the fields cloudy_r1_out, cloudy_hden_out, cloudy_Omega_out, and cloudy_zeta_out

model_name [string] Base file name to give the model to be written. Can include a directory specification if desired.

fmt [string] Format for the output file. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'.

Returns Nothing

slugpy.cloudy.**write_integrated_cloudylines**(*data*, *model_name*, *fmt*)
Write out line luminosities computed by cloudy on a slug spectrum

Parameters

data [namedtuple] Integrated cloudy line data to be written; a namedtuple containing the fields time, cloudy_linelist, cloudy_linewl, cloudy_linelum

model_name [string] Base file name to give the model to be written. Can include a directory specification if desired.

fmt [string] Format for the output file. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'.

Returns Nothing

slugpy.cloudy.**write_integrated_cloudyphot**(*data*, *model_name*, *fmt*)
Write out photometry for nebular emission computed by cloudy on a slug spectrum

Parameters

data [namedtuple] Integrated cloudy photometry data to be written; a namedtuple containing the fields time, cloudy_filter_names, cloudy_filter_units, cloudy_phot_trans, cloudy_phot_emit, and cloudy_phot_trans_emit

model_name [string] Base file name to give the model to be written. Can include a directory specification if desired.

fmt [string] Format for the output file. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'.

Returns Nothing

slugpy.cloudy.**write_integrated_cloudyspec**(*data*, *model_name*, *fmt*)

Write out data computed by cloudy on a slug spectrum

Parameters

data [namedtuple] Integrated cloudy spectral data to be written; a namedtuple containing the field time, cloudy_wl, cloudy_inc, cloudy_trans, cloudy_emit, and cloudy_trans_emit

model_name [string] Base file name to give the model to be written. Can include a directory specification if desired.

fmt [string] Format for the output file. Allowed values are 'ascii', 'bin' or 'binary, and 'fits'.

Returns Nothing

CHAPTER

TWELVE

BAYESPHOT: BAYESIAN INFERENCE FOR STOCHASTIC STELLAR POPULATIONS

12.1 What Does bayesphot Do?

Bayesphot is a package for performing Bayesian inference for the physical properties of a stellar system using its measured photometric properties, in a case where the photometric properties vary non-deterministically with the physical properties. Formally, bayesphot answers the following question: consider a stellar system characterized by a vector of $\mathbf{x} = (x_1, x_2, \ldots x_N)$ physical properties. We have a physical model that lets us sample the expected photometric properties as a function of physical properties, i.e., that for some sample of K systems with physical properties $\mathbf{x}_k$ we are able to compute the corresponding photometric properties $\mathbf{y}_k = \mathbf{y} = (y_1, y_2, \ldots y_M)_k$. Now suppose that we observe such a system, and we observe it to have photometric properties $\mathbf{y}_{\rm obs}$, with some set of photometric errors $\sigma_{\mathbf{y}} = (\sigma_{y_1}, \sigma_{y_2}, \ldots \sigma_{y_M})$, which are assumed to be Gaussian-distributed. What should we infer about the posterior probability distribution of the physical properties, i.e., given a set of prior probabilities $p(\mathbf{x})$, plus our measurements, what is $p(\mathbf{x} \mid \mathbf{y}_{\rm obs}, \sigma_{\rm y})$?

The kernel density estimation algorithm that bayesphot uses to answer this question is described and derived in the slug methods paper. Bayesphot is implemented in two parts: a shared object library that is implemented in c, and that is compiled at the same time that slug is built, and a python wrapper class called `bp` that is included in the slugpy.bayesphot module. The following sections describe how to use `bp` objects to generate posterior PDFs.

12.2 Creating `bp` Objects

The `bp` class can be imported via:

```
from slugpy.bayesphot import *
```

or:

```
from slugpy.bayesphot import bp
```

Once imported, a `bp` object can be instantiated. The call signature for the `bp` constructor class is:

```
def __init__(self, dataset, nphys, filters=None, bandwidth='auto',
             ktype='gaussian', priors=None, sample_density=None,
             reltol=1.0e-3, abstol=1.0e-10, leafsize=16):
```

A full description of all options is included in the *Full Documentation of slugpy.bayesphot*, but the essential features are summarized here.

The argument `dataset` is an array of shape (N, M) that contains the library of N models that represents the training set for the Bayesian analysis. Each model consists of M properties; the first `nphys` of these are physical properties

that are the quantities to be inferred from the observations, while the remaining ones are photometric properties. An important point is that the `dataset` object is NOT copied, so altering it after the `bp` object is created will result in erroneous results.

The `priors` and `sample_density` arguments are used to compute the weighting to apply to the input models. The `priors` argument specifies the prior probability to assign to each model; it can be either an array giving a prior probability directly, or a callable that can take the physical properties of models as an input and return the prior probability as an output. Similarly, the `sample_density` argument specifies the probablity distribution from which the physical models were selected; as with `priors`, it can be an array or a callable.

The `bandwidth` argument specifies the bandwidth to use in the kernel density estimation; this need not be the same in each dimension. The `bandwidth` can be specified as a float, in which case it is the same for every dimension, or as an array of M elements giving the bandwidth for every dimension. Finally, it can be set to the string `auto`, in which case the `bp` will attempt to make a reasonable choice of bandwidth autonomously. However, this autonomous choice will probably perform less well than something that is hand-chosen by the user based on their knowledge of the library. As a rule of thumb, bandwidths should be chosen so that, for typical input photometric values, there are ~10 simulations within the 1 kernel size.

Note that both `priors` and `bandwidth` are properties of the `bp` class, and can be altered after the `bp` object is created. This makes it possible to alter the priors and bandwidth without incurring the computational or memory cost of generating an entirely new `bp` object.

12.3 Using `bp` Objects

Once a `bp` object is instantiated, it can be used to compute likelihood functions, marginal probabilities, and MCMC sample ensembles, and to search the library for the best matches to an input set of photometry.

The likelihood function is implemented via the `bp.logL` method, which has the call signature:

```
def logL(self, physprop, photprop, photerr=None):
```

The argument `physprop` is a set of physical properties, the argument `photprop` is a set of photometric properties, and the argument `photerr` is an (optional) set of photometric errors. All of these must be arrays, the size of whose trailing dimension matches the number of physical properties (for `physprop`) or the number of photometric properties (for `photprop` and `photerr`); the leading dimensions of these arrays are broadcast together using normal broadcasting rules. The quantity returned is the log of the joint probability distribution of physical and photometric properties. Specifically, the quantity returned for each input set of physical and photometric properties is

$$\log p(\mathbf{x}, \mathbf{y}, \sigma_{\mathbf{y}}) = \log A \sum_{i=1}^{N} w_i G(\mathbf{x}, \mathbf{y}; \mathbf{h}')$$

where A is a normalization constant chosen to ensure that the PDF integrated over all space is unity, $\mathbf{x}$ is the vector of physical properties, $\mathbf{y}$ is the vector of photometric properties, $\sigma_{\mathbf{y}}$ is the vector of photomtric errors, w_i is the weight of the ith model as determined by the priors and sample density,

$$G(\mathbf{x}, \mathbf{y}; \mathbf{h}') \propto \exp\left[-\left(\frac{x_1^2}{2h_{x_1}'^2} + \cdots + \frac{x_N^2}{2h_{x_N}'^2} + \frac{y_1^2}{2h_{y_1}'^2} + \cdots + \frac{y_M^2}{2h_{y_M}'^2}\right)\right]$$

is the N-dimensional Gaussian function, and

$$\mathbf{h}' = \sqrt{\mathbf{h}^2 + \sigma_{\mathbf{y}}^2}$$

is the modified bandwidth, which is equal to the bandwidth used for kernel density estimation added in quadrature sum with the errors in the photometric quantities (see the slug method paper for details).

Estimation of marginal PDFs is done via the `bp.mpdf` method, which has the call signature:

```
def mpdf(self, idx, photprop, photerr=None, ngrid=128,
         qmin=None, qmax=None, grid=None, norm=True):
```

The argument `idx` is an int or a list of ints between 0 and nphys-1, which specifies for which physical quantity or physical quantities the marginal PDF is to be computed. These indices refer to the indices in the `dataset` array that was input when the `bp` object was instantiated. The arguments `photprop` and `photerr` give the photometric measurements and their errors for which the marginal PDFs are to be computed; they must be arrays whose trailing dimension is equal to the number of photometric quantities. The leading dimensions of these arrays are broadcast together following the normal broadcasting rules. By default each physical quantity will be estimated on a grid of 128 points, evenly spaced from the lowest value of that physical property in the model library to the highest value. The parameters `qmin`, `qmax`, `ngrid`, and `grid` can be used to override this behavior and set the grid of evaluation points manually. The function returns a tuple `grid_out, pdf`; here `grid_out` is the grid of points on which the marginal PDF has been computed, and `pdf` is the value of the marginal PDF evaluated at those gridpoints.

MCMC calculations are implemented via the method `bp.mcmc`; this method relies on the emcee python module, and will only function if it is installed. The call signature is:

```
def mcmc(self, photprop, photerr=None, mc_walkers=100,
         mc_steps=500, mc_burn_in=50):
```

The quantities `photprop` and `photerr` have the same meaning as for `bp.mpdf`, and the quantities `mc_walkers`, `mc_steps`, and `mc_burn_in` are passed directly to `emcee`, and are described in emcee's documentation. The quantity returned is an array of sample points computed by the MCMC; its format is also described in emcee's documentation. Note that, although `bp.mcmc` can be used to compute marginal PDFs of the physical quantities, for marginal PDFs of 1 quantity or joint PDFs of 2 quantities it is almost always faster to use `bp.mpdf` than `bp.mcmc`. This is because `bp.mpdf` takes advantage of the fact that integrals of cuts through N-dimensional Gaussians can be integrated analytically to compute the marginal PDFs directly, though needing to evaluate the likelihood function point by point. In contrast, the general MCMC algorithm used by `emcee` effectively does the integral numerically.

The `bp.bestmatch` method searches through the model library and finds the N library entries that are closest to an input set of photometry. The call signature is:

```
def bestmatch(self, phot, nmatch=1, bandwidth_units=False):
```

Here `phot` is the set of photometric properties, which is identical to the `photprop` parameter used by `logL`, `mpdf`, and `mcmc`. The argument `nmatch` specifies how many matches to return, and the argument `bandwidth_units` specifies whether distances are to be measured using an ordinary Euclidean metric, or in units of the kernel bandwidth in a given direction. The function returns, for each input set of photometry, the physical and photometric properties of the `nmatch` models in the library that are closest to the input photometric values. This can be used to judge if a good match to the input photometry is present in the library.

12.4 Caching

The `bp` class is built to compute posterior PDFs of some quantities, marginalising over others. To speed this process, it uses an internal KD tree representation of the data. By default the KD tree spans all physical and photometric dimensions of the underlying data set. When marginalising over some dimensions, however, it can be much more efficient to have a KD tree that only spans the dimensions that are not being marginalised, particularly if there are many of them. For example, suppose that we have a library of sample star clusters with a range of masses, ages, extinctions, metallicities, and photometric measurements. We are interested in the posterior PDF of mass given a set of photometry, marginalising over age, extinction, and metallicity. In this case it is more efficient to have a KD tree that does not split in the age, extinction, or metallicity dimensions. Since evaluating the marginal posterior PDF of mass using such a tree is much faster, if we are going to compute marginal posterior PDFs of mass many times (for example for many photometric measurements) it is advantageous to pay the one-time cost of constructing the better-optimised KD tree and then use it for all subsequent calculations.

To handle this, `bp` has an optional caching capability that will cache KD tree representations of the data that are optimal for computing posterior PDFs marginalised over certain dimensions. One can construct such a cached tree by invoking the `bp.make_cache` method. The syntax is simple:

```
bp.make_cache(margindims)
```

where `margindims` is a listlike containing the dimensions that will be marginalized out. Once created, the cache will be used for all subsequent evaluations using `bp.mpdf` and related methods (`mpdf_phot` and `mpdf_gen`) that marginalise over those dimensions.

Caching can also be done automatically. The `bp` constructor accepts the keyword `caching` which specifies the type of caching to use. The default value, `none`, uses no caching. The value `lazy` causes a cached KD tree to be build whenever one of the marginal PDF methods is invoked, and is stored for future use. (Warning: `lazy` mode is not thread-safe – see *Parallelism in bayesphot*.) Finally, `aggressive` mode constructs caches for all possible one-dimensional marginalisations of physical variables given observed photometry, and all possible one-dimensional marginalisations of variables by themselves, when the `bp` object is first constructed.

12.5 Parallelism in bayesphot

The `bp` class supports parallel calculations of posterior PDFs and related quantities, through the python multiprocessing module. This allows efficient use of multiple cores on a shared memory machine, circumventing the python global interpreter lock, without the need for every process to read a large simulation library or store it in memory. The recommended method for writing threaded code using `bp` objects is to use have a master process create the `bp` object, and then use a Process or Pool object to create child processes the perform computations using `bp` methods such as `bp.logL` or `bp.mpdf`. It is often most efficient to combine this with shared memory objects such as RawArray to hold the outputs.

An example use case for computing 1D marginal PDFs on a large set of photometric values is:

```
# Import what we need
from slugpy.bayesphot import bp
from multiprocessing import Pool, RawArray
from ctypes import c_double
import numpy as np

# Some code here to create / read the data set to be used by
# bayesphot and store it in a variable called dataset

# Create the bayesphot object
my_bp = bp(dataset, nphys)

# Some code here to create / read the photometric data we want to
# process using bayesphot and store it in an array called phot,
# which is of shape (nphot, nfilter). There is also an array of
# photometric errors, called photerr, of the same shape.

# Create a holder for the output
pdf_base = RawArray(c_double, 128*nphot)
pdf = np.frombuffer(pdf_base, dtype=c_double). \
      reshape((nphot,128))
grd_base = RawArray(c_double, 128)
grd = np.frombuffer(grd_base, dtype=c_double)

# Define the function that will be responsible for computing the
# marginal PDF
def mpdf_apply(i):
```

```
    grd[:], pdf[i] = my_bp.mpdf(0, phot[i], photerr[i])

# Main thread starts up a process pool and starts the computation
if __name__ == '__main__':
    pool = Pool()
    pool.map(mpdf_apply, range(nphot))

    # At this point the grd and PDF contain the same as they would
    # if we had done
    #     grd, pdf = my_bp.mpdf(0, phot, photerr)
    # but the results will be computed much faster this way
```

For an example of a more complex use case, see the LEGUS cluster pipeline.

The full list of `bp` methods that are thread-safe is:

- `bp.logL`
- `bp.mpdf`
- `bp.mcmc`
- `bp.bestmatch`
- `bp.make_approx_phot`
- `bp.make_approx_phys`
- `bp.squeeze_rep`
- `bp.mpdf_approx`

Thread safety involves a very modest overhead in terms of memory and speed, but for non-threaded computations this can be avoided by specifying:

```
bp.thread_safe = False
```

or by setting the `thread_safe` keyword to `False` when the `bp` object is constructed.

Finally, note that this parallel paradigm avoids duplicating the large library only on unix-like operating systems that support copy-on-write semantics for fork. Code such as the above example should still work on windows (though this has not been tested), but each worker process will duplicate the library in physical memory, thereby removing one of the main advantages of working in parallel.

12.6 Full Documentation of slugpy.bayesphot

This defines a class that can be used to estimate the PDF of physical quantities from a set of input photometry in various bands, together with a training data set.

class slugpy.bayesphot.bp.**bp**(*dataset*, *nphys*, *filters=None*, *bandwidth='auto'*, *ktype='gaussian'*, *priors=None*, *pobs=None*, *sample_density=None*, *reltol=0.01*, *abstol=1e-10*, *leafsize=16*, *nosort=None*, *thread_safe=True*, *caching='none'*)

A class that can be used to estimate the PDF of the physical properties of stellar population from a training set plus a set of measured photometric values.

Properties

priors [array, shape (N) | callable | None] prior probability on each data point; interpretation depends on the type passed; array, shape (N): values are interpreted as the prior probability of each data point;

callable: the callable must take as an argument an array of shape (N, nphys), and return an array of shape (N) giving the prior probability at each data point; None: no reweighting is performed, so all data points in the library have equal prior probability, i.e. the prior is the same as the sampling of points in the library

pobs [array, shape (N) | callable | None] the probability that a particular object would be observed, which is used, like prior, to weight the library; interpretation depends on type. None means all objects are equally likely to be observed, array is an array giving the observation probability of each object in the library, and callable means must be a function that takes an array containing the photometry, of shape (N, nhpot), as an argument, and returns an array of shape (N) giving the probability of observation for that object

bandwidth ['auto' | float | array, shape (M)] bandwidth for kernel density estimation; if set to 'auto', the bandwidth will be estimated automatically; if set to a scalar quantity, the same bandwidth is used for all dimensions

nphys [int] number of physical properties in the library

nphot [int] number of photometric properties in the library

ndim [int] nphys + nphot

__init__(*dataset*, *nphys*, *filters=None*, *bandwidth='auto'*, *ktype='gaussian'*, *priors=None*, *pobs=None*, *sample_density=None*, *reltol=0.01*, *abstol=1e-10*, *leafsize=16*, *nosort=None*, *thread_safe=True*, *caching='none'*)

Initialize a bp object.

Parameters

dataset [array, shape (N, M)] training data set; this is a set of N sample stellar populations, having M properties each; the first nphys represent physical properties (e.g., log mass, log age), while the next M - nphys represent photometric properties

nphys [int] number of physical properties in dataset

filters [listlike of strings] names of photometric filters; not used, but can be stored for convenience

bandwidth ['auto' | float | array, shape (M)] bandwidth for kernel density estimation; if set to 'auto', the bandwidth will be estimated automatically; if set to a scalar quantity, the same bandwidth is used for all dimensions

ktype [string] type of kernel to be used in densty estimation; allowed values are 'gaussian' (default), 'epanechnikov', and 'tophat'; only Gaussian can be used with error bars

priors [array, shape (N) | callable | None] prior probability on each data point; interpretation depends on the type passed; array, shape (N): values are interpreted as the prior probability of each data point; callable: the callable must take as an argument an array of shape (N, nphys), and return an array of shape (N) giving the prior probability at each data point; None: all data points have equal prior probability

pobs [array, shape (N) | callable | None] the probability that a particular object would be observed, which is used, like prior, to weight the library; interpretation depends on type. None means all objects are equally likely to be observed, array is an array giving the observation probability of each object in the library, and callable means must be a function that takes an array containing the photometry, of shape (N, nhpot), as an argument, and returns an array of shape (N) giving the probability of observation for that object

sample_density [array, shape (N) | callable | 'auto' | None] the density of the data samples at each data point; this need not match the prior density; interpretation depends on the type passed; array, shape (N): values are interpreted as the density of data sampling at each sample point; callable: the callable must take as an argument an array of shape (N, nphys), and return an array of shape (N) giving the sampling density at each point; 'auto': the sample density will be computed directly

from the data set; note that this can be quite slow for large data sets, so it is preferable to specify this analytically if it is known; None: data are assumed to be uniformly sampled

reltol [float] relative error tolerance; errors on all returned probabilities p will satisfy either abs(p_est - p_true) <= reltol * p_est OR abs(p_est - p_true) <= abstol, where p_est is the returned estimate and p_true is the true value

abstol [float] absolute error tolerance; see above

leafsize [int] number of data points in each leaf of the KD tree

nosort [arraylike of bool, shape (N) | None] if specified, this keyword causes the KD tree not to be sorted along the dimensions for which nosort is True

thread_safe [bool] if True, bayesphot will make extra copies of internals as needed to ensure thread safety when the computation routines (logL, mpdf, mcmc, bestmatch, make_approx_phot, make_approx_phys, mpdf_approx) are used with multiprocessing; this incurs a minor performance penalty, and can be disabled by setting to False if the code will not be run with the multiprocessing module

caching ['aggressive' | 'lazy' | 'none'] strategy for caching subsets of the data with some dimensions marginalised out; behavior is as follows:

'agressive' on construction, store sorted data for fast calculation of 1D PDFs of variables by themselves, and 1D PDFs of all physical variables marginalised over all other physical variables; this significantly increases the memory footprint and construction time, but greatly speeds up subsequent evaluation of any of these quantities, and is generally the best choice for prudction work in parallel

'lazy' sorted data sets for fast computation are created and cached as needed; this will make the first computation of any marginal PDF slower, but speed up all subsequent ones, without imposing any extra time at initial construction; this mode is generally best for interactive work, but is not thread_safe = True; memory cost depends on how many different marginal PDF combinations are calculated, but is always less than aggressive

'none' no caching is performed automatically; the user may still manually cache data by calling the make_cache method

Returns Nothing

Raises IOError, if the bayesphot c library cannot be found

Notes Because the data sets passed in may be large, this class does not make copies of any of its arguments, and instead modifies them in place. However, this means it is the responsibility of the user not to alter the any of the arguments once they are passed to this class; for example, dataset must not be modified after it is passed. Altering the arguments, except through this class's methods, may cause incorrect results to be generated.

Caching with 'aggressive' or 'lazy' does create significant memory overhead.

__weakref__
list of weak references to the object (if defined)

bandwidth
The current bandwidth

bestmatch(*phot*, *photerr=None*, *nmatch=1*, *bandwidth_units=False*)
Searches through the simulation library and returns the closest matches to an input set of photometry.

Parameters:

phot [arraylike, shape (nfilter) or (..., nfilter)] array giving the photometric values; for a multidimensional array, the operation is vectorized over the leading dimensions

photerr [arraylike, shape (nfilter) or (..., nfilter)] array giving photometric errors, which must have the same shape as phot; if this is not None, then distances will be measured in units of the photometric error if bandwidth_units is False, or in units of the bandwidth added in quadrature with the errors if it is True

nmatch [int] number of matches to return; returned matches will be ordered by distance from the input

bandwidth_units [bool] if False, distances are computed based on the logarithmic difference in luminosity; if True, they are measured in units of the bandwidth

Returns:

matches [array, shape (..., nmatch, nphys + nfilter)] best matches to the input photometry; shape in the leading dimensions will be the same as for phot, and if nmatch == 1 then that dimension will be omitted

dist [array, shape (..., nmatch)] distances between the matches and the input photometry

bestmatch_phys (*phys*, *nmatch=1*, *bandwidth_units=False*)

Searches through the simulation library and returns the closest matches to an input set of photometry.

Parameters:

phot [arraylike, shape (nphys) or (..., nphys)] array giving the physical values; for a multidimensional array, the operation is vectorized over the leading dimensions

nmatch [int] number of matches to return; returned matches will be ordered by distance from the input

bandwidth_units [bool] if False, distances are computed based on the logarithmic difference in physical properties; if True, they are measured in units of the bandwidth

Returns:

matches [array, shape (..., nmatch, nphys + nfilter)] best matches to the input properties; shape in the leading dimensions will be the same as for phot, and if nmatch == 1 then that dimension will be omitted

dist [array, shape (..., nmatch)] distances between the matches and the input physical properties

clear_cache (*margindims=None*)

This method deletes from the cache

Parameters

margindims [listlike of integers] list of marginalised dimensions that should be removed from the cache, in the same format as make_cache; if left as None, the cache is completely emptied

Returns Nothing

draw_phot (*physprop*, *physidx=None*, *photerr=None*, *nsample=1*)

Returns a randomly-drawn sample of photometric properties for one or more input sets of physical properties.

Parameters:

physprop [arraylike] physical properties to be used; the final dimension of the input must have len(physidx) indices, or nphys indicates if physidx is None; if the input is a multidimensional array, the operation is vectorized over the leading dimensions physical properties

physidx [arraylike] indices of the physical quantities being constrained; if left as None, all physical properties are set, and physprop must have a trailing dimension of size equal to nphys; otherwise

this must be an arraylike of <= nphys positive integers, each unique and in the range [0, nphys), specying which physical dimensions are constrained

photerr [arraylike, shape (nphot)] photometric errors to apply to the output photometry; these are added in quadrature with the kernel density estimation bandwidth

nsample [int] number of random samples to draw for each set of physical properties; must be positive

Returns:

samples [arraylike] sample of photometric properties; the shape of the output is (..., nsample, nphot), where nphot is the number of photometric properties, and the leading dimension(s) match the leading dimension(s) of physprop

draw_sample(*photerr=None*, *nsample=1*)

Returns a randomly-drawn sample of physical and photometric properties from the kernel density function

Parameters:

nsample [int] number of random samples to draw for each set of physical properties; must be positive

photerr [arraylike, shape (nphot)] photometric errors to apply to the output photometry; these are added in quadrature with the kernel density estimation bandwidth

Returns:

samples [array, shape (nsample, nphys+nphot)] random sample drawn from the kernel density object; for the final dimension in the output, the first nphys elements are the physical quantities, the next nphot are the photometric quantities

logL(*physprop*, *photprop*, *photerr=None*)

This function returns the natural log of the likelihood function evaluated at a particular log mass, log age, extinction, and set of log luminosities

Parameters

physprop [arraylike, shape (nphys) or (..., nphys)] array giving values of the physical properties; for a multidimensional array, the operation is vectorized over the leading dimensions

photprop [arraylike, shape (nfilter) or (..., nfilter)] array giving the photometric values; for a multidimensional array, the operation is vectorized over the leading dimensions

photerr [arraylike, shape (nfilter) or (..., nfilter)] array giving photometric errors; for a multidimensional array, the operation is vectorized over the leading dimensions

Returns

logL [float or arraylike] natural log of the likelihood function

make_approx_phot(*phys*, *squeeze=True*, *filter_ignore=None*)

Returns an object that can be used for a fast approximation of the PDF of photometric properties that corresponds to a set of physical properties. The PDF produced by summing over the points returned is guaranteed to account for at least 1-reltol of the marginal photometric probability, and to represent the shape of the PDF in photometric space within a local accuracy of reltol as well.

Parameters:

phys [arraylike, shape (nphys) or (N, nphys)] the set or sets of physical properties for which the approximation is to be generated

squeeze [bool] if True, the representation returned will be squeezed to minimize the number of points included, using reltol as the error tolerance

filter_ignore [None or listlike of bool] if None, the kernel density representation returned covers all filters; otherwise this must be a listlike of bool, one entry per filter, with a value of False

indicating that filter should be excluded from the values returned; suppressing filters can allow for more efficient representations

Returns:

x [array, shape (M, nphot), or a list of such arrays] an array containing the list of points to be used for the approximation, where nphot is the number of photometric filters being returned

wgts [array, shape (M), or a list of such arrays] an array containing the weights of the points

Notes: if the requested relative tolerance cannot be reached for numerical reasons (usually because the input point is too far from the library to allow accurate computation), x and wgts will be return as None, and a warning will be issued

make_approx_phys (*phot*, *photerr=None*, *squeeze=True*, *phys_ignore=None*, *tol=None*)

Returns an object that can be used for a fast approximation of the PDF of physical properties that corresponds to a set of photometric properties. The PDF produced by summing over the points returned is guaranteed to account for at least 1-reltol of the marginal photometric probability, and to represent the shape of the PDF in photometric space within a local accuracy of reltol as well.

Parameters:

phot [arraylike, shape (nfilter) or (N, nfilter)] the set or sets of photometric properties for which the approximation is to be generated

photerr [arraylike, shape (nfilter) or (N, nfilter)] array giving photometric errors; the number of elements in the output lists will be the size that results from broadcasting together the leading dimensions of phot and photerr

squeeze [bool] if True, the representation returned will be squeezed to minimize the number of points included, using reltol as the error tolerance

phys_ignore [None or listlike of bool] if None, the kernel density representation returned covers all physical properties; otherwise this must be a listlike of bool, one entry per physical dimension, with a value of False indicating that dimension should be excluded from the values returned; suppressing dimensions can allow for more efficient representations

tol [float] if set, this tolerance overrides the value of reltol

Returns:

x [array, shape (M, nphys), or a list of such arrays] an array containing the list of points to be used for the approximation, where nphys is the number of physical dimensions being returned

wgts [array, shape (M), or a list of such arrays] an array containing the weights of the points

Notes: if the requested relative tolerance cannot be reached for numerical reasons (usually because the input point is too far from the library to allow accurate computation), x and wgts will be return as None, and a warning will be issued

make_cache (*margindims*)

This method builds a cache to do faster calculation of PDFs where certain dimensions are marginalised out. If such caches exist, they are used automatically by all the computation methods.

Parameters

margindims [listlike of integers] list of dimensions to be marginalised out; physical dimensions go from 0 - nphys-1, photometric dimensions from nphys to nphys + nphot - 1

Returns Nothing

mcmc (*photprop*, *photerr=None*, *mc_walkers=100*, *mc_steps=500*, *mc_burn_in=50*)

This function returns a sample of MCMC walkers sampling the physical parameters at a specified set of photometric values.

Parameters:

photprop [arraylike, shape (nfilter) or (..., nfilter)] array giving the photometric values; for a multi-dimensional array, the operation is vectorized over the leading dimensions

photerr [arraylike, shape (nfilter) or (..., nfilter)] array giving photometric errors; for a multidimensional array, the operation is vectorized over the leading dimensions

mc_walkers [int] number of walkers to use in the MCMC

mc_steps [int] number of steps in the MCMC

mc_burn_in [int] number of steps to consider "burn-in" and discard

Returns

samples [array] array of sample points returned by the MCMC

mpdf(*idx*, *photprop*, *photerr=None*, *ngrid=128*, *qmin=None*, *qmax=None*, *grid=None*, *norm=True*)

Returns the marginal probability for one or mode physical quantities for one or more input sets of photometric properties. Output quantities are computed on a grid of values, in the same style as meshgrid.

Parameters:

idx [int or listlike containing ints] index of the physical quantity whose PDF is to be computed; if this is an iterable, the joint distribution of the indicated quantities is returned

photprop [arraylike, shape (nfilter) or (..., nfilter)] array giving the photometric values; for a multi-dimensional array, the operation is vectorized over the leading dimensions

photerr [arraylike, shape (nfilter) or (..., nfilter)] array giving photometric errors; for a multidimensional array, the operation is vectorized over the leading dimensions

ngrid [int or listlike containing ints] number of points in each dimension of the output grid; if this is an iterable, it must have the same number of elements as idx

qmin [float or listlike] minimum value in the output grid in each quantity; if left as None, defaults to the minimum value in the library; if this is an iterable, it must contain the same number of elements as idx

qmax [float or listlike] maximum value in the output grid in each quantity; if left as None, defaults to the maximum value in the library; if this is an iterable, it must contain the same number of elements as idx

grid [listlike of arrays] set of values defining the grid on which the PDF is to be evaluated, in the same format used by meshgrid

norm [bool] if True, returned pdf's will be normalized to integrate to 1

Returns:

grid_out [array] array of values at which the PDF is evaluated; contents are the same as returned by meshgrid

pdf [array] array of marginal posterior probabilities at each point of the output grid, for each input set of photometry; the leading dimensions match the leading dimensions produced by broadcasting the leading dimensions of photprop and photerr together, while the trailing dimensions match the dimensions of the output grid

mpdf_approx(*x*, *wgts*, *dims='phys'*, *dims_return=None*, *ngrid=64*, *qmin='all'*, *qmax='all'*, *grid=None*, *norm=True*)

Returns the marginal posterior PDF computed from a kernel density approximation returned by make_approx_phys or make_approx_phot. Outputs are computed on a grid of values, in the same style as meshgrid.

Parameters:

x [array, shape (M, ndim), or a list of such arrays] array of points retured by make_approx_phot or make_approx_phys

wgts [array, shape (M) or a list of such arrays] array of weights returned by make_approx_phot or make_approx_phys

dims ['phys' | 'phot' | arraylike of ints] dimensions covered by x and wgts; the strings 'phys' or 'phot' indicate that they cover all physical or photometric dimensions, and correspond to the defaults returned by make_approx_phys and make_approx_phot, respectively; if dims is an array of ints, these specify the dimensions covered by x and wgts, where the physical dimensions are numbered 0, 1, ... nphys-1, and the photometric ones are nphys, nphys+1, ... nphys+nphot-1

dims_return [None or arraylike of ints] if None, the output PDF has the same dimensions as specified in dms; if not, then dimreturn must be a subset of dim, and a marginal PDF in certain dimensions will be generated

ngrid [int or listlike containing ints] number of points in each dimension of the output grid; if this is an iterable, it must have the same number of elements as idx

qmin [float | listlike | 'zoom' | 'all'] minimum value in the output grid in each quantity; if this a float, it is applied to each dimension; if it is an iterable, it must contain the same number of elements as the number of dimensions being returned, as gives the minimum in each dimension; if it is 'zoom' or 'all', the minimum is chosen automatically, with 'zoom' focusing on a region encompassing the probability maximum, and 'all' encompassing all the points in the representation

qmax [float | listlike | 'zoom' | 'all'] same as qmin, but for the maximum of the output grid

grid [listlike of arrays] set of values defining the grid on which the PDF is to be evaluated, in the same format used by meshgrid

norm [bool] if True, returned pdf's will be normalized to integrate to 1

Returns:

grid_out [array] array of values at which the PDF is evaluated; contents are the same as returned by meshgrid

pdf [array] array of marginal posterior probabilities at each point of the output grid, for each input cluster; the leading dimensions match the leading dimensions produced by broadcasting the leading dimensions of photprop and photerr together, while the trailing dimensions match the dimensions of the output grid

mpdf_gen(*fixeddim*, *fixedprop*, *margindim*, *ngrid=128*, *qmin=None*, *qmax=None*, *grid=None*, *norm=True*)

Returns the marginal probability for one or more physical or photometric properties, keeping other properties fixed and marginalizing over other quantities. This is the most general marginal PDF routine provided.

Parameters:

fixeddim [int | arraylike of ints | None] The index or indices of the physical or photometric properties to be held fixed; physical properties are numbered 0 ... nphys-1, and phtometric ones are numbered nphys ... nphys + nphot - 1. This can also be set to None, in which case no properties are held fixed.

fixedprop [array | None] The values of the properties being held fixed; the size of the final dimension must be equal to the number of elements in fixeddim, and if fixeddim is None, this must be too

margindim [int | arraylike of ints | None] The index or indices of the physical or photometric properties to be maginalized over, numbered in the same way as with fixeddim; if set to None, no marginalization is performed

ngrid [int or listlike containing ints] number of points in each dimension of the output grid; if this is an iterable, it must have nphys + nphot - len(fixeddim) - len(margindim) elements

qmin [float | arraylike] minimum value in the output grid in each quantity; if left as None, defaults to the minimum value in the library; if this is an iterable, it must contain a number of elements equal to nphys + nphot - len(fixeddim) - len(margindim)

qmax [float | arraylike] maximum value in the output grid in each quantity; if left as None, defaults to the maximum value in the library; if this is an iterable, it must have the same number of elements as qmin

grid [listlike of arrays] set of values defining the grid on which the PDF is to be evaluated, in the same format used by meshgrid

norm [bool] if True, returned pdf's will be normalized to integrate to 1

Returns:

grid_out [array] array of values at which the PDF is evaluated; contents are the same as returned by meshgrid

pdf [array] array of marginal posterior probabilities at each point of the output grid, for each input set of properties; the leading dimensions match the leading dimensions produced by broadcasting the leading dimensions of photprop and photerr together, while the trailing dimensions match the dimensions of the output grid

mpdf_phot(*idx*, *physprop*, *ngrid=128*, *qmin=None*, *qmax=None*, *grid=None*, *norm=True*)

Returns the marginal probability for one or mode photometric quantities corresponding to an input set of physical properties. Output quantities are computed on a grid of values, in the same style as meshgrid.

Parameters:

idx [int or listlike containing ints] index of the photometric quantity whose PDF is to be computed, starting at 0; if this is an iterable, the joint distribution of the indicated quantities is returned

physprop [arraylike, shape (nphys) or (..., nphys)] physical properties to be used; if this is an array of nphys elements, these give the physical properties; if it is a multidimensional array, the operation is vectorized over the leading dimensions physical properties – the function must take an array of (nphys) elements as an input, and return a floating point value representing the PDF evaluated at that set of physical properties as an output

ngrid [int or listlike containing ints] number of points in each dimension of the output grid; if this is an iterable, it must have the same number of elements as idx

qmin [float or listlike] minimum value in the output grid in each quantity; if left as None, defaults to the minimum value in the library; if this is an iterable, it must contain the same number of elements as idx

qmax [float or listlike] maximum value in the output grid in each quantity; if left as None, defaults to the maximum value in the library; if this is an iterable, it must contain the same number of elements as idx

grid [listlike of arrays] set of values defining the grid on which the PDF is to be evaluated, in the same format used by meshgrid

norm [bool] if True, returned pdf's will be normalized to integrate to 1

Returns:

grid_out [array] array of values at which the PDF is evaluated; contents are the same as returned by meshgrid

pdf [array] array of marginal posterior probabilities at each point of the output grid, for each input set of properties; the leading dimensions match the leading dimensions produced by broadcasting the leading dimensions of photprop and photerr together, while the trailing dimensions match the dimensions of the output grid

ndim
This is the number of physical plus photometric properties for the bayesphot object.

nphot
This is the number of photometric properties for the bayesphot object.

nphys
This is the number of physical properties for the bayesphot object.

pobs
The current set of observation probabilities for every simulation in the library; data returned are in the same order as the data originally used to construct the library

priors
The current set of prior probabilities for every simulation in the library; data returned are in the same order as the data originally used to construct the library

sample_density
The density with which the library was sampled, evaluated for each simulation in the library

squeeze_rep(*x*, *wgts*, *dims=None*)
Takes an input array of positions and weights that form a kernel density representation and approximates them using fewer points, using an error tolerance of reltol

Parameters:

x [array, shape (N, ndim)] an array of points forming a kernel density representation; on exit, x will be resized to (M, ndim) with M <= N

wgts [array, shape (N)] an array of weights for the kernel density representation; on exit, wgts will be resized to (M), with M <= N

dims [array, shape (ndim)] array specifying which dimensions in the kernel density representation the coordinates in x correspond to; if left as None, they are assumed to correspond to the first ndim dimensions in the data set

Returns: Nothing

CHAPTER

THIRTEEN

CLUSTER_SLUG: BAYESIAN INFERENCE OF STAR CLUSTER PROPERTIES

The slugpy.cluster_slug module computes posterior probabilities for the mass, age, and extinction of star clusters from a set of input photometry. It is implemented as a wrapper around *bayesphot: Bayesian Inference for Stochastic Stellar Populations*, so for details on how the calculation is performed see the bayesphot documentation.

13.1 Getting the Default Library

The cluster_slug module requires a pre-computed library of slug simulations to use as a "training set" for its calculations. Due to its size, the default library *is not* included in the slug git repository. Instead, it is provided for download from the SLUG data products website. Download the two files `clusterslug_mw_cluster_phot.fits` and `clusterslug_mw_cluster_prop.fits` and save them in the `cluster_slug` directory of the main respository. If you do not do so, and do not provide your own library when you attempt to use cluster_slug, you will be prompted to download the default library.

13.2 Basic Usage

For an example of how to use cluster_slug, see the file `cluster_slug/cluster_slug_example.py` in the repository. All funtionality is provided through the cluster_slug class. The basic steps are as follows:

1. Import the library and instantiate an `sfr_slug` object (see *Full Documentation of slugpy.cluster_slug* for full details):

```
from slugpy.cluster_slug import cluster_slug
cs = cluster_slug(photsystem=photsystem)
```

This creates a cluster_slug object, using the default simulation library. If you have another library of simulations you'd rather use, you can use the `libname` keyword to the `cluster_slug` constructor to select it. The optional argument `photsystem` specifies the photometric system you will be using for your data. Possible values are `L_nu` (flux per unit frequency, in erg/s/Hz), `L_lambda` (flux per unit wavelength, in erg/s/Angstrom), `AB` (AB magnitudes), `STMAG` (ST magnitudes), and `Vega` (Vega magnitudes). If left unspecified, the photometric system will be whatever the library was written in; the default library is in the `L_nu` system. Finally, if you have already read a library into memory using `read_cluster`, you can set the keyword `lib` in the `cluster_slug` constructor to specify that library should be used.

2. Specify your filter(s), for example:

```
cs.add_filters(['WFC3_UVIS_F336W', 'WFC3_UVIS_F438W', 'WFC3_UVIS_F555W',
                'WFC3_UVIS_F814W', 'WFC3_UVIS_F657N'])
```

The `add_filter` method takes as an argument a string or list of strings specifying which filters were used for the observations you're going to analyze. You can have more than one set of filters active at a time (just by calling `add_filters` more than once), and then specify which set of filters you're using for any given calculation.

3. Specify your priors, for example:

```
# Set priors to be flat in log T and A_V, but vary with log M as
# p(log M) ~ 1/M
def priorfunc(physprop):
   # Note: physprop is an array of shape (N, 3) where physprop[:,0] =
   # log M, physprop[:,1] = log T, physprop[:,2] = A_V
   return 1.0/exp(physprop[:,0])
cs.priors = prorfunc
```

The `priors` property specifies the assumed prior probability distribution on the physical properties of star clusters. It can be either `None` (in which case all simulations in the library are given equal prior probability), an array with as many elements as there are simulations in the library giving the prior for each one, or a callable that takes a vector of physical properties as input and returns the prior for it.

4. Generate a marginal posterior probability distribuiton via:

```
logm, pdf = cs.mpdf(idx, phot, photerr = photerr)
```

The first argument `idx` is an index for which posterior distribution should be computed – a value of 0 generates the posterior in log mass, a value of 1 generates the posterion on log age, and a value of generates the posterior in A_V. The second argument `phot` is an array giving the photometric values in the filters specified in step 2; make sure you're using the same photometric system you used in step 1. For the array `phot`, the trailing dimension must match the number of filters, and the marginal posterior-finding exercise is repeated over every value in the leading dimensions. If you have added two or more filter sets, you need to specify which one you want to use via the `filters` keyword. The optional argument `photerr` can be used to provide errors on the photometric values. The shape rules on it are the same as on `phot`, and the two leading dimensions of the two arrays will be broadcast together using normal broadcasting rules.

The `cluster_slug.mpdf` method returns a tuple of two quantities. The first is a grid of values for log M, log T, or A_V, depending on the value of `idx`. The second is the posterior probability distribution at each value of of the grid. Posteriors are normalized to have unit integral. If the input consisted of multiple sets of photometric values, the output will contains marginal posterior probabilities for each input. The output grid will be created automatically be default, but all aspects of it (shape, size, placement of grid points) can be controlled by keywords – see *Full Documentation of slugpy.cluster_slug*.

13.3 Using cluster_slug in Parallel

The `cluster_slug` module has full support for threaded computation using the python multiprocessing module. This allows efficient use of multiple cores on a shared memory machine, without the need for every project to read a large simulation library or store it in memory. See *Parallelism in bayesphot* for full details on the recommended paradigm for parallel computing. The full list of thread-safe `cluster_slug` methods is:

- `cluster_slug.logL`
- `cluster_slug.mpdf`
- `cluster_slug.mcmc`
- `cluster_slug.bestmatch`
- `cluster_slug.make_approx_phot`
- `cluster_slug.make_approx_phys`

- `cluster_slug.squeeze_rep`
- `cluster_slug.mpdf_approx`

13.4 Making Your Own Library

You can generate your own library by running slug; you might want to do this, for example, to have a library that works at different metallicity or for a different set of stellar tracks. An example parameter file (the one that was used to generate the default clusterslug_mw library) is included in the `cluster_slug` directory. This file uses slug's capability to pick the output time and the cluster mass from specified PDFs.

One subtle post-processing step you should take once you've generated your library is to read it in using *slugpy – The Python Helper Library* and then write the photometry back out using the `slugpy.write_cluster_phot` routine with the format set to `fits2`. This uses an alternative FITS format that is faster to search when you want to load only a few filters out of a large library. For large data sets, this can reduce cluster_slug load times by an order of magnitude. (To be precise: the default format for FITS outputs to put all filters into a single binary table HDU, while the `fits2` format puts each filter in its own HDU. This puts all data for a single filter into a contiguous block, rather than all the data for a single cluster into a contiguous block, and is therefore faster to load when one wants to load the data filter by filter.)

13.5 Variable Mode IMF

If your library was run with variable IMF parameters, these can also be used in `cluster_slug`. When creating a `cluster_slug` object, you can pass the array `vp_list` as an argument. This list should have an element for each variable parameter in your library. Each element should then be either `True` or `False` depending on whether you wish to include this parameter in the analysis. For example, for a library with four variable parameters you could have:

```
vp_list=[True,False,True,True]
```

13.6 Full Documentation of slugpy.cluster_slug

class slugpy.cluster_slug.**cluster_slug**(*libname=None*, *filters=None*, *photsystem=None*, *lib=None*, *bw_phys=0.1*, *bw_phot=None*, *ktype='gaussian'*, *priors=None*, *sample_density=None*, *pobs=None*, *reltol=0.01*, *abstol=1e-08*, *leafsize=16*, *use_nebular=True*, *use_extinction=True*, *thread_safe=True*, *caching='none'*, *vp_list=[]*)

A class that can be used to estimate the PDF of star cluster properties (mass, age, extinction) from a set of input photometry in various bands.

Properties

priors [array, shape (N) | callable | None] prior probability on each data point; interpretation depends on the type passed; array, shape (N): values are interpreted as the prior probability of each data point; callable: the callable must take as an argument an array of shape (N, nphys), and return an array of shape (N) giving the prior probability at each data point; None: all data points have equal prior probability

abstol [float] absolute error tolerance for kernel density estimation

reltol [float] relative error tolerance for kernel density estimation

thread_safe [bool] if True, the computation routines will run in thread-safe mode, allowing use with multiprocessing; this incurs a small performance penalty

Methods

filters() [] returns list of filters available in the library

filtersets() : return a list of the currently-loaded filter sets

filter_units() : returns units for available filters

add_filters() [] adds a set of filters for use in parameter estimation

logL() [] compute log likelihood at a particular set of physical and photometric parameters

mpdf() [] computer marginal posterior probability distribution for a set of photometric measurements

mcmc() : due MCMC estimation of the posterior PDF on a set of photometric measurments

bestmatch() [] find the simulations in the library that are the closest matches to the input photometry

make_approx_phot() : given a set of physical properties, return a set of points that can be used for fast approximation of the corresponding photometric properties

make_approx_phys() : given a set of photometric properties, return a set of points that can be used for fast approximation of the corresponding physical properties

draw_sample(): draw a random sample of cluster physical and photometric properties

draw_phot(): given a set of physical properties, return a randomly-selected set of photometric properties

__init__(*libname=None*, *filters=None*, *photsystem=None*, *lib=None*, *bw_phys=0.1*, *bw_phot=None*, *ktype='gaussian'*, *priors=None*, *sample_density=None*, *pobs=None*, *reltol=0.01*, *abstol=1e-08*, *leafsize=16*, *use_nebular=True*, *use_extinction=True*, *thread_safe=True*, *caching='none'*, *vp_list=[]*)

Initialize a cluster_slug object.

Parameters

libname [string] name of the SLUG model to load; if left as None, the default is $SLUG_DIR/cluster_slug/modp020_chabrier_MW

lib [object] a library read by the read_cluster function; if specified this overrides the libname option; the library must contain both physical properties and photometry, and must include filter data; if this is not None, then the photsystem keyword is ignored

filters [iterable of stringlike] list of filter names to be used for inferenence

photsystem [None or string] If photsystem is None, the library will be left in whatever photometric system was used to write it. Alternately, if it is a string, the data will be converted to the specified photometric system. Allowable values are 'L_nu', 'L_lambda', 'AB', 'STMAG', and 'Vega', corresponding to the options defined in the SLUG code. Once this is set, any subsequent photometric data input are assumed to be in the same photometric system.

bw_phys ['auto' | float | array, shape (2) | array, shape (3)] bandwidth for the physical quantities in the kernel density estimation; if set to 'auto', the bandwidth will be estimated automatically; if set to a scalar quantity, this will be used for all physical quantities; if set to an array, the array must have 2 elements if use_extinction is False, or 3 if it is True

bw_phot [None | 'auto' | float | array] bandwidth for the photometric quantities; if set to None, defaults to 0.25 mag / 0.1 dex; if set to 'auto', bandwidth is estimated automatically; if set to a float, this bandwidth is used for all photometric dimensions; if set to an array, the array must have the same number of dimensions as len(filters)

ktype [string] type of kernel to be used in densty estimation; allowed values are 'gaussian' (default), 'epanechnikov', and 'tophat'; only Gaussian can be used with error bars

priors [array, shape (N) | callable | None] prior probability on each data point; interpretation depends on the type passed; array, shape (N): values are interpreted as the prior probability of each data point; callable: the callable must take as an argument an array of shape (N, nphys), and return an array of shape (N) giving the prior probability at each data point; None: all data points have equal prior probability

pobs [array, shape (N) | callable | None] probability of being observed on each data point; interpretation depends on the type passed; array, shape (N): values are interpreted as the prior probability of each data point; callable: the callable must take as an argument an array of shape (N, nphys), and return an array of shape (N) giving the prior probability at each data point; None: all data points have equal prior probability

sample_density [array, shape (N) | callable | 'auto' | None] the density of the data samples at each data point; this need not match the prior density; interpretation depends on the type passed; array, shape (N): values are interpreted as the density of data sampling at each sample point; callable: the callable must take as an argument an array of shape (N, nphys), and return an array of shape (N) giving the sampling density at each point; 'auto': the sample density will be computed directly from the data set; note that this can be quite slow for large data sets, so it is preferable to specify this analytically if it is known; None: data are assumed to be uniformly sampled, or to be sampled as the default library is if libname is also None

reltol [float] relative error tolerance; errors on all returned probabilities p will satisfy either abs(p_est - p_true) <= reltol * p_est OR abs(p_est - p_true) <= abstol, where p_est is the returned estimate and p_true is the true value

abstol [float] absolute error tolerance; see above

leafsize [int] number of data points in each leaf of the KD tree

use_nebular [bool] if True, photometry including nebular emission will be used if available; if not, nebular emission will be omitted

use_extinction [bool] if True, photometry including extinction will be used; if not, it will be omitted, and in this case no results making use of the A_V dimension will be available

thread_safe [bool] if True, cluster_slug will make extra copies of internals as needed to ensure thread safety when the computation routines (logL, mpdf, mcmc, bestmatch, make_approx_phot, make_approx_phys, mpdf_approx) are used with multiprocessing; this incurs a minor performance penalty, and can be disabled by setting to False if the code will not be run with the multiprocessing module

caching ['aggressive' | 'lazy' | 'none'] strategy for caching subsets of the data with some dimensions marginalised out; behavior is as follows:

'agressive' on construction, store sorted data for fast calculation of 1D PDFs of variables by themselves, and 1D PDFs of all physical variables marginalised over all other physical variables; this significantly increases the memory footprint and construction time, but greatly speeds up subsequent evaluation of any of these quantities, and is generally the best choice for prudction work in parallel

'lazy' sorted data sets for fast computation are created and cached as needed; this will make the first computation of any marginal PDF slower, but speed up all subsequent ones, without imposing any extra time at initial construction; this mode is generally best for interactive work, but is not thread_safe = True; memory cost depends on how many different marginal PDF combinations are calculated, but is always less than aggressive

'none' no caching is performed automatically; the user may still manually cache data by calling the make_cache method

vp_list [list] A list with an element for each of the variable parameters in the data. An element is set to True if we wish to use that parameter here, or False if we do not.

Returns Nothing

Raises IOError, if the library cannot be found

__weakref__
list of weak references to the object (if defined)

add_filters(*filters*, *bandwidth=None*, *pobs=None*)
Add a set of filters to use for cluster property estimation

Parameters

filters [iterable of stringlike] list of filter names to be used for inferenence

bandwidth [None | 'auto' | float | array] bandwidth for the photometric quantities; if set to None, the bandwidth is unchanged for an existing filter set, and for a newly-created one the default physical and photometric bandwidths are used; if set to 'auto', bandwidth is estimated automatically; if set to a float, this bandwidth is used for all physical and photometric dimensions; if set to an array, the array must have the same number of entries as nphys+len(filters)

pobs [array, shape (N) | callable | 'equal' | None] the probability that a particular object would be observed, which is used, like prior, to weight the library; interpretation depends on type. 'equal' means all objects are equally likely to be observed, array is an array giving the observation probability of each object in the library, and callable means must be a function that takes an array containing the photometry, of shape (N, nhpot), as an argument, and returns an array of shape (N) giving the probability of observation for that object. Finally, None leaves the observational probability unchanged

Returns nothing

bestmatch(*phot*, *photerr=None*, *nmatch=1*, *bandwidth_units=False*, *filters=None*)
Searches through the simulation library and returns the closest matches to an input set of photometry.

Parameters:

phot [arraylike, shape (nfilter) or (..., nfilter)] array giving the photometric values; for a multidimensional array, the operation is vectorized over the leading dimensions

photerr [arraylike, shape (nfilter) or (..., nfilter)] array giving photometric errors, which must have the same shape as phot; if this is not None, then distances will be measured in units of the photometric error if bandwidth_units is False, or in units of the bandwidth added in quadrature with the errors if it is True

nmatch [int] number of matches to return; returned matches will be ordered by distance from the input

bandwidth_units [bool] if False, distances are computed based on the logarithmic difference in luminosity; if True, they are measured in units of the bandwidth

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns:

matches [array, shape (..., nmatch, nphys + nfilter)] best matches to the input photometry; shape in the leading dimensions will be the same as for phot, and if nmatch == 1 then that dimension will be omitted; in the final dimension, the first 3 elements give log M, log T, and A_V, while the

last nfilter give the photometric values; if created with use_extinct = False, the A_V dimension is omitted

dist [array, shape (..., nmatch)] distances between the matches and the input photometry

bestmatch_phys (*phys*, *nmatch=1*, *bandwidth_units=False*, *filters=None*)
Searches through the simulation library and returns the closest matches to an input set of photometry.

Parameters:

phys [arraylike, shape (nphys) or (..., nphys)] array giving the physical values; for a multidimensional array, the operation is vectorized over the leading dimensions

nmatch [int] number of matches to return; returned matches will be ordered by distance from the input

bandwidth_units [bool] if False, distances are computed based on the logarithmic difference in physical properties; if True, they are measured in units of the bandwidth

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns:

matches [array, shape (..., nmatch, nphys + nfilter)] best matches to the input properties; shape in the leading dimensions will be the same as for phot, and if nmatch == 1 then that dimension will be omitted

dist [array, shape (..., nmatch)] distances between the matches and the input physical properties

clear_cache (*margindims=None*)
This method deletes from the cache

Parameters

margindims [listlike of integers] list of marginalised dimensions that should be removed from the cache, in the same format as make_cache; if left as None, the cache is completely emptied

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns Nothing

del_filters (*filters*)
Remove a set of filters, freeing the memory associated with them. Note that this does not delete the underlying library data, just the data for the KD tree used internally.

Parameters

filters [iterable of stringlike] list of filter names

Returns Nothing

Raises KeyError if the input set of filters is not loaded

draw_phot (*physprop*, *physidx=None*, *photerr=None*, *nsample=1*, *filters=None*)
Returns a randomly-drawn sample of clusters for a given filter set

Parameters:

physprop [arraylike] physical properties to be used; the final dimension of the input must have len(physidx) indices, or nphys indicates if physidx is None; if the input is a multidimensional array, the operation is vectorized over the leading dimensions physical properties

physidx [arraylike] indices of the physical quantities being constrained; if left as None, all physical properties are set, and physprop must have a trailing dimension of size equal to nphys; otherwise this must be an arraylike of <= nphys positive integers, each unique and in the range [0, nphys), specying which physical dimensions are constrained

photerr [arraylike, shape (nphot)] photometric errors to apply to the output photometry; these are added in quadrature with the kernel density estimation bandwidth

nsample [int] number of random samples to draw for each set of physical properties; must be positive

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns:

samples [array, shape (nsample, nphys+nphot)] random sample drawn from the kernel density object; for the final dimension in the output, the first nphys elements are the physical quantities, the next nphot are the photometric quantities

draw_sample(*photerr=None*, *nsample=1*, *filters=None*)

Returns a randomly-drawn sample of clusters for a given filter set

Parameters:

photerr [arraylike, shape (nphot)] photometric errors to apply to the output photometry; these are added in quadrature with the kernel density estimation bandwidth

nsample [int] number of random samples to draw for each set of physical properties; must be positive

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns:

samples [array, shape (nsample, nphys+nphot)] random sample drawn from the kernel density object; for the final dimension in the output, the first nphys elements are the physical quantities, the next nphot are the photometric quantities

filter_units()

Returns list of all available filter units

Parameters: None

Returns:

units [list of strings] list of available filter units

filters()

Returns list of all available filters

Parameters: None

Returns:

filters [list of strings] list of available filter names

filtersets()

Returns list of all currently-loaded filter sets

Parameters: None

Returns:

filtersets [list of list of strings] list of currently-loaded filter sets

load_data(*filter_name*, *bandwidth=None*, *force_reload=False*)
Loads photometric data for the specified filter into memory

Parameters:

filter_name [string] name of filter to load

bandwidth [float] default bandwidth for this filter

force_reload [bool] if True, reinitialize the data even if has already been stored

Returns: None

Raises: ValueError, if filter_name is not one of the available filters

logL(*physprop*, *photprop*, *photerr=None*, *filters=None*)
This function returns the natural log of the likelihood function evaluated at a particular log mass, log age, extinction, and set of log luminosities

Parameters:

physprop [arraylike, shape (nhpys) or (..., nphys)] array giving values of the log M, log T, and A_V; for a multidimensional array, the operation is vectorized over the leading dimensions; if created with use_extinct = False, the A_V dimension should be omitted. Will also include variable any variable parameters VPx if they are requested.

photprop [arraylike, shape (nfilter) or (..., nfilter)] array giving the photometric values; for a multidimensional array, the operation is vectorized over the leading dimensions

photerr [arraylike, shape (nfilter) or (..., nfilter)] array giving photometric errors; for a multidimensional array, the operation is vectorized over the leading dimensions

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns:

logL [float or arraylike] natural log of the likelihood function

make_approx_phot(*phys*, *squeeze=True*, *filter_ignore=None*, *filters=None*)
Returns an object that can be used for a fast approximation of the PDF of photometric properties that corresponds to a set of physical properties. The PDF produced by summing over the points returned is guaranteed to account for at least 1-reltol of the marginal photometric probability, and to represent the shape of the PDF in photometric space within a local accuracy of reltol as well.

Parameters:

phys [arraylike, shape (nphys) or (N, nphys)] the set or sets of physical properties for which the approximation is to be generated

squeeze [bool] if True, the representation returned will be squeezed to minimize the number of points included, using reltol as the error tolerance

filter_ignore [None or listlike of bool] if None, the kernel density representation returned covers all filters; otherwise this must be a listlike of bool, one entry per filter, with a value of False indicating that filter should be excluded from the values returned; suppressing filters can allow for more efficient representations

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns:

x [array, shape (M, nphot), or a list of such arrays] an array containing the list of points to be used for the approximation

wgts [array, shape (M), or a list of such arrays] an array containing the weights of the points

Notes: if the requested relative tolerance cannot be reached for numerical reasons (usually because the input point is too far from the library to allow accurate computation), x and wgts will be return as None, and a warning will be issued

make_approx_phys(*phot*, *photerr=None*, *squeeze=True*, *phys_ignore=None*, *filters=None*, *tol=None*)

Returns an object that can be used for a fast approximation of the PDF of physical properties that corresponds to a set of photometric properties. The PDF produced by summing over the points returned is guaranteed to account for at least 1-reltol of the marginal photometric probability, and to represent the shape of the PDF in photometric space within a local accuracy of reltol as well.

Parameters:

phot [arraylike, shape (nfilter) or (N, nfilter)] the set or sets of photometric properties for which the approximation is to be generated

photerr [arraylike, shape (nfilter) or (N, nfilter)] array giving photometric errors; the number of elements in the output lists will be the size that results from broadcasting together the leading dimensions of phot and photerr

squeeze [bool] if True, the representation returned will be squeezed to minimize the number of points included, using reltol as the error tolerance

phys_ignore [None or listlike of bool] if None, the kernel density representation returned covers all physical properties; otherwise this must be a listlike of bool, one entry per physical dimension, with a value of False indicating that dimension should be excluded from the values returned; suppressing dimensions can allow for more efficient representations

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

tol [float] if set, this tolerance overrides the value of reltol

Returns:

x [array, shape (M, nphys), or a list of such arrays] an array containing the list of points to be used for the approximation, where nphys is the number of physical dimensions being returned

wgts [array, shape (M), or a list of such arrays] an array containing the weights of the points

Notes: if the requested relative tolerance cannot be reached for numerical reasons (usually because the input point is too far from the library to allow accurate computation), x and wgts will be return as None, and a warning will be issued

make_cache(*margindims*, *filters=None*)

This method builds a cache to do faster calculation of PDFs where certain dimensions are marginalised out. If such caches exist, they are used automatically by all the computation methods.

Parameters

margindims [listlike of integers] list of dimensions to be marginalised out; physical dimensions go from 0 - nphys-1, photometric dimensions from nphys to nphys + nphot - 1; note that the indexing depends on the filter set specified by filters

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns Nothing

mcmc(*photprop*, *photerr=None*, *mc_walkers=100*, *mc_steps=500*, *mc_burn_in=50*, *filters=None*)

This function returns a sample of MCMC walkers for cluster mass, age, and extinction

Parameters:

photprop [arraylike, shape (nfilter) or (..., nfilter)] array giving the photometric values; for a multidimensional array, the operation is vectorized over the leading dimensions

photerr [arraylike, shape (nfilter) or (..., nfilter)] array giving photometric errors; for a multidimensional array, the operation is vectorized over the leading dimensions

mc_walkers [int] number of walkers to use in the MCMC

mc_steps [int] number of steps in the MCMC

mc_burn_in [int] number of steps to consider "burn-in" and discard

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns

samples [array] array of sample points returned by the MCMC

mpdf(*idx*, *photprop*, *photerr=None*, *ngrid=128*, *qmin=None*, *qmax=None*, *grid=None*, *norm=True*, *filters=None*)

Returns the marginal probability for one or mode physical quantities for one or more input sets of photometric properties. Output quantities are computed on a grid of values, in the same style as meshgrid

Parameters:

idx [int or listlike containing ints] index of the physical quantity whose PDF is to be computed; 0 = log M, 1 = log T, 2 = A_V, (2 or 3)+x = VPx; if this is an iterable, the joint distribution of the indicated quantities is returned

photprop [arraylike, shape (nfilter) or (..., nfilter)] array giving the photometric values; for a multidimensional array, the operation is vectorized over the leading dimensions

photerr [arraylike, shape (nfilter) or (..., nfilter)] array giving photometric errors; for a multidimensional array, the operation is vectorized over the leading dimensions

ngrid [int or listlike containing ints] number of points in each dimension of the output grid; if this is an iterable, it must have the same number of elements as idx

qmin [float or listlike] minimum value in the output grid in each quantity; if left as None, defaults to the minimum value in the library; if this is an iterable, it must contain the same number of elements as idx

qmax [float or listlike] maximum value in the output grid in each quantity; if left as None, defaults to the maximum value in the library; if this is an iterable, it must contain the same number of elements as idx

grid [listlike of arrays] set of values defining the grid on which the PDF is to be evaluated, in the same format used by meshgrid

norm [bool] if True, returned pdf's will be normalized to integrate to 1

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns:

grid_out [array] array of values at which the PDF is evaluated; contents are the same as returned by meshgrid

pdf [array] array of marginal posterior probabilities at each point of the output grid, for each input cluster; the leading dimensions match the leading dimensions produced by broadcasting the

leading dimensions of photprop and photerr together, while the trailing dimensions match the dimensions of the output grid

mpdf_approx(*x*, *wgts*, *dims='phys'*, *dims_return=None*, *ngrid=64*, *qmin='all'*, *qmax='all'*, *grid=None*, *norm=True*, *filters=None*)

Returns the marginal posterior PDF computed from a kernel density approximation returned by make_approx_phys or make_approx_phot. Outputs are computed on a grid of values, in the same style as meshgrid.

Parameters:

x [array, shape (M, ndim), or a list of such arrays] array of points retured by make_approx_phot or make_approx_phys

wgts [array, shape (M) or a list of such arrays] array of weights returned by make_approx_phot or make_approx_phys

dims ['phys' | 'phot' | arraylike of ints] dimensions covered by x and wgts; the strings 'phys' or 'phot' indicate that they cover all physical or photometric dimensions, and correspond to the defaults returned by make_approx_phys and make_approx_phot, respectively; if dims is an array of ints, these specify the dimensions covered by x and wgts, where the physical dimensions are numbered 0, 1, ... nphys-1, and the photometric ones are nphys, nphys+1, ... nphys+nphot-1

dims_return [None or arraylike of ints] if None, the output PDF has the same dimensions as specified in dims; if not, then dims_return must be a subset of dims, and a marginal PDF in certain dimensions will be generated

ngrid [int or listlike containing ints] number of points in each dimension of the output grid; if this is an iterable, it must have the same number of elements as idx

qmin [float | listlike | 'zoom' | 'all'] minimum value in the output grid in each quantity; if this a float, it is applied to each dimension; if it is an iterable, it must contain the same number of elements as the number of dimensions being returned, as gives the minimum in each dimension; if it is 'zoom' or 'all', the minimum is chosen automatically, with 'zoom' focusing on a region encompassing the probability maximum, and 'all' encompassing all the points in the representation

qmax [float | listlike | 'zoom' | 'all'] same as qmin, but for the maximum of the output grid

grid [listlike of arrays] set of values defining the grid on which the PDF is to be evaluated, in the same format used by meshgrid

norm [bool] if True, returned pdf's will be normalized to integrate to 1

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns:

grid_out [array] array of values at which the PDF is evaluated; contents are the same as returned by meshgrid

pdf [array] array of marginal posterior probabilities at each point of the output grid, for each input cluster; the leading dimensions match the leading dimensions produced by broadcasting the leading dimensions of photprop and photerr together, while the trailing dimensions match the dimensions of the output grid

mpdf_gen(*fixeddim*, *fixedprop*, *margindim*, *ngrid=128*, *qmin=None*, *qmax=None*, *grid=None*, *norm=True*, *filters=None*)

Returns the marginal probability for one or more physical or photometric properties, keeping other properties fixed and marginalizing over other quantities. This is the most general marginal PDF routine provided.

Parameters:

fixeddim [int | arraylike of ints | None] The index or indices of the physical or photometric properties to be held fixed; physical properties are numbered 0 ... nphys-1, and photometric ones are numbered nphys ... nphys + nphot - 1. This can also be set to None, in which case no properties are held fixed.

fixedprop [array | None] The values of the properties being held fixed; the size of the final dimension must be equal to the number of elements in fixeddim, and if fixeddim is None, this must be too

margindim [int | arraylike of ints | None] The index or indices of the physical or photometric properties to be maginalized over, numbered in the same way as with fixeddim; if set to None, no marginalization is performed

ngrid [int or listlike containing ints] number of points in each dimension of the output grid; if this is an iterable, it must have nphys + nphot - len(fixeddim) - len(margindim) elements

qmin [float | arraylike] minimum value in the output grid in each quantity; if left as None, defaults to the minimum value in the library; if this is an iterable, it must contain a number of elements equal to nphys + nphot - len(fixeddim) - len(margindim)

qmax [float | arraylike] maximum value in the output grid in each quantity; if left as None, defaults to the maximum value in the library; if this is an iterable, it must have the same number of elements as qmin

grid [listlike of arrays] set of values defining the grid on which the PDF is to be evaluated, in the same format used by meshgrid

norm [bool] if True, returned pdf's will be normalized to integrate to 1

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns:

grid_out [array] array of values at which the PDF is evaluated; contents are the same as returned by meshgrid

pdf [array] array of marginal posterior probabilities at each point of the output grid, for each input set of properties; the leading dimensions match the leading dimensions produced by broadcasting the leading dimensions of photprop and photerr together, while the trailing dimensions match the dimensions of the output grid

mpdf_phot(*idx*, *physprop*, *ngrid=128*, *qmin=None*, *qmax=None*, *grid=None*, *norm=True*, *filters=None*)

Returns the marginal probability for one or more photometric quantities corresponding to an input set or distribution of physical properties. Output quantities are computed on a grid of values, in the same style as meshgrid.

Parameters:

idx [int or listlike containing ints] index of the photometric quantity whose PDF is to be computed, starting at 0; indices correspond to the order of elements in the filters argument; if this is an iterable, the joint distribution of the indicated quantities is returned

physprop [arraylike, shape (nphys) or (..., nphys)] physical properties to be used; if this is an array of nphys elements, these give the physical properties; if it is a multidimensional array, the operation is vectoried over the leading dimensions physical properties – the function must take an array of (nphys) elements as an input, and return a floating point value representing the PDF evaluated at that set of physical properties as an output

ngrid [int or listlike containing ints] number of points in each dimension of the output grid; if this is an iterable, it must have the same number of elements as idx

qmin [float or listlike] minimum value in the output grid in each quantity; if left as None, defaults to the minimum value in the library; if this is an iterable, it must contain the same number of elements as idx

qmax [float or listlike] maximum value in the output grid in each quantity; if left as None, defaults to the maximum value in the library; if this is an iterable, it must contain the same number of elements as idx

grid [listlike of arrays] set of values defining the grid on which the PDF is to be evaluated, in the same format used by meshgrid

norm [bool] if True, returned pdf's will be normalized to integrate to 1

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns:

grid_out [array] array of values at which the PDF is evaluated; contents are the same as returned by meshgrid

pdf [array] array of marginal posterior probabilities at each point of the output grid, for each input set of properties; the leading dimensions match the leading dimensions produced by broadcasting the leading dimensions of photprop and photerr together, while the trailing dimensions match the dimensions of the output grid

squeeze_rep(*x*, *wgts*, *dims=None*, *filters=None*)

Takes an input array of positions and weights that form a kernel density representation and approximates them using fewer points, using an error tolerance of reltol

Parameters:

x [array, shape (N, ndim)] an array of points forming a kernel density representation; on exit, x will be resized to (M, ndim) with M <= N

wgts [array, shape (N)] an array of weights for the kernel density representation; on exit, wgts will be resized to (M), with M <= N

dims [array, shape (ndim)] array specifying which dimensions in the kernel density representation the coordinates in x correspond to; if left as None, they are assumed to correspond to the first ndim dimensions in the data set

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns: Nothing

CHAPTER

FOURTEEN

SFR_SLUG: BAYESIAN INFERENCE OF STAR FORMATION RATES

The slugy.sfr_slug module computes posterior probabilities on star formation rates given a set of star formation rates estimated using the "point mass estimate" (i.e., the estimate you would get for a fully sampled stellar population) for the SFR based on the ionizing, FUV, or bolometric luminosity. It is implemented as a wrapper around *bayesphot: Bayesian Inference for Stochastic Stellar Populations*, so for details on how the calculation is performed see the bayesphot documentation.

14.1 Getting the Default Library

The sfr_slug module requires a pre-computed library of slug simulations to use as a "training set" for its calculations. Due to its size, the default library *is not* included in the slug git repository. Instead, it is provided for download from the SLUG data products website. Download the two files `SFR_SLUG_integrated_phot.fits` and `SFR_SLUG_integrated_prop.fits` and save them in the `sfr_slug` directory of the main respository. If you do not do so, and do not provide your own library when you attempt to use sfr_slug, you will be prompted to download the default library.

14.2 Basic Usage

The `sfr_slug/sfr_slug_example.py` file in the repository provides an example of how to use sfr_slug. Usage of is simple, as the functionality is all implemented through a single class, sfr_slug. The required steps are as follows:

1. Import the library and instantiate an `sfr_slug` object (see *Full Documentation of slugpy.sfr_slug* for full details):

```
from slugpy.sfr_slug import sfr_slug
sfr_estimator = sfr_slug()
```

This creates an sfr_slug object, using the default simulation library, $SLUG_DIR/sfr_slug/SFR_SLUG. If you have another library of simulations you'd rather use, you can use the `libname` keyword to the `sfr_slug` constructor to select it.

2. Specify your filter(s), for example:

```
sfr_estimator.add_filters('QH0')
```

The `add_filter` method takes as an argument a string or list of strings specifying which filters you're going to point mass SFRs based on. You can have more than one set of filters active at a time (just by calling `add_filters` more than once), and then specify which set of filters you're using for any given calculation.

3. Specify your priors, for example:

```
sfr_estimator.priors = 'schechter'
```

The `priors` property specifies the assumed prior probability distribution on the star formation rate. It can be either `None` (in which case all simulations in the library are given equal prior probability), an array with as many elements as there are simulations in the library giving the prior for each one, a callable that takes a star formation rate as input and returns the prior for it, or a string whose value is either "flat" or "prior". The two strings specify, respectively, a prior distribution that is either flat in log SFR or follows the Schechter function SFR distribution from Bothwell et al. (2011):

$$p(\log \mathrm{SFR}) \propto \mathrm{SFR}^{\alpha} \exp(-\mathrm{SFR}/\mathrm{SFR}_*)$$

with $\alpha = -0.51$ and $\mathrm{SFR}_* = 9.2\, M_\odot\, \mathrm{yr}^{-1}$.

4. Generate the posterior probability distribuiton of SFR via:

```
logSFR, pdf = sfr_estimator.mpdf(logSFR_in, logSFRphoterr = logSFR_err)
```

The argument `logSFR_in` can be a float or an array specifying one or more point mass estimates of the SFR in your chosen filter. For a case with two or more filters, then `logSFR_in` must be an array whose trailing dimension matches the number of filters. If you have added two or more filter sets, you need to specify which one you want to use via the `filters` keyword. The optional argument `logSFRphoterr` can be used to provide errors on the photometric SFRs. Like `logSFR_in`, it can be a float or an array.

The `sfr_slug.mpdf` method returns a tuple of two quantities. The first is a grid of log SFR values, and the second is the posterior probability distribution at each value of log SFR. If the input consisted of multiple photometric SFRs, the output will contains posterior probabilities for each input. The output grid will be created automatically be default, but all aspects of it (shape, size, placement of grid points) can be controlled by keywords – see *Full Documentation of slugpy.sfr_slug*.

14.3 Full Documentation of slugpy.sfr_slug

class slugpy.sfr_slug.**sfr_slug**(*libname=None*, *detname=None*, *filters=None*, *bandwidth=0.1*, *ktype='gaussian'*, *priors=None*, *sample_density='read'*, *reltol=0.001*, *abstol=1e-10*, *leafsize=16*)

A class that can be used to estimate the PDF of true star formation rate from a set of input point mass estimates of the star formation rate.

Properties

priors [array, shape (N) | callable | 'flat' | 'schechter' | None] prior probability on each data point; interpretation depends on the type passed; array, shape (N): values are interpreted as the prior probability of each data point; callable: the callable must take as an argument an array of shape (N, nphys), and return an array of shape (N) giving the prior probability at each data point; None: all data points have equal prior probability; the values 'flat' and 'schechter' use priors p(log SFR) ~ constant and p(log SFR) ~ SFR^alpha exp(-SFR/SFR_*), respectively, where alpha = -0.51 and SFR_* = 9.2 Msun/yr are the values measured by Bothwell et al. (2011)

bandwidth ['auto' | array, shape (M)] bandwidth for kernel density estimation; if set to 'auto', the bandwidth will be estimated automatically

__init__(*libname=None*, *detname=None*, *filters=None*, *bandwidth=0.1*, *ktype='gaussian'*, *priors=None*, *sample_density='read'*, *reltol=0.001*, *abstol=1e-10*, *leafsize=16*)

Initialize an sfr_slug object.

Parameters

libname [string] name of the SLUG model to load; if left as None, the default is $SLUG_DIR/sfr_slug/SFR_SLUG

detname [string] name of a SLUG model run with the same parameters but no stochasticity; used to establish the non-stochastic photometry to SFR conversions; if left as None, the default is libname_DET

filters [iterable of stringlike] list of filter names to be used for inferenence

bandwidth ['auto' | float | array, shape (M)] bandwidth for kernel density estimation; if set to 'auto', the bandwidth will be estimated automatically; if set to a float, the same bandwidth is used in all dimensions

ktype [string] type of kernel to be used in densty estimation; allowed values are 'gaussian' (default), 'epanechnikov', and 'tophat'; only Gaussian can be used with error bars

priors [array, shape (N) | callable | None] prior probability on each data point; interpretation depends on the type passed; array, shape (N): values are interpreted as the prior probability of each data point; callable: the callable must take as an argument an array of shape (N, nphys), and return an array of shape (N) giving the prior probability at each data point; None: all data points have equal prior probability

sample_density [array, shape (N) | callable | 'auto' | 'read' | None] the density of the data samples at each data point; this need not match the prior density; interpretation depends on the type passed; array, shape (N): values are interpreted as the density of data sampling at each sample point; callable: the callable must take as an argument an array of shape (N, nphys), and return an array of shape (N) giving the sampling density at each point; 'auto': the sample density will be computed directly from the data set; note that this can be quite slow for large data sets, so it is preferable to specify this analytically if it is known; 'read': the sample density is to be read from a numpy save file whose name matches that of the library, with the extension _density.npy added; None: data are assumed to be uniformly sampled

reltol [float] relative error tolerance; errors on all returned probabilities p will satisfy either abs(p_est - p_true) <= reltol * p_est OR abs(p_est - p_true) <= abstol, where p_est is the returned estimate and p_true is the true value

abstol [float] absolute error tolerance; see above

leafsize [int] number of data points in each leaf of the KD tree

Returns Nothing

Raises IOError, if the library cannot be found

__weakref__
list of weak references to the object (if defined)

add_filters(*filters*)
Add a set of filters to use for cluster property estimation

Parameters

filters [iterable of stringlike] list of filter names to be used for inferenence

Returns nothing

filters()
Returns list of all available filters

Parameters: None

Returns:

filters [list of strings] list of available filter names

logL(*logSFR*, *logSFRphot*, *logSFRphoterr=None*, *filters=None*)
This function returns the natural log of the likelihood function evaluated at a particular log SFR and set of log luminosities

Parameters:

logSFR [float or arraylike] float or array giving values of the log SFR; for an array, the operation is vectorized

logSFRphot [float or arraylike, shape (nfilter) or (..., nfilter)] float or array giving the SFR inferred from photometry using a deterministic conversion; for an array, the operation is vectorized over the leading dimensions

logSFRphoterr [float arraylike, shape (nfilter) or (..., nfilter)] float or array giving photometric SFR errors; for a multidimensional array, the operation is vectorized over the leading dimensions

filters [listlike of strings] list of photometric filters used for the SFR estimation; if left as None, and only 1 set of photometric filters has been defined for the sfr_slug object, that set will be used by default

Returns:

logL [float or arraylike] natural log of the likelihood function

mcmc(*photprop*, *photerr=None*, *mc_walkers=100*, *mc_steps=500*, *mc_burn_in=50*, *filters=None*)
This function returns a sample of MCMC walkers for log SFR

Parameters:

photprop [arraylike, shape (nfilter) or (..., nfilter)] array giving the photometric values; for a multidimensional array, the operation is vectorized over the leading dimensions

photerr [arraylike, shape (nfilter) or (..., nfilter)] array giving photometric errors; for a multidimensional array, the operation is vectorized over the leading dimensions

mc_walkers [int] number of walkers to use in the MCMC

mc_steps [int] number of steps in the MCMC

mc_burn_in [int] number of steps to consider "burn-in" and discard

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns

samples [array] array of sample points returned by the MCMC

mpdf(*logSFRphot*, *logSFRphoterr=None*, *ngrid=128*, *qmin=None*, *qmax=None*, *grid=None*, *norm=True*, *filters=None*)
Returns the marginal probability of log SFR for one or more input sets of photometric properties. Output quantities are computed on a grid of values, in the same style as meshgrid

Parameters:

logSFRphot [float or arraylike] float or array giving the log SFR inferred from photometry using a deterministic conversion; if the argument is an array, the operation is vectorized over it

logSFRphoterr [arraylike, shape (nfilter) or (..., nfilter)] array giving photometric errors; for a multidimensional array, the operation is vectorized over the leading dimensions

ngrid [int] number of points in the output log SFR grid

qmin [float] minimum value in the output log SFR grid

qmax [float] maximum value in the output log SFR grid

grid [array] set of values defining the grid of SFR values at which to evaluate; if set, overrides ngrid, qmin, and qmax

norm [bool] if True, returned pdf's will be normalized to integrate to 1

filters [listlike of strings] list of photometric filters to use; if left as None, and only 1 set of photometric filters has been defined for the cluster_slug object, that set will be used by default

Returns:

grid_out [array] array of log SFR values at which the PDF is evaluated

pdf [array] array of marginal posterior probabilities at each point of the output grid, for each input photometric value; the leading dimensions match the leading dimensions produced by broadcasting the leading dimensions of photprop and photerr together, while the trailing dimensions match the dimensions of the output grid

CHAPTER

FIFTEEN

TEST PROBLEMS

This section describes a set of problems that can be used to test and explore the different capabilities of SLUG. SLUG ships a set of problems `problemname` that are specified by a parameter file `param/problemname.param`. Problems that require multiple simulations are described instead by multiple paramater files, each with unique ID XX: `param/problemnameXX.param`. Users can reproduce the output of the test problems with the provided executable scripts `test/run_problemname.sh`. For each problem, a script for analysis is distributed in `test/problemname.py`. Details for each test problem are given below. Throughout this section, it is assumed that the `SLUG_DIR` has been properly set. These test problems are designed to work with outputs in FITS format, but that can be easily changed in the `.param` files. To run all the problems and the analysis scripts in one go, the user can simply run `test/run_alltest.sh`. It will take around 15 minutes for the script to complete on a standard laptop. About 700MB of data are generated. If SLUG is correctly installed and working, the first part of the script (i.e. the SLUG simulations) should run flawlessly. The second part of the script relies instead on external python procedures, including slugpy, numpy, and matplotlib. While these packages are fairly standard, the user needs to ensure that they are properly installed and visible to the script. This script has been written for and tested with Python 2.7.

15.1 Problem `example_galaxy`: basic galaxy simulation

This problem illustrates the basic usage of slugin `galaxy` mode by running 48 realizations of a galaxy with constant SFR $= 0.001\ M_\odot\ \mathrm{yr}^{-1}$, up to a maximum time of 2×10^8 yr. By issuing the command `test/run_example_galaxy.sh` the output files `SLUG_GALAXY_EXAMPLE*` are generated. Once the models are ready, `python test/plot_example_galaxy.py` produces a multi-panel figure `test/SLUG_GALAXY_EXAMPLE_f1.pdf`.

The top-left panel shows the actual mass produced by SLUG for each of the 48 models at different time steps as a function of the targeted mass. One can see that SLUG realizations only approximate the desired mass, which is a consequence of SLUG core algorithm. The 1:1 relation is shown by a red dashed line. The remaining panels show examples of integrated photometry (as labeled) of all simulated galaxies at different time steps, as a function of the actual mass. Due to its stochastic nature, SLUG produces distributions rather than single values for each time step. The expected rate of ionizing photon and the bolometric luminosities for a deterministic model with a continuous star formation rate of SFR $= 0.001\ M_\odot\ \mathrm{yr}^{-1}$ are shown by red dashed lines in the relevant panels.

15.2 Problem `example_cluster`: basic cluster simulation

This problem illustrates the basic usage of SLUG in `cluster` mode by running 1000 realizations of a cluster with mass 500 $M_\odot$, up to a maximum time of 10 Myr. By issuing the command `test/run_example_cluster.sh` the output files `SLUG_CLUSTER_EXAMPLE*` are generated. Once the models are ready, `python test/plot_example_cluster.py` produces a multi-panel figure `test/SLUG_CLUSTER_EXAMPLE_f1.pdf`.

This figure is divided in two columns: the left one shows outputs at the first time step, 1 Myr, while the second one shows outputs at the last time step, 10 Myr. The top row shows the actual cluster mass for an input mass of 500 $M_\odot$.

In `cluster` mode, all clusters are generated at the first time step and they evolve passively after that. Thus, the mass does not change. As a consequence of the random drawing from the IMF, masses are distributed around the input mass. As the wanted mass is large enough to allow for many stars to be drawn, the actual mass distribution is narrow.

The second row shows instead the distribution of the maximum mass of all stars that are still alive at a given time step. At 1 Myr, this distribution is a good approximation of the input distribution, which is the result of random draws from the IMF. At 10 Myr, which is the typical lifetime of a 15-20 $M_\odot$ star, the most massive stars have died, and SLUG stops following them. The distribution of luminosities, and particularly those most sensitive to the presence of massive stars, change accordingly (third and fourth row for Q_{H_0} and FUV).

15.3 Problem `constsampl`: importance of constrained sampling

This problem illustrates in more detail the effects of constrained sampling on SLUG simulations. This is the first key ingredient in the core algorithm of SLUG. With the command `test/run_constsampl.sh`, three different `cluster` simulations are run, each with 1000 trials, but with masses of 50 $M_\odot$, 250 $M_\odot$, and 500 $M_\odot$. A single timestep of 10^6 yr is generated. The analysis script `python test/plot_constsampl.py` produces a multi-panel figure `test/SLUG_CONSTSAMPL_f1.pdf`.

This figure shows the maximum mass of the stars in these realizations (top row), the rate of ionizing photons Q_{H_0} (central row), and the FUV luminosity (bottom row). Histograms refer, form left to right, to clusters with 50 $M_\odot$, 250 $M_\odot$, and 500 $M_\odot$.

Due to the small timestep, the distributions of stellar masses shown in the top panels reflect to good approximation the distribution of the maximum stellar masses that are drawn from the IMF by SLUG in each realization. For a cluster of 50 $M_\odot$, the vast majority of the stars are drawn below $20 - 50$ $M_\odot$. This is an obvious consequence of the fact that a cluster cannot contain stars much more massive than its own mass. However, stars more massive then the targeted mass are not impossible realizations for the default sampling algorithm (see below). For instance, if the first star to be drawn has mass 60 $M_\odot$, then SLUG would add it to the cluster and stop. Leaving this star out would indeed be a worse approximation than overshooting the targeted cluster mass by only 10 $M_\odot$. From left to right, one can see that, as the targeted cluster mass increases, the histogram shifts to progressively higher masses. In the limit of an infinite cluster, all stellar masses would be represented, and the histogram would peak at 120 $M_\odot$. Essentially, this constrained sampling introduces a stochastic (and not deterministic) variation in the IMF. An IMF truncated above 60 $M_\odot$ would roughly approximate the results of the left column; however, a deterministic cut-off would not correctly reproduce the non-zero tail at higher masses, thus artificially reducing the scatter introduced by random sampling.

The second and third row simply reflect what said above: for large clusters that can host stars at all masses, the luminosity peaks around what is expected according to a deterministic stellar population synthesis codes. At lower cluster masses, ionizing and UV fluxes are instead suppresses, due to the lack of massive stars. However, tails to high values exist in all cases.

15.4 Problem `sampling`: different sampling techniques

As highlighted in the previous section, the method with which stars are sampled from the IMF has a great influence on the final output. Starting from v2, SLUG has the capability of specifying the desired sampling algorithm for a given PDF. The command `test/run_sampling.sh` runs four `cluster` simulations, each with 1000 trials of masses of 50 $M_\odot$, and a Kroupa (2002) IMF. The following four sampling methods are chosen for each simulation: 1) `stop_nearest`, which is the default in SLUG; 2) `stop_before`; 3) `stop_after`; 4) `sorted_sampling`. A description of each method is provided in Section *Sampling Methods*. The analysis script `python test/plot_sampling.py` produces a multi-panel figure `test/SLUG_SAMPLING_f1.pdf`.

By comparing the panels in each column, one can understand the fundamental differences induced by the sampling technique. The top row shows the maximum stellar mass drawn from the IMF in each realization. The targeted cluster mass is also shown with red vertical lines. In the default mode, SLUG is allowed to overshoot the targeted mass if that

constitutes a good approximation for the total cluster mass. Thus, a tail at stellar masses above the targeted cluster mass is visible. This tail is accentuated when the stop after method is selected (third column). In this case, SLUG always overshoots the cluster mass, and thus extreme realizations above $100\ M_\odot$ are possible. Conversely, in the stop after method (second column), SLUG always under-fills the clusters, and (in this case) the cluster mass becomes a limit to the maximum stellar mass that can be drawn. A similar effect is seen when sorted sampling is enable (fourth column). However, the correspondence between the cluster mass and the maximum stellar mass is not trivially established, as it depends on the shape of the IMF. The second and third row show how the sampling techniques affect the output photometry.

15.5 Problem `imfchoice`: different IMF implementations

This problem highlights how SLUG can handle different IMF implementations by running three simulations with a Kroupa, a Salpeter, and a Chabrier IMF. However, SLUG is not restricted to these choices, as the user can in fact easily input an arbitrary IMF. The command `test/run_imfchoice.sh` runs three `cluster` simulations, each with 1000 trials of masses of $500\ M_\odot$ and different IMF. The analysis script `python test/plot_imfchoice.py` produces a multi-panel figure `test/SLUG_IMFCHOICE_f1.pdf`. Each column shows different statistics for the three IMF. From top to bottom, these are: the maximum stellar mass in a cluster, the number of stars that SLUG treats stochastically, and the distributions of Q_{H_0} and bolometric luminosities. As expected for a steep lower-end of the IMF, in the Salpeter case SLUG prefers to fill the clusters with a higher number of low mass stars.

15.6 Problem `clfraction`: cluster fraction at work

With the exception of the first example, these test problems have focused on how SLUG handles cluster simulations, and how these clusters are filled with stars drawn from the IMF. This new problem highlights instead the presence of additional stochasticity induced by a second level in the hierarchy of `galaxy` simulations: how clusters are drawn from the CMF to satisfy the targeted galaxy mass. Although it may not appear obvious at first, the fraction of stars that are formed in clusters, f_c, is a very important parameter that regulates the stochastic behavior of SLUG. This can be understood by considering two limiting cases. In the limit $f_c \rightarrow 0$, SLUG fills a galaxy by drawing stars from the IMF. Thus, because the mass of a galaxy is typically much larger than the mass of the upper end of the IMF, the effects of mass-constrained sampling highlighted in *Problem constsampl: importance of constrained sampling* are simply not relevant anymore. In this case, stochasticity is minimal. Conversely, in the limit $f_c \rightarrow 1$, not only the IMF sampling contributes to the stochastic behavior of SLUG, but also clusters themselves contribute to additional stochasticity, as clusters are now drawn from the CMF to fill the targeted galaxy mass following the similar rules to those specified for the IMF draws. Thus, in this case, constrained mass sampling applies to both stars in clusters and clusters in galaxies, and stochasticity is amplified.

The command `test/run_clfraction.sh` runs three `galaxy` simulations, each with 500 trials of continuous SFR $= 0.001\ \mathrm{M}_\odot\ \mathrm{yr}^{-1}$ which are evolved for a single timestep of 2×10^6 yr. A Chabrier IMF and a cluster mass function $\propto M^{-2}$ are adopted. Cluster disruption is disabled. The three simulations differ only for the fraction of stars formed in clusters, respectively $f_c = 1, 0.5, 0.01$. The analysis script `python test/plot_clfraction.py` produces a multi-panel figure `test/SLUG_CLFRACTION_f1.pdf`. Each column shows properties of simulations for different fractions of stars formed in clusters.

The top row shows the maximum stellar mass in clusters. Clearly, f_c has no effect on the way clusters are filled up with stars, but the normalization changes. Thus, the least probable realizations in the tail of the distribution simply do not appear for $f_c \rightarrow 0$. The second row shows the number of stars in clusters. Obviously, this scales directly with f_c, as it does the number of field stars in the third row. This is expected as, by definition, f_c regulates the number of stars in clusters versus the field. However, as discussed, f_c also affects the stochastic behavior of the simulation. The fourth row shows histograms of the actual galaxy mass versus the targeted mass (red line). As f_c increases, one can see that the spread around the targeted mass increase. This is again a consequence of the mass-constrained sampling and the stop-nearest condition. For $f_c \rightarrow 0$, the code tries to fill a galaxy of mass $0.001\ \mathrm{M}_\odot\ \mathrm{yr}^{-1} \times 2 \times 10^6$ yr with stars. Thus, since the targeted mass is at least a factor of 10 larger than the mass of the building block, SLUG

can approximate the desired mass very well (to better than 120 $M_\odot$, in fact). Conversely, for $f_c \rightarrow 1$, SLUG is using clusters as building blocks. As the typical mass of the building blocks is now more comparable to the targeted galaxy mass, the problem of the mass constrained sampling becomes a relevant one. Not only f_c affects the precision with which SLUG builds galaxies, but, as shown in the bottom row, it also affects photometry. One can see that Q_{H_0} increases as f_c decreases (the red lines indicate medians). The reason for this behavior should now be clear: in the case of clustered star formation ($f_c \rightarrow 1$), the mass of the most massive stars is subject to the mass constrained sampling of the IMF at the cluster level, reducing the occurrence of very massive stars and thus suppressing the flux of ionizing radiation. Conversely, for non clustered star formation ($f_c \rightarrow 0$), the sampling of the IMF is constrained only at the galaxy mass level, and since this is typically much greater than the mass of the most massive stars, one recovers higher fluxes on average.

15.7 Problem `cmfchoice`: different CMF implementations

Given the ability of SLUG v2 to handle generic PDFs, the user can specify arbitrary CMF, similarly to what shown in *Problem imfchoice: different IMF implementations*. The command `test/run_cmfchoice.sh` runs three `galaxy` simulations, each with 500 trials of continuous SFR $= 0.001$ $M_\odot$ yr^{-1} which are evolved for a single timestep of 2×10^6 yr. A Chabrier IMF and $f_c = 1$ are adopted. Cluster disruption is disabled. The three simulations differ only for the cluster mass function, which are: 1) the default powerlaw M^{-2} between $20 - 10^7$ $M_\odot$; 2) a truncated powerlaw M^{-2} between $20 - 100$ $M_\odot$; 3) a mass-independent CMF M^0 between $20 - 10^3$ $M_\odot$. The analysis script `python test/plot_cmfchoice.py` produces a multi-panel figure `test/SLUG_CMFCHOICE_f1.pdf`. Each column shows properties of simulations for the different cluster mass functions.

The top row shows the maximum stellar mass in clusters. Compared to the default case, the histogram of the truncated CMF is steeper towards low masses. Given that the upper end of the CMF is comparable to the maximum stellar mass of the chosen IMF, low stellar masses are typically preferred as a result of the stop-nearest condition. A flat CMF prefers instead more massive clusters on average, which in turn results in higher probabilities of drawing massive stars. In this case, the residual slope of the distribution towards low stellar masses is a result of the shape of the IMF. A reflection of the effects induced by the shape of the CMF are also apparent in the bottom row, which shows the distribution of ionizing photons from these simulations. The second row shows instead the difference between the targeted galaxy mass (red line), and the distribution of actual masses. The spread is minimal for the truncated CMF because, as discussed above, SLUG is using small building blocks, and it can approximate the targeted galaxy mass very well. Larger spread is visible in the case of the flat CMF, as this choice allows for clusters with masses up to 10^3 $M_\odot$, without imposing an excess of probability at the low mass end. The largest scatter is visible for the default case, as this CMF is virtually a pure powerlaw without cutoff at the high mass end, and thus clusters as massive as the entire galaxy are accessible to SLUG.

15.8 Problem `sfhsampling`: realizations of SFH

The algorithm at the heart of SLUG is quite simple: for a given star formation history $\dot{\psi}(t)$ a stellar population with mass $\dot{\psi}(t) \times \Delta t$ is generated at each timestep, according to the constraints set by IMF, CMF and other controlling parameters. As discussed in the previous examples, SLUG builds a best approximation for the targeted mass $\dot{\psi}(t) \times \Delta t$. This means that the input SFH and the output SFHs are not identical. SLUG receives an input SFH which is used to constrain the rate with which clusters and stars are drawn to achieve the desired targeted mass in each timestep. However, the output SFHs are only realizations and not exact copies of the input SFH. This problem is designed to illustrate this behavior.

The command `test/run_sfhsampling.sh` runs two `galaxy` simulations, each with 100 trials of continuous SFR $= 0.0001$ $M_\odot$ yr^{-1} which are evolved for a 10 timesteps of 5×10^6 yr. A Chabrier IMF and a M^{-2} CMF are adopted. Cluster disruption is disabled. The two simulations differ only for the fraction of stars in clusters, $f_c = 1$ and $f_c = 0$ respectively. The analysis script `python test/plot_sfhsampling.py` produces a two-panel figure

`test/SLUG_SFHSAMPLING_f1.pdf`, showing the box plot for the output SFH of the two simulations ($f_c = 1$ top, and $f_c = 0$ bottom).

In each panel, the median SFH over 100 trials is represented by the red lines, while the red squares show the mean. The box sizes represent instead the first and third quartile, with the ends of the whiskers representing the 5th and 95th percentiles. One can see that the input SFH at $\dot{\psi}(t) = 10^{-4}$ M$_\odot$ yr^{-1} is recovered on average, albeit with significant variation in each realization. The reason for this variation lies in the fact that, at low SFRs, SLUG samples the input SFH with coarse sampling points, which are clusters and stars. One can also notice a widely different scatter between the $f_c = 1$ and $f_c = 0$ case. In the former case, the basic elements used by SLUG to sample the targeted mass in a given interval are clusters. In the latter case, they are stars. Given that the typical mass of a cluster is of the same order of the targeted mass in each interval, the output SFH for the $f_c = 1$ case are more sensitive to the history of drawings from the CMF. Conversely, for $f_c = 0$, the sampling elements are less massive than the targeted mass in a given interval, resulting in an output SFH distribution which is better converged towards the input value. Clearly, a comparable amplitude in the scatter will be present in the output photometry, especially for the traces that are more sensitive to variations in the SFHs on short timescales.

15.9 Problem `cldisrupt`: cluster disruption at work

One additional ingredient in SLUG is the lifetime distribution for clusters. Since v2, SLUG is flexible in controlling the rate with which clusters are disrupted. This problem shows a comparison between two simulations with and without cluster disruption.

The command `test/run_cldisrup.sh` runs two `galaxy` simulations, each with 100 trials which are evolved in timesteps of 5×10^5 yr up to a maximum age of 1×10^7 yr. Both simulations are characterized by a burst of star formation $= 0.001$ M$_\odot$ yr^{-1} within the first Myr. A Chabrier IMF and a M^{-2} CMF are adopted, and $f_c = 1$. For the first simulation, cluster disruption is disabled. In the second simulation, cluster disruption operates at times > 1 Myr, with a cluster lifetime function which is a powerlaw of index -1.9. The analysis script `python test/plot_cldisrup.py` produces the figure `test/SLUG_CLDISRUP_f1.pdf`. The two columns show results with (right) and without (left) cluster disruption.

The first row shows the median stellar mass of the 100 trials as a function of time. The blue dashed lines show the mass inside the galaxy, while the black solid lines show the median mass in clusters. The red band shows the first and fourth quartile of the distribution. One can see that in both cases the galaxy mass rises in the first Myr up to the desired targeted mass of $= 1000$ M$_\odot$ given the input SFH. After 1Myr, star formation stops and the galaxy mass does not evolve with time. Conversely, the cluster mass (black line, red regions) evolves differently. In the case without cluster disruption, because $f_c = 1$, the cluster mass tracks the galaxy mass at all time. When cluster disruption is enabled (right), one can see that the mass in clusters rise following the galaxy mass in the first Myr. Past that time, clusters start being disrupted and the mass in clusters declines. The same behavior is visible in the second row, which shows the median number of alive (black) and disrupted (black) clusters. To the left, without cluster disruption, the number of clusters alive tracks the galaxy mass. Conversely, this distribution declines with time to the right when cluster disruption is enabled. The complementary quantity (number of disrupted clusters) rises accordingly. The last two rows show instead the integrated fluxes in FUV and bolometric luminosity. Again, medians are in black and the first and third quartiles in red. One can see a nearly identical distribution in the left and right panels. In these simulations, the controlling factors of the integrated photometry are the SFH and the sampling techniques, which do not depend on the cluster disruption rate. Clearly, the photometry of stars in cluster would exhibit instead a similar dependence to what shown in the top panels.

15.10 Problem `spectra`: full spectra

Since v2, SLUG is able to generate spectra for star clusters and for galaxies, which can also be computed for arbitrary redshifts. This problem highlights the new features. It also demonstrates how SLUG can handle dust extinction, both in a deterministic and stochastic way.

The command `test/run_spectra.sh` runs four `galaxy` simulations, each with 500 trials of continuous SFR $= 0.001$ $\mathrm{M}_\odot$ yr^{-1} which are evolved for a single timestep of 2×10^6 yr. A Chabrier IMF and a M^{-2} CMF are adopted, cluster disruption is disabled, and $f_c = 1$. The simulations differ in the following way: 1) the reference model, computed without extinction and at $z = 0$; 2) same as the reference model, but at $z = 3$; 3) same as the reference model at $z = 0$, but with a deterministic extinction of $A_V = 0.5$ and a Calzetti+2000 starburst attenuation curve; 4) same as model number 3, but with stochastic extinction. The analysis script `python test/plot_spectra.py` produces the figure `test/SLUG_SPECTRA_f1.pdf`, which shows a gallery of galaxy SEDs for each model. The median SED is shown in black, the blue region corresponds to the first and third quartile of the distribution, and the red shaded region marks the 5 and 95 percentiles.

The top panel shows the default model, where stochasticity occurs as detailed in the previous examples. The second panel from the top shows instead a model with deterministic extinction. This is simply a scaled-down version of the reference model, according to the input dust law and normalization coefficient A_V. As the dust law extends only to 915 Angstrom the output SED is truncated. The third panel shows that, once SLUG handles dust in a stochastic way, the intrinsic scatter is amplified. This is a simple consequence of applying dust extinction with varying normalizations, which enhances the final scatter about the median. Finally, the bottom panel shows the trivial case in which the spectrum is shifted in wavelength by a constant factor $(1 + z)$. Obviously, redshift enhances the stochasticity in the optical due to a simple shift of wavelengths.

CHAPTER

SIXTEEN

USING SLUG AS A LIBRARY

In addition to running as a standalone program, SLUG can be compiled as a library that can be called by external programs. This is useful for including stellar population synthesis calculations within some larger code, e.g., a galaxy simulation code in which star particles represent individual star clusters, where the stars in them are treated stochastically.

16.1 Compiling in Library Mode

To compile in library mode, simply do:

```
make lib
```

in the main directory. This will cause a dynamically linked library file `libslug.x` to be created in the `src` directory, where `x` is whatever the standard extension for dynamically linked libraries on your system is (`.so` for unix-like systems, `.dylib` for MacOS).

Alternately, if you prefer a statically-linked version, you can do:

```
make libstatic
```

and a statically-linked archive `libslug.y` will be created instead, where `y` is the standard statically-linked library extension on your system (generally `.a`).

In addition to `lib` and `libstatic`, the makefile supports `lib-debug` and `libstatic-debug` as targets as well. These compile the same libraries, but with optimization disabled and debugging symbols enabled.

Finally, if you want MPI functionality, you can compile with:

```
make lib MPI=ENABLE_MPI
```

See *Compiling* for more on compiling with MPI enabled.

16.2 Predefined Objects

In order to make it more convenient to use slug as a library, the library pre-defines some of the most commonly-used classes, in order to save users the need to construct them. These predefined objects can be accessed by including the file `slug_predefined.H` in your source file. This function defines the class `slug_predef`, which pre-defines all the IMFs, evolutionary tracks, spectral synthesizers, and yields that ship with slug, without forcing the user to interact with the parameter parsing structure.

The `slug_predef` class provides the methods `imf`, `tracks`, `specsyn`, and `yields`. These methods take as arguments a string specifying one of the predefined names of an IMF, set of tracks, or spectral synthesizer, and return

an object of that class that can then be passed to `slug_cluster` to produce a cluster object. For example, the following sytax creates a `slug_cluster` with ID number 1, a mass of 100 solar masses, age 0, a Chabrier IMF, Padova solar metallicity tracks, starburst99-style spectral synthesis, and slug's default nuclear yields:

```
#include "slug_predefined.H"
#include "slug_cluster.H"

slug_cluster *cluster =
   new slug_cluster(1, 100.0, 0.0, slug_predef.imf("chabrier"),
                    slug_predef.tracks("modp020.dat"),
                    slug_predef.specsyn("sb99"),
                    nullptr, nullptr, nullptr,
                    slug_predef.yields());
```

16.3 Using SLUG as a Library with MPI-Enabled Codes

In large codes where one might wish to use slug for subgrid stellar models, it is often necessary to pass information between processors using MPI. Since slug's representation of stellar populations is complex, and much information is shared between particles rather than specific to individual particles (e.g., tables of yields and evolutionary tracks), passing slug information between processors is non-trivial.

To facilitate parallel implementations, slug provides routines that wrap the base MPI routines and allow seamless and efficient exchange of the slug_cluster class (which slug uses to represent simple stellar populations) between processors. The prototypes for these functions are found in the `src/slug_MPI.H` header file, and the functions are available if the library was compiled with MPI support enabled (see *Compiling in Library Mode*).

Here is an example of MPI usage, in which one processor creates a cluster and then sends it to another one:

```
#include "slug_cluster.H"
#include "slug_MPI.H"
#include "mpi.h"
#include <vector>
#include <cstdio>

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

  // Start MPI
  MPI_Init(&argc, &argv);

  // Get rank
  int rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  // Rank 0 creates a cluster and prints out the masses of the stars
  slug_cluster *cluster;
  if (rank == 0) {
    cluster =
       new slug_cluster(1, 100.0, 0.0, slug_predef.imf("chabrier"),
                        slug_predef.tracks("modp020.dat"),
                        slug_predef.specsyn("sb99"),
                        nullptr, nullptr, nullptr,
                        slug_predef.yields());
    const std::vector<double> stars = cluster->get_stars();
    for (int j=0; j<stars.size(); j++)
      std::cout << "rank 0, star " << j
                << ": " << stars[j] << std::endl;
```

```
  }

  // Barrier to make sure rank 0 outputs come first
  MPI_Barrier(MPI_COMM_WORLD);

  // Rank 0 sends cluster, rank 1 receives it
  if (rank == 0) {
    MPI_send_slug_cluster(*cluster, 1, 0, MPI_COMM_WORLD);
  } else if (rank == 1) {
    cluster = MPI_recv_slug_cluster(0, 1, MPI_COMM_WORLD,
                                    slug_predef.imf("chabrier"),
                                    slug_predef.tracks("modp020.dat"),
                                    slug_predef.specsyn("sb99"),
                                    nullptr, nullptr, nullptr,
                                    slug_predef.yields());
  }

  // Rank 1 prints the masses of the stars; the resulting masses
  // should be identical to that produced on rank 0
  if (rank == 1) {
    const std::vector<double> stars = cluster->get_stars();
    for (int j=0; j<stars.size(); j++)
      std::cout << "rank 1, star " << j
                << ": " << stars[j] << std::endl;
  }
}
```

CHAPTER

SEVENTEEN

CONTRIBUTORS AND ACKNOWLEDGEMENTS

The following people contributed to slug2:

- Mark Krumholz: primary author of slug2
- Michele Fumagalli: primary author of the slug2 test suite, co-author of version 1 of slug
- Robert da Silva: primary author of version 1 of slug and of sfr_slug, wrote the first prototype version of slug2 and sfr_slug
- Greg Ashworth: wrote the variable PDF and high-resolution UV modules
- Jonathan Parra: contributed code that become part of the slug_PDF module
- Teddy Rendahl: wrote the first version of cloudy_slug
- Michelle Myers: contributed to the development of cluster_slug
- Evan Demers: wrote the first version of the yield module
- Yusuke Fujimoto: helped debug callable library mode, and wrote an interface between slug and enzo

In addition to these direct contributors, we gratefully acknowledge the following people who provided some of the data on which slug relies:

- The library of stellar evolutionary tracks and stellar atmospheres is taken from Claus Leitherer's starburst99 package.
- Much of the library of photometric filters is taken from Charlie Conroy's FSPS package.
- Daniela Calzetti provided the extinction curves
- Tuguldur Sukhbold provided the core collapse supernova yield tables.
- Amanda Karakas and Carolyn Doherty provided the AGB yield tables.

CHAPTER

EIGHTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

D

F

L

M