slug Documentation

Release 2.0

Mark Krumholz, Robert da Silva, Michele Fumagalli, Jonathan Par

October 03, 2014

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

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 paper, da Silva, R. L., Fumagalli, M., & Krumholz, M. R., 2012, The Astrophysical Journal, 745, 145. A second method paper, describing the upgraded version 2 code and a set of ancillary tools, is in preparation at this time.

2.1 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. 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 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).

2.2 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.

2.3 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.

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.

2.4 Spectra, Photometry, and Extinction

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, it computes photometry for the stellar population by integrating the computed spectra over a set of specified photometric filters. If extinction is enabled, photometric values are computed for both the unextincted and the extincted spectrum. 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

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

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

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

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

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

- The ST magnitude, defined by

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

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

- The Vega magnitude, defined by

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

where $M_{\rm AB}({\rm Vega})$ 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,

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

Fourth and finally, if extinction is enabled, SLUG also computes an extincted spectrum

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

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. Photometric values for the extincted spectrum are computed exactly as for the unextincted one.

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).

2.5 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.

CHAPTER

THREE

COMPILING AND INSTALLING SLUG

3.1 Dependencies

The core SLUG program requires

- The Boost C++ libraries
- The GNU scientific library
- The cfitsio library (optional, only required for FITS capabilities)

Compilation will be easiest if you install these libraries such that the header files are included in your `CXX_INCLUDE_PATH` and the compiled object files are in your `LD_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). 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.

3.2 Compiling

If you have boost, GSL, and (if you're using it) cfitsio included in your `CXX_INCLUDE_PATH` and `LD_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 compile without cfitsio, do:

```
make FITS=DISABLE_FITS
```

Alternately, you can manually specify the compiler flags to be used 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`.

Finally, 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.

CHAPTER

FOUR

RUNNING A SLUG SIMULATION

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*.

If you have more than one core at your disposal, you can also run SLUG in parallel, using 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:

- `-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.

CHAPTER

FIVE

PARAMETER SPECIFICATION

5.1 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.
- Some keywords have default values, indicated in parenthesis in the list below. These keywords are optional and need not appear in the parameter file. All others 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.
- Any time a file or directory is specified, if it is given as a relative rather than absolute path, it is assumed to be relative to the environment variable `$SLUG_DIR`. If this environment variable is not set, it is assumed to be relative to the current working directory.

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

5.2 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: `output`): name of the directory into which output should be written.
- `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.

5.3 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
- `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).
- `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`.
- `end_time`: last output time, in yr. 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.
- `sfr`: star formation rate. Only used if `sim_type` is `galaxy`; for `cluster`, it will be ignored, and can be omitted. If, instead of specifying a numerical value for this parameter, you specify 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.
- `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.

5.4 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_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`.
- `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.

5.5 Physical 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: `lib/tracks/Z0140v00.txt`): stellar evolution tracks to use. The following tracks ship with SLUG**
  - `ZXXXXvYY.txt`: 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`).
  - `modcXXX.dat`: 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`).
  - `modeXXX.dat`: same as `modcXXX.dat`, but with higher mass loss rates.
  - `modpXXX.dat`: Padova tracks with thermally pulsing AGB stars; metallicities use the same scale as `modcXXX.dat` files (i.e., `020` is Solar).
  - `modsXXX.dat`: same as `modpXXX.dat`, but without thermally pulsing AGB stars
- `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`: metallicity of the stellar population, relative to solar. This may be omitted if `tracks` is set to one of the default sets of tracks that ships with SLUG, as the metallicities for these tracks are hardwired in. This keyword is provided to allow users to supply their own tracks.
- `WR_mass`: minimum starting mass that stars must have in order to pass through a Wolf-Rayet phase. This can be omitted if `tracks` is set to one of the default sets of tracks that ships with SLUG, as the WR cutoff masses for these tracks are hardwired in. This keyword is provided to allow users to supply their own tracks.
- `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

5.6 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 H^0 ionizing luminosity, in photons/sec
  - `QHe0`: the He^0 ionizing luminosity, in photons/sec
  - `QHe1`: the 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

CHAPTER

SIX

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.

6.1 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 6.1: 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_*

6.2 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 $M_\odot\ {\rm yr}^{-1}$.

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.

6.3 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

SEVEN

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 six 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`, `cluster_prop`, `cluster_spec`, or `cluster_phot`. The production of these output files is controlled by the parameters `out_integrated`, `out_integrated_spec`, `out_integrated_phot`, `out_cluster`, `out_cluster_spec`, and `out_cluster_phot` in the parameter file. The files are formatted as described 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

7.1 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:

- `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`: actual mass of stars produced in the galaxy up to that time, and which have not yet reached the end of their lives (as marked by the final entry in the stellar evolution tracks)
- `ClusterMass`: actual mass of stars produced in the galaxy up to that time that are still members of non-disrupted 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 *Physical Model Keywords*)

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

- Time (double)
- TargetMass (double)
- ActualMass (double)
- LiveMass (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)

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.

7.2 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:

- Time: evolution time at which the output is produced
- Wavelength: observed frame wavelength at which the spectrum is evaluated
- L_lambda: specific luminosity at the specified wavelength, without extinction
- L_lambda_ex: specific luminosity at the specified wavelength after extinction is applied (only present if SLUG was run with extinction enabled)

If output_mode is ascii, these data are output in a series of columns, with different trials separated by lines of dashes. Note that some entries for L_lambda_ex may be blank. This indicates wavelengths for which a stellar atmosphere model was available, but outside the wavelength range included in the user-specified extinction curve file. Extincted luminosities are unavailable at these wavelengths due to the lack of an extinction curve.

> 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; if extinction was enabled in the SLUG calculation, it also contains a field Wavelength_ex listing the wavelengths at which the extincted spectrum is computed. The second table has three fields if SLUG was run without extinction, or four fields if it was run with extinction. The first three fields, always present, give the trial number, the time, and the spectrum L_lambda at that time. The final field, L_lambda_ex, gives the extincted spectrum at that time and trial. 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

- 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 spectra
- Wavelength (NWavelength entries of type double)
- NWavelength_ex (std::vector<double>::size_type, usually unsigned long long): the number of wavelength entries in the extinted spectra; only present if Extinct is 1
- Wavelength_ex (NWavelength_ex entries of type double); only present if Extinct is 1

and then contains a series of records in the format

- Time (double)
- L_lambda (NWavelength entries of type double)
- L_lambda_ex (NWavelength_ex entries of type double); only present if Extinct is 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.

7.3 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:

- 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_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
- ...

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 extinction enabled, in for each filter FILTERNAME there is a corresponding column FILTERNAME_ex containing the photometric value for that filter applied to the extincted spectrum. Some of these 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
- `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

- `Time` (`double`)
- `PhotFilter` (`NFilter` entries of type `double`)
- `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).

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.

7.4 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`: actual mass of stars produced in the cluster at this output time that have not yet reached the end of their lives (as marked by the final entry in the stellar evolution tracks)
- `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 *Physical 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 *Physical Model Keywords*)
- `A_V`: visual extinction for that cluster, in mag; present only if SLUG was run with extinction enabled

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

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)
- NumStar (std::vector<double>::size_type, usually unsigned long long)
- MaxStarMass (double)
- A_V (double); present only if Extinct is 1

7.5 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 spectrum is evaluated
- L_lambda: specific luminosity at the specified wavelength
- L_lambda_ex: specific luminosity at the specified wavelength after extinction is applied (only present if SLUG was run with extinction enabled)

If output_mode is ascii, these data are output in a series of columns, with different trials separated by lines of dashes. The column L_lambda_ex is present only if SLUG was run with extinction enabled. Some entries for L_lambda_ex 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; if extinction was enabled in the SLUG calculation, it also contains a field Wavelength_ex listing the wavelengths at which the extincted spectrum is computed. The second table has four fields if SLUG was run without extinction, or five fields if it was run with extinction. The first four fields, always present, give the trial number, unique ID of the cluster, the time, and the spectrum L_lambda at that time. The final field, L_lambda_ex, gives the extincted spectrum at that time and trial. Both the ASCII- and FITS-formatted output should be fairly self-documenting.

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

- 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 spectra
- Wavelength (NWavelength entries of type double)
- NWavelength_ex (std::vector<double>::size_type, usually unsigned long long): the number of wavelength entries in the extinted spectra; only present if Extinct is 1
- Wavelength_ex (NWavelength_ex entries of type double); only present if Extinct is 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_ex` (`NWavelength_ex` entries of type `double`); only present if `Extinct` is 1

7.6 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_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`
- `...`

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 extinction enabled, in for each filter `FILTERNAME` there is a corresponding column `FILTERNAME_ex` containing the photometric value for that filter applied to the extincted spectrum. Some of these values may be `NaN`; see *The integrated_phot File*. In addition to the time, unique ID, 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
- `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_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).

CHAPTER

EIGHT

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, Photometry, and Extinction* 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

NINE

CLOUDY_SLUG: AN AUTOMATED INTERFACE TO CLOUDY

SLUG stochastically generates stellar spectra, but it does not compute the nebular lines produced when those photons interact with the interstellar medium. To perform such calculations, 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.

9.1 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*). If running in integrated mode (the default – see *The cloudy_slug Physical Model: Integrated Mode Versus Cluster Mode*), integrated specta and photometry are required, and if running in cluster mode, 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: Integrated Mode Versus Cluster Mode* 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.

9.2 The cloudy_slug Physical Model: Integrated Mode Versus Cluster Mode

Associating nebular emission with the stellar populations produced by SLUG requires some assumptions about geometry, and some choices about what quantities one is interested in computed. SLUG outputs both integrated spectra for all the stars in a galaxy, and spectra for individual clusters. One on hand, one could make the exteme assumption that all the star clusters are spatially close enough to one another that one can think of the entire galaxy as a single giant HII region, and compute the nebular emission for the galaxy as a whole. This may be a reasonable assumption for galaxies where the star formation is highly spatially-concentrated. At the other extreme, one may assume that there is no overlap whatsoever between the HII regions surrounding different star clusters, so that nebular emission should be computed for each one independently. This may be a reasonable assumption for extended, slowly star-forming systems like the outer disk of the Milky Way. Each of these assumptions entails somewhat different choices about how to set the inner radius and inner density the HII region, as required by cloudy. The cloudy_slug interface can compute nebular emission under either of these scenarios; we refer to the former as integrated mode, and to the latter as cluster mode. Note that, in either mode, the spectrum that is used to compute the nebular emission will be the *unextincted, non-redshifted* spectrum computed by SLUG.

9.2.1 Integrated Mode

In integrated mode, cloudy_slug will read all the spectra contained in the SLUG integrated_spec output file, and for each stellar spectrum it will perform a cloudy run to produce a calculation of the nebular emission produced by that stellar spectrum interacting with a surrounding HII region. The density in the first zone of the HII region will be as specified by the standard `hden` keyword in the cloudy input template (see *The cloudy_slug Input Template*). The inner radius of the HII region will be computed automatically, and will be set to 10^{-3} of the Stromgren radius for that density, where

$$r_{\rm St} = \left(\frac{3.0 Q({\rm H}^0)}{4\pi\alpha_B n_{\rm H}^2}\right)^{1/3}$$

where $Q({\rm H}^0)$ is the ionizing luminosity computed by SLUG, $n_{\rm H}$ is the hydrogen number density stored in the cloudy input template, and α_B is the case B recombination coefficient, which is taken to have a value of 2.59×10^{-13} cm^3 s^{-1}.

9.2.2 Cluster Mode

In cluster mode, cloudy_slug will read all the individual cluster spectra contained in the SLUG cluster_spec file, and for each one it will perform a cloudy calculation to determine the corresponding nebular emission. The density and radius are handled somewhat differently in this case, since, for a mono-age stellar population, it is possible to compute the time evolution of the HII region radius and density.

In cluster mode, the hydrogen number density $n_{\rm H}$ stored in the cloudy input template (see *The cloudy_slug Input Template*) is taken to specify the density of the *neutral* gas around the HII region, not the density of the gas inside the HII region. The outer radius of the HII region is then computed using the approximate analytic solution for the expansion of an HII region into a uniform medium, including the effects of radiation presssure and stellar wind momentum deposition, given by Krumholz & Matzner (2009, ApJ, 703, 1352). The radius is computed from the

ionizing luminosity $Q(\mathrm{H}^0)$, hydrogen number density n_{H}, and star cluster age t as

$$r_{\mathrm{II}} = r_{\mathrm{ch}} \left(x_{\mathrm{II,rad}}^{7/2} + x_{\mathrm{II,gas}}^{7/2} \right)^{2/7}$$
$$x_{\mathrm{II,rad}} = (2\tau^2)^{1/4}$$
$$x_{\mathrm{II,gas}} = (49\tau^2/36)^{2/7}$$
$$\tau = t/t_{\mathrm{ch}}$$
$$r_{\mathrm{ch}} = \frac{\alpha_B}{12\pi\phi} \left(\frac{\epsilon_0}{2.2 k_B T_{\mathrm{II}}} \right)^2 f_{\mathrm{trap}}^2 \frac{\psi^2 Q(\mathrm{H}^0)}{c^2}$$
$$t_{\mathrm{ch}} = \left(\frac{4\pi\mu m_{\mathrm{H}} n_{\mathrm{H}} c r_{\mathrm{ch}}^4}{3 f_{\mathrm{trap}} Q(\mathrm{H}^0) \psi \epsilon_0} \right)^{1/2}$$

where $\alpha_B = 2.59 \times 10^{-13}$ cm^3 s^{-1} is the case B recombination coefficient, $\phi = 0.73$ is the fraction of ionizing photons absorbed by hydrogen atoms rather than dust, $\epsilon_0 = 13.6$ eV is the hydrogen ionization potential, $T_{\mathrm{II}} = 10^4$ K is the temperature inside the HII region, $f_{\mathrm{trap}} = 2$ is the trapping factor that accounts for stellar wind and trapped infrared radiation pressure, $\psi = 3.2$ is the mean photon energy in Rydberg for a fully sampled IMF at zero age, and $\mu = 1.33$ is the mean mass per hydrogen nucleus for gas of the standard cosmic composition. See Krumholz & Matzner (2009) for a discussion of the fiducial choices of these factors.

Once the outer radius is known, cloudy_slug sets the starting radius for the cloudy calculation to $10^{-3} r_{\mathrm{II}}$, and sets the starting density to the value expected for photoionization equilibrum in a uniform HII region,

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

Note that this approximation will be highly inaccurate if $r_{\mathrm{II}} \ll r_{\mathrm{ch}}$, but no better analytic approximation is available, and this phase should be very short-lived for most clusters.

9.3 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 *must not* contain a radius command. This too will be computed automatically by the cloudy_slug script.
3. The input file *must* contain an entry `hden N` where `N` is the log base 10 of the hydrogen density. This will be interpreted differently depending on whether cloudy_slug is being run in cluster mode or integrated mode – see *The cloudy_slug Physical Model: Integrated Mode Versus Cluster Mode*.
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. 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.
8. 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.

9.4 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] [-nl NICELEVEL]
               [-n NPROC] [-s] [--slugpath SLUGPATH] [-v]
               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 (in *Integrated Mode*) or cluster spectra (in *Cluster Mode*). If this argument is set, cloudy will only be run in spectra starting with the specified trial number (in *Integrated Mode*) or cluster number (in *Cluster Mode*); numbers are 0-offset, to the first trial/cluster is 0, the next is 1, etc.
- `end_spec`: default behavior is to run cloudy on all the integrated spectra (in *Integrated Mode*) or cluster spectra (in *Cluster Mode*). If this argument is set, cloudy will only be run on spectra up to the specified trial number (in *Integrated Mode*) or cluster number (in *Cluster Mode*); numbers are 0-offset, to the first trial/cluster is 0, the next is 1, etc.

The optional arguments are as follows:

- `-h, --help`: prints a help message and then exits
- `-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 4 Myr. This argument only has an effect if running in *Cluster Mode*; otherwise it is ignored.
- `--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`
- `-cm, --clustermode`: if this argument is set, then cloudy_slug will run in *Cluster Mode*; default behavior is to run in *Integrated Mode*

- `-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
- `-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. WARNING: cloudy's outputs are written in ASCII and are quite voluminous, so only choose this option if you are only running cloudy on a small number of SLUG spectra and/or you are prepared to store hundreds of GB more more.
- `--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`
- `-v, --verbose`: if this option is set, cloudy_slug produces verbose output as it runs

9.5 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 run in *Integrated Mode*, the three output files will be `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 in *Cluster Mode*, the three output files will be `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 those files is described below.

9.5.1 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

- `Time` (`double`)
- `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.

9.5.2 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:

- `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

- `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.

9.5.3 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:

- `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.

9.5.4 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`)

9.5.5 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`)

9.5.6 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

CHAPTER

TEN

SLUGPY – THE PYTHON HELPER LIBRARY

10.1 Basic Usage

SLUG comes with the python module slugpy, which contains an extensive set of routines for reading, writing, and manipulating SLUG outputs. 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, Photometry, and Extinction*
- filter_wl_c: pivot wavelength used to set the normalization for each filter for which $\beta \neq 0$ – see *Spectra, Photometry, and Extinction*
- phot: photometry in each filter

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 may contain fewer elements than wl_ex, 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. May contain fewer entries than spec because the extinction curve does not cover the full wavelength range of the computed stellar spectra.
- 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 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).

10.2 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_info=None*)

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_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.

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

(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

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

(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

(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

(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

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*)

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 [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

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; 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

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_ex [array, shape (N_filter, N_times, N_trials)] same as phot, but after extinction has been applied (present only if SLUG was run with extinction enabled)

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

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

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 [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] 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

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)

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 [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

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_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)

Raises IOError, if no spectrum file can be opened

`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*)

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.

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)] 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

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

(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

(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

slugpy.**read_integrated_phot**(*model_name*, *output_dir=None*, *fmt=None*, *nofilterdata=False*, *photsystem=None*, *verbose=False*, *read_info=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 [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] times at which colors 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; 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

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_ex [array, shape (N_filter, N_times, N_trials)] same as phot, but after extinction has been applied (present only if SLUG was run with extinction enabled)

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*)

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 [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] Times at which data are output

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

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

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 [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] times at which spectra are output, in yr

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_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)

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 [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.

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

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

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

10.3 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_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_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] times at which line luminosities 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_lines, N_times, N_trials)] luminosity of each line at each time for each trial, in erg/s

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] times at which 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_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] times at which spectra are output, in yr

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_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_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

10.4 Full Documentation of slugpy.sfr_slug

class `slugpy.sfr_slug.`**`sfr_slug`**(*libname=None*, *detname=None*, *bandwidth=0.1*)
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.

Attributes

dataset [array] the training dataset to be used for KDE estimation

dataset_filters [list of string] filters represented in the training data set

conversions [array] conversions between luminosity and SFR in the point mass estimate

kde [KernelDensity object] a KernelDensity estimator constructed from the dataset

kde_filters [list of string] filters represented in the KDE object

kde_limits [array] range over which KDE is non-zero

libname [string] name of the SLUG model from which the data set was read

detname [string] name of a SLUG model run with the same parameters as libname, but with no stochasticity

__call__(*logsfr_est*, *logsfr_err=None*, *filter_name=None*, *nmesh=100*, *logsfr_lim=None*, *prior=None*, *error_est=False*, *gkorder=None*)

Return an estimate of the PDF of true star formation rate for one or more point mass estimates of the SFR using a particular photometric filter.

Parameters

logsfr_est [float or array] a point mass estimate of log_10 SFR; can be a float, 1D array or 2D array. For a 1D array the data are taken to be a series of point mass estimates of log_10 SFR. For a 2D array, the trailing dimension must match the length of filter_name, and the data are taken to represent one or more log_10 SFR point mass estimates using different filters

logsfr_err [float or array] error on logsfr_est; must be the same shape as logsfr_est; if left as None, data are assumed to have negligible errors and are treated as delta functions

filter_name [string or iterable] name or names of filters used for photometric SFR estimates; if left as None, stored values are used

nmesh [int] number of mesh points at which to evaluate the SFR; note that accurate normalization requires that this not be too small

logsfr_lim [array, shape (2)] limit the log SFR values considered to lie in the range logsfr_lim[0] to logsfr_lim[1]

prior [callable] a callable that returns the prior probability distribution of log SFR; if set to None, the prior simply matches the distribution of input models

error_est: bool if True, an estimate of the error in the numerical convolution of the observaitonal uncertainties with the model is returned; ignored if logsfr_err is None

gkorder [string] order Gauss-Kronrod quadrature; allowed values are '15', '21', '31', '41', '51', '61'; default is '61' for a single filter, and '15' for >1 filter; this parameter has no effect unless logsfr_err is not None

Returns A namedtuple consisting of:

logsfr [array, shape (nmesh)] an array of log_10 SFR values at which the PDF is evaluated

sfrpdf [array, shape (nmesh) or shape (nmesh, ndata)] the PDF of log_10 SFR evaluate at the points in the logsfr array; if logsfr_est is a float, this will be a 1D array, while if logsr_est is an array whose leading dimension has ndata elements, it will be a 2D array where entry [:,M] gives the PDF for the Mth input data value

pdf_err [array, shape (nmesh) or shape (nmesh, ndata)] an estimate of the numerical error in pdf; returned only if error_est is True

Raises ValueError, if filter_name is not one of the filters available in dataset_filters

__init__(*libname=None*, *detname=None*, *bandwidth=0.1*)

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

bandwidth [float] bandwidth of the kernel to use in density estimates

Returns Nothing

Raises IOError, if the library cannot be found

__weakref__
list of weak references to the object (if defined)

get_kde(*filter_name*)
Builds a KernelDensity object to estimate the joint PDF of the dataset in one or more filters, then returns it. Same as set_kde, but the kde is returned rather than being stored interally.

Parameters

filter_name [string or iterable] name or names of filters; the string 'SFR' corresponds to the true star formation rate

Returns

kde [KernelDensity object] the computed KernelDensity object

kde_limts [array, shape (N_dim, 2)] range over which the kde is non-zero; element [:,0] gives the minimum in each dimension, and [:,1] gives the maximum

Raises ValueError, if filter_name is not one of the filters available in dataset_filters

pdf(*x*, *filter_name=None*, *nosave=False*)
Return the PDF of the data set in one or more filters.

Parameters

x [array] point or points at which the PDF is to be evaluated; the trailing dimension of x must match the number of elements in filter_name

filter_name [string or iterable] name or names of filters used for photometric SFR estimates, or 'SFR' for the true SFR; if left as None, the currently-stored filters are used

nosave [bool] if True, the KDE constructed as part of the computation is not saved; if False, it is saved and overwrites the existing KDE

Returns

logpdf [array] log of the value of the PDF evaluated at each of the input points

Raises ValueError, if filter_name is left as None and no filters are set

pdfgrid(*filter_name=None*, *nmesh=50*, *lim=None*, *nosave=False*)
Return the PDF of the data set in one or more filters, evaluated on a uniformly-spaced grid over the data.

Parameters

filter_name [string or iterable] name or names of filters used for photometric SFR estimates; if left as None, the currently-stored filters are used

nmesh [int] number of sample points per dimension to use in constructing a sampling grid

lim [array, shape (Ndim, 2)] limits of the sampling grid; element [i,0] gives the lower limit in the dimension corresponding to filter_name[i], and element [i,1] gives the upper limit; if left as None, limits are chosen automatically to fit the data set

nosave [bool] if True, the KDE constructed as part of the computation is not saved; if False, it is saved and overwrites the existing KDE

Returns

xlim [array, shape (Ndim, 2)] limits of the grid over which the data was evaluated; Ndim is the number of dimensions of the data, which is equal to the number of elements in filter_name

logpdf [ndarray] log of the value of the PDF evaluated at each of the input points

Raises ValueError, if filter_name is left as None and no filters are set

set_kde(*filter_name*)

Builds a KernelDensity object to estimate the joint PDF of the dataset in one or more filters

Parameters

filter_name [string or iterable] name or names of filters used for photometric SFR estimates, or 'SFR' for the true SFR; if left as None, the currently-stored filters are used

Returns Nothing

Raises ValueError, if filter_name is not one of the filters available in dataset_filters

CHAPTER

ELEVEN

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.

11.1 Problem `example`: basic galaxy simulation

This problem illustrates the basic usage of slugin `galaxy` mode by running 48 realizations of a galaxy with constant $\mathrm{SFR} = 0.001\ M_\odot\ \mathrm{yr}^{-1}$, up to a maximum time of 2×10^8 yr. By issuing the command `test/run_example.sh` the output files `SLUG_EXAMPLE*` are generated. Once the models are ready, `python test/plot_example.py` produces a multi-panel figure `test/SLUG_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 $\mathrm{SFR} = 0.001\ M_\odot\ \mathrm{yr}^{-1}$ are shown by red dashed lines in the relevant panels.

11.2 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 raw 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).

11.3 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.

11.4 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`, 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`.

This problem highlights the flexible choice of sampling techniques in SLUG, which is a new capability of v2.

[basic run with different sampling]

11.5 Problem `imfchoice`: different IMF implementations

This problem highlights the flexible choice of IMF implementations in SLUG.

[basic run with different imfs]

11.6 Problem `cmfchoice`: different CMF implementations

This problem highlights the flexible choice of CMF implementations in SLUG.

[basic run with different cmfs]

11.7 Problem `sfhsampling`: realizations of SFH

This problem illustrates the conceptual difference between an input SFH and the effective realizations produced by SLUG, in comparison to deterministic codes.

[show how a input SFH gets implemented in different realizations]

11.8 Problem `cldisrupt`: cluster disruption at work

This problem highlights the flexible choice of CLF implementations in SLUG.

[basic run with different clfs]

11.9 Problem `clfraction`: cluster fraction at work

This problem highlights the flexible choice of cluster fraction during SLUG simulations.

[basic run with different fc]

11.10 Problem `spectra`: full spectra

This problem highlights the power of the new feature offered in SLUG v2: the ability to produce full spectra.

[basic run with full spectra out: shows stochasticity applied to spectra]

11.11 Problem `redshift`: trivial redshift example

This problem shows a trivial example of the redshift capability in SLUG v2.

[basic run with full spectra out at a different redshift]

CHAPTER

TWELVE

ACKNOWLEDGEMENTS

SLUG was made possible by contributions from a number of people. The original SLUG v1 code was written by Robert da Silva and Michele Fumagalli. This reimplementation was primarily written by Mark Krumholz and Michele Fumagalli, with contributions from Jonathan Parra. The slugpy library was written by Mark Krumholz, with contributions from Teddy Rendahl (who wrote the first version of cloudy_slug) and Robert da Silva (who wrote the first version of sfr_slug). In addition to these contributors, several other people 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.
- The library of photometric filters is taken from Charlie Conroy's FSPS package.
- Daniela Calzetti provided the extinction curves

CHAPTER

THIRTEEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

S

C

P

R

S

W