User's Guide for `slug` v. 2.0

Mark Krumholz

August 21, 2014

Contents

1 License and Citations

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. 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 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), and cluster lifetime function (CLF), 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 IMF, SFH, CMF, and CLF 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).

2.1 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.

2.2 Probability Distribution Functions: the IMF, SFH, CMF, and CLF

As mentioned above, `slug` regards the IMF, SFH, CMF, and CLF as probability distribution functions. These PDFs can be described by a very wide range of possible functional forms; see Section 6 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 Section 6.

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. 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. 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.

2.3 Spectra and Photometry

Once `slug` has drawn a population of stars, its final step is to compute the light they produce. `Slug` does this in several steps. First, it computes the physical properties of all the stars present user-specified times using a set of stellar evolutionary tracks. Second, it uses these physical properties to compute the composite spectra produced by the stars, using a user-specified set of stellar atmosphere models. Formally, the quantity computed is the specific luminosity per unit wavelength L_λ. Third and finally, it computes photometry for the stellar population by integrating the computed spectra over a set of specified photometric filters. Depending on the options specified by the user and the filter under consideration, the photometric value output will be one of the following:

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

$$\langle L_\nu \rangle_R = \frac{\int L_\nu \, d\ln\nu}{\int R_\nu (\nu/\nu_c)^\beta \, d\ln\nu}, \tag{1}$$

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

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

$$\langle L_\lambda \rangle_R = \frac{\int L_\lambda \, d\ln\lambda}{\int R_\lambda (\lambda/\lambda_c)^{-\beta} \, d\ln\lambda}, \tag{2}$$

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

$$M_{\rm AB} = -2.5 \log_{10} \left[\frac{\langle L_\nu \rangle_R}{4\pi \left(10\,{\rm pc}\right)^2} \right] - 48.6, \tag{3}$$

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

- The ST magnitude, defined by

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

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

- The Vega magnitude, defined by

$$M_{\rm Vega} = M_{\rm AB} - M_{\rm AB}({\rm Vega}), \tag{5}$$

where $M_{\rm AB}$(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

$$Q(\nu_0) = \int_{\nu_0}^{\infty} \frac{L_\nu}{h\nu}\, d\nu. \tag{6}$$

- The bolometric luminosity,

$$L_{\rm bol} = \int_0^{\infty} L_\nu\, d\nu. \tag{7}$$

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).

2.4 Monte Carlo Simulation

The steps described in the previous two simulations 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, in order to build up statistics.

3 Compiling and Installing `slug`

3.1 Dependencies

The core `slug` program requires

- The boost C++ libraries
- The GNU scientific library
- The cfitsio library (optional)

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 man 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).

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

- numpy
- scipy
- astropy (optional)

As with cfitsio, astropy is only required if you want the ability to read and manipulate FITS output.

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, then 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 $\geq$ 4.7, clang/llvm $\geq$ 3.3, icc $\geq$ 14.0. Earlier versions may work as well, but no guarantees.

4 Running a `slug` Simulation in Serial or Parallel

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 Section 5. The code will write a series of output files as described in Section 7.

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.

5 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` (`SLUG_DEF`): name of the model. This will become the base filename for the output files.
- `out_dir` (`output`): name of the directory into which output should be written.
- `verbosity` (`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` (`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` (`1`): number of trials to run
- `log_time` (`0`): set to 0 for logarithmic time step, 1 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 Section 6. 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`.

- `redshift` (`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 keyword

These control what quantities are computed and written to disk.

- `out_cluster` (`1`): write out the physical properties of star clusters? Set to 1 for yes, 0 for no.

- `out_cluster_phot` (`1`): write out the photometry of star clusters? Set to 1 for yes, 0 for no.

- `out_cluster_spec` (`1`): write out the spectra of star clusters? Set to 1 for yes, 0 for no.

- `out_integrated` (`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` (`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` (`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` (`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; see Section 7 for a description of the output format. 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, etc.

- `imf` (`lib/imf/chabrier.imf`): name of the IMF descriptor file; this is a PDF file, formatted as described in Section 6. Note that `slug` ships with the following IMF files pre-defined (in the directory `lib/imf`)
  - `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`, `http://www.stsci.edu/science/starburst99/docs/default.htm`)
  - `salpeter.imf` (single-component power law IMF from Salpeter, 1955, ApJ, 121, 161)
- `cmf` (`lib/cmf/slug_default.cmf`): name of the CMF descriptor file; this is a PDF file, formatted as described in Section 6. 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`.
- `clf` (`lib/clf/slug_default.clf`): name of the CLF descriptor file; this is a PDF file, formatted as described in Section 6. 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` (`lib/tracks/Z0140v00.txt`): stellar evolution tracks to use. The following tracks ship with `slug` (all in the directory `lib/tracks`):
  - `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× Solar (`040`), Solar (`020`), 0.4× Solar (`008`), 0.2× Solar (`004`), and 0.05× 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` (`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` (`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 (Hiller & 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` (`1.0`): fraction of stars formed in clusters

- `min_stoch_mass` (`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.

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 whitespace-separated. For a list of available photometric filters, see the file `lib/filters/FILTER_LIST`. In addition to these filters, `slug` always allows four special "bands":

  - `QH0`: the H^0 ionizing luminosity, in photons sec^{-1}

  - `QHe0`: the He^0 ionizing luminosity, in photons sec^{-1}
  - `QHe1`: the He^+ ionizing luminosity, in photons sec^{-1}
  - `Lbol`: the bolometric luminosity, in $L_\odot$

- `filters` (`lib/filters`): directory containing photometric filter data

- `phot_mode` (`L_nu`): photometric system to be used when writing photometric outputs. Allowed values are:

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

  Full definitions of the quantities computed for each of these choices are given in Section 2.3. Note that these values are ignored for the four special bands `QH0`, `QHe0`, `QHe1`, and `Lbol`. These four bands are always written out in the units specified above.

6 Probability Distribution Functions

The `slug` code regards the IMF, the CMF, the CLF, and the SFH as probability distribution functions – see Section 2.2. 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

$$\frac{dp}{dx} = n_1 f_1(x; x_{1,a}, x_{1,b}) + n_2 f_2(x; x_{2,a}, x_{2,b}) + n_3 f_3(x; x_{3,a}, x_{3,b}) + \cdots, \tag{8}$$

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) $n_i f_i(x_{i,b}; x_{i,a}, x_{i,b}) = n_{i+1} f_{i+1}(x_{i+1,a}; x_{i+1,a}, x_{i+1,b})$, and (3) the overall PDF is normalized such that $\int (dp/dx)\, dx = 1$.[1] Given these constraints, the PDF can be specified

[1] 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.

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.

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

$$\frac{dp}{dm} \propto \begin{cases} \exp[-\log(m/m_0)^2/(2\sigma^2)](m/m_b)^{-1}, & m < m_b \\ \exp[-\log(m_b/m_0)^2/(2\sigma^2)](m/m_b)^{-2.35}, & m \geq m_b \end{cases} , \tag{9}$$

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

Name	Functional form	Keyword	Meaning	Keyword	Meaning
`delta`	$\delta(x - x_a)^2$				
`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_*

Table 1: Types of PDF segments and corresponding keywords.

```
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 Table 1. 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.

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
```

[2] A `delta` PDF must be defined by a segment where $x_a = x_b$. It is an error if $x_a \neq x_b$.

```
# 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^-2 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   1.0e6          # Form 10^6 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   1.0e6
scale    1.0e7

segment
type exponential
min      2.0e8
max      3.0e8
weight   1.0e6
scale    1.0e7

segment
type exponential
min      3.0e8
max      4.0e8
weight   1.0e6
scale    1.0e7

segment
type exponential
min      4.0e8
max      5.0e8
```

```
weight   1.0e6
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

$$\dot{M}_* = ne^{-(t \bmod P)/t_{\rm dec}}, \tag{10}$$

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 $(1/P) \int_0^P \dot{M}_* \, dt/ = 0.01 \ M_\odot$ yr^{-1}, i.e., that the mean SFR averaged over a single burst period is 0.01 $M_\odot$ yr^{-1}.

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

```
advanced
```

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

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

The `type` keyword is exactly the same as in basic mode, as are the segment-specific parameter keywords `key1`, `key2`, ... The same functional forms, listed in Table 1, 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

$$w_i = n_i \int_{x_{i,a}}^{x_{i,b}} f_i(x) \, dx. \tag{11}$$

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 10^6 $M_\odot$ 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.

7 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. 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 Ångstrom
- Specific luminosities are in erg s^{-1} $Å^{-1}$
- 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 Section 5.5)

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

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 output FITS file has two binary table extensions. The first table contains a single field listing the wavelengths at which the spectra are given. The second table has three fields, giving the trial number, the time, and the spectrum `L_lambda` at that time. 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

- NWavelength (std::vector<double>::size_type, usually unsigned long long): the number of wavelength entries in the spectra
- Wavelength (NWavelength entries of type double)

and then contains a series of records in the format

- Time (double)
- L_lambda (NWavelength entries of type double)

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 Section 5.6)
- PhotFilter2
- PhotFilter3
- ...

If output_mode is ascii, these data are output in a series of columns, with different trials separated by lines of dashes. If output_mode is fits, the data are stored as a series of columns in a binary table extension to the FITS file; the filter names and units are included in the header information for the columns. 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

This is followed by a series of entries of the form

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

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 Section 5.5)
- `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 Section 5.5)

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, 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)

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

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 output FITS file has two binary table extensions. The first table contains a single field listing the wavelengths at which the spectra are given. The second table has four fields, giving the trial number, the unique ID of the cluster, the time, and the spectrum L_lambda. Both the ASCII- and FITS-formatted output should be fairly self-documenting.

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

- NWavelength (std::vector<double>::size_type, usually unsigned long long): 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` (`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`)

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 Section 5.6)
- `PhotFilter2`
- `PhotFilter3`
- ...

If `output_mode` is `ascii`, these data are output in a series of columns, with different trials separated by lines of dashes. If `output_mode` is `fits`, the data are stored as a series of columns in a binary table extension to the FITS file; the filter names and units are included in the header information for the columns. 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

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`)

8 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 Section 5.6) 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 2.3 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 Ångstrom, and the second is the filter response function at that wavelength.

9 `slugpy` – The Python Helper Library