Generating mock catalogs and images with EGG: the Empirical Galaxy Generator

Contents

1 Installing EGG

1.1 Forewords

EGG is written in C++ and has a few dependencies. I have tried to keep the number of these dependencies as low as possible, and in fact for the moment there are four:

- *phy*$_{++}$, a library for numerical analysis that I have developed during my PhD,
- cfitsio, for handling FITS files,
- WCSlib, for handling sky-to-pixel conversions,
- and CMake, for managing the building process (dependencies, and platform specific stuff).

1.2 Install dependencies

If your operating system comes with a package manager, this should be very easy. Apart from *phy*$_{++}$ that we will address in the next section, these dependencies are standard libraries and tools that should be available in all the package managers.

- Mac users:

  ```
  sudo port install cfitsio wcslib cmake
  ```

  or

  ```
  sudo brew install cfitsio wcslib cmake
  ```

- Linux/Ubuntu users:

  ```
  sudo apt-get install libcfitsio3-dev wcslib-dev cmake
  ```

- Other Linux distributions: You get the point. Use `yum`, `apt`, `pacman`, or whatever package manager is supported by your distribution.
- Windows users: no package manager (see below).

If you don't have a package manager, then you have to compile these tools and libraries yourself... I hope it doesn't come to that, because you may loose a lot of time figuring this out. But in the eventuality, here are the links to places where you can download the source code. Follow the build instructions given on their respective each web page.

- cfitsio: `http://heasarc.gsfc.nasa.gov/fitsio/fitsio.html`
- WCSlib: `http://www.atnf.csiro.au/people/mcalabre/WCS/`
- CMake: `http://www.cmake.org/download/` (they also offer binaries, check this out first)

1.3 Install *phy*$_{++}$ and EGG: the short way

Once the dependencies are properly installed, you can download, build and install the *phy*$_{++}$ library and EGG. Thanks to CMake, the installing process is the same on all computers, and is rather straightforward. The next section describes each steps in detail, in case you are not familiar with CMake, and is the recommended way to go.

However, if you don't have the patience to spend a few minutes on this, or if you are completely lost, you can use the `install.sh` script provided in the `doc/script/` directory. The script does exactly what is written in the next section, and only requires you to specify three parameters ("Configurable options", at the beginning of the script):

- `INSTALL_ROOT_DIR`: This is the directory in which *phy*$_{++}$ and EGG will be installed. If you leave it blank, the script will use the default value adequate for your system (e.g., `/usr/local`). This directory is a "root" directory, in the sense that C++ headers will be installed in `$INSTALL_ROOT_DIR/include`, while executables will be installed in `$INSTALL_ROOT_DIR/bin`, etc.

- `CFITSIO_ROOT_DIR` and `WCSLIB_ROOT_DIR`: These directories tell the script where to find the cfitsio and WCSlib libraries. As above, this is a "root" directory: it must contain the header files in the `include` subdirectory, and the library files in `lib`. Leave it blank if you have installed these libraries through your package manager, or if you installed them manually in the default system folders.

Once you have modified these parameters (if need be), just make sure the script is executable and run it (you can run it from anywhere, the current directory does not matter):

```
chmod +x install.sh
./install.sh
# give your 'sudo' password, if asked
```

In the end, the script will inform you that EGG was successfully installed. If not, please fall back to the manual installation described in the next section.

1.4 Install *phy*$_{++}$ and EGG: the long way

1. Make yourself a temporary directory and open a terminal there.

2. Download the following archives and extract them in this directory:

   - https://github.com/cschreib/phypp/archive/master.tar.gz
   - https://github.com/cschreib/egg/archive/master.tar.gz

   This bash script will do that for you:

   ```
   wget https://github.com/cschreib/phypp/archive/master.tar.gz
   tar -xvzf master.tar.gz && rm master.tar.gz
   wget https://github.com/cschreib/egg/archive/master.tar.gz
   tar -xvzf master.tar.gz && rm master.tar.gz
   ```

   In the end, this should create two directories:

   ```
   egg-master
   phypp-master
   ```

3. Open a terminal and navigate to the `phypp-master` directory. Then, if you are using cfitsio and WCSlib from your package manager, run the following commands:

   ```
   mkdir build && cd build
   cmake ../
   ```

   If instead you have installed one of these two libraries by hand, in a non-standard directory, you have to provide this directory to the CMake script. This is done this way:

   ```
   mkdir build && cd build
   cmake ../ -DWCSLIB_ROOT_DIR=... -DCFITSIO_ROOT_DIR=...
   ```

   The "..." have to be replaced by the actual directory in which each library was installed. For example, if you have installed cfitsio in the `/opt/local/share/cfitsio` directory, then the "..." after `-DCFITSIO_ROOT_DIR` in the above command has to be replaced by `"/opt/local/share/cfitsio"`.

   Another thing to consider is that, by default, CMake will try to install the library inside the system default directories (e.g., `/usr/local/include`). If you have root access to your computer, this is the recommended way to go as it will make things simpler. If you do not have root access, or if for some reason you prefer to install the library somewhere else than the default location, you can specify manually the install directory using `-DCMAKE_INSTALL_PREFIX`:

   ```
   mkdir build && cd build
   cmake ../ -DCMAKE_INSTALL_PREFIX=...
   ```

   Here, the "..." have to be replaced by the root directory in which you want to install the library. For example, if you want to install it in `/opt/local/include`, just replace "..." by `"/opt/local"` (yes, omit `include`). This can of course be combined with the manual installation directories for WCSlib and cfitsio.

   If all goes well, this will configure the *phy*$_{++}$ library and prepare it for installation. The script will most likely warn you about missing dependencies, but this is OK since none of these are needed for EGG. Just make sure that cfitsio and WCSlib are found correctly, then install the library with the following command:

   ```
   sudo make install
   # or just 'make install' if you do not need root access
   ```

4. Using the terminal, navigate now inside the `egg-master` directory. Similarly, run the following commands:

   ```
   mkdir build && cd build
   cmake ../
   ```

   As for *phy*$_{++}$, the default behavior is to install EGG in the system directory (e.g., `/usr/local/bin`). If you want to change this, use instead:

   ```
   mkdir build && cd build
   cmake ../ -DCMAKE_INSTALL_PREFIX=...
   ```

   Again, the "..." have to be replaced by the directory in which you want to install the programs. For example, if you want to install it in `/opt/local/bin`, just replace "..." by `"/opt/local"` (yes, omit `bin`).

   Note that if you have installed the *phy*$_{++}$ library in a non-standard folder, for example in the `/opt/local` directory (as above), you will have to manually specify this location to CMake and use instead:

```
mkdir build && cd build
cmake ../ -DPHYPP_ROOT_DIR="/opt/local"
```

The same is true for WCSlib and cfitsio: if you had to specify their location manually when installing *phy*$_{++}$, you will have to repeat this here.

CMake will generate an error if, somehow, there was an issue in the installation of the *phy*$_{++}$ library. Else, it will configure EGG and make it ready to be built. Finally, run the last command:

```
sudo make install
# or just 'make install' if you do not need root access
```

At the end of the process, CMake will remind you in which directory the EGG executables are installed, e.g.:

```
-- Installed: /usr/local/bin/egg-gencat
-- Installed: /usr/local/bin/egg-genmap
-- Installed: /usr/local/bin/egg-gennoise
-- Installed: /usr/local/bin/egg-buildmf
-- Installed: /usr/local/bin/egg-2skymaker
```

If you chose a non-standard install directory, make sure this directory is in your `PATH`, and you are done. See, not that hard!

1.5 Making sure everything works

Navigate into a directory of your choosing and call:

```
egg-gencat verbose maglim=27 selection_band=hst-f160w area=0.08
```

This will take a few seconds to run. In the end, you should get something like:

```
note: initializing filters...
note: initializing SED libraries...
note: initializing redshift bins...
note: min dz: 0.1, max dz: 1.00632
note: 33 redshift slices
note: estimating redshift-dependend mass limit...
note: will generate masses from as low as 5.50406, up to 12
note: reading mass functions...
note: found 19 redshift bins and 181 mass bins
note: generating redshifts...
note: generated 66229 galaxies
note: generating masses...
note: generating sky positions...
[---------------------------------------] 33 100%, 57ms elapsed, 0ns left, 57ms total
note: generating morphology...
note: generating SFR...
note: assigning optical SEDs...
[---------------------------------------] 900 100%, 169ms elapsed, 0ns left, 169ms total
[---------------------------------------] 900 100%, 168ms elapsed, 0ns left, 168ms total
note: generate IR properties...
note: assigning IR SED...
note: computing fluxes ...
[---------------------------------------] 66229 100%, 10s elapsed, 0ns left, 10s total
[---------------------------------------] 66229 100%, 14s elapsed, 0ns left, 14s total
note: saving catalog...
```

Also, a file called `egg-2015xxxx.fits` (e.g., for me it was `egg-20151127.fits`) weighting about 27MB will be created in the same directory. This is the output catalog, in FITS format. You can open it in IDL to check its content with the following IDL command:

```
; Load the catalog
cat = mrdfits('egg-2015xxxx.fits', 1)
; Look at its content
help, cat, /str
; Then do some plots
plot, cat.z, cat.m, psym=3, xtit='redshift', ytit='stellar mass'
```

Then it remains to test the program that will translate this catalog into a *SkyMaker*-compatible catalog, one per band. Try:

```
egg-2skymaker cat=egg-2015xxxx.fits verbose band=hst-f160w \
    template=goodss-hst-f160w.conf
```

This will use one of the pre-defined *SkyMaker* template to configure a *Hubble* image at the same quality as in the GOODS–*South* field. This should be very fast and print a few lines in the terminal:

```
note: reading catalog...
note: found 66229 galaxies
note: convert parameters for ingestion by SkyMaker...
note: define image size and pixel coordinates...
note: image dimensions: 17951,17967 (1.2015 GB)
note: write catalog...
note: done.
```

In addition, three files should have been created in the same directory, `egg-2015xxxx-hst-f160w.cat` (the *SkyMaker* input catalog), `egg-2015xxxx-hst-f160w-hdr.txt` (the WCS header to feed to *SkyMaker*), and `egg-2015xxxx-sky.conf` (the *SkyMaker* configuration file).

If you have *SkyMaker* installed on your computer, you can generate the corresponding image by simply running:

```
sky egg-2015xxxx-hst-f160w.cat -c egg-2015xxxx-hst-f160w-sky.conf
```

This should take a couple of minutes (77 seconds on my computer), and finally produce a *Hubble* image in `egg-2015xxxx-hst-f160w-sci.fits` (about 1.3GB in size). You can open it with DS9 and admire the Universe you just created. Congratulations!

2 Using EGG

As you may have seen from the previous section, EGG is actually composed of several different tools, each taking care of a different step of the simulation process:

- `egg-gencat`: creates a new EGG mock catalog,
- `egg-getsed`: pick the complete spectrum of a galaxy from an SED data base created by `egg-gencat`,
- `egg-2skymaker`: convert an EGG catalog into a *SkyMaker* input catalog,
- `egg-gennoise`: create an empty image with noise,
- `egg-genmap`: paint galaxies from an EGG catalog to an empty map created with `egg-gennoise`.

In this section, I will describe the features and capabilities of each of these programs. You can refer to this documentation if you need help understanding this or that command line argument, or if you want to discover new features that you didn't know existed. Alternatively, each program can provide you with some limited help on the spot, if you simply call

```
egg-[xxx] help
```

in the terminal (where `[xxx]` is to be replaced by the name of the program you want the help for).

2.1 Generic information (applies to all programs)

2.1.1 Command line arguments

All the tools in the EGG suite are compiled into binary executables. This means you do not need to run Python, IDL, or any other interpreter to launch them. It also means they are fast!

To specify the parameters of each programs, EGG relies exclusively on command line arguments, rather than configuration files (as done, e.g., with *T-PHOT*, *SkyMaker* or *SExtractor*). The reason why is that it allows easy scripting with *bash* (or your favorite shell), and also allows you to experiment and tweak the parameters directly inside the terminal rather than having to go back and forth between the terminal and the configuration file.

The syntax for command line arguments is simple: `variable=value` or `-variable=value` (the two are perfectly equivalent). Spaces are not allowed on either side of the `=` sign, and if `value` contains spaces, you must wrap it within double quotes: `variable="some value"`. If you want to provide an array of values, the syntax is: `variable=[value1,value2,...]`, with no space. If at least one of the values does contain a space, you must wrap the whole array within double quotes:
`variable="[some value1,value2 foo,...]"`
(in this case, spaces can be used freely within the quotes). Lastly, if you want to provide multiple command line arguments, simply put one (or more) space between each argument: `x1=10 x2=5`. The order of the arguments does not matter.

Regarding the *type* of the argument. From the command line, everything is a string of characters. So there is no difference between `"0.08"` and `0.08`, or `"foo"` and `foo`. The values that you provide are translated into numbers/booleans/whatever by the program, using the standard C++ parsing. This means in particular that you can use scientific notation for large/small numbers (`-1e56`). Also, if the parameter that you want to modify is a boolean (i.e., either `0` or `1`), you can simply write the command line argument as: `variable` (without a value), which is equivalent to `variable=1`.

Some examples:

```
egg-gencat area=0.08         # good
egg-gencat area = 0.08       # bad! don't put spaces around '='

egg-gencat note="my catalog v1.0" # good
egg-gencat note=my catalog v1.0   # bad! missing the quotes "..."

egg-gencat area=0.08 zmin=2 # good
egg-gencat area=0.08zmin=2  # bad! need a space between each argument

egg-gencat bands=[hst-f160w,spitzer-irac1]     # good
egg-gencat bands="[hst-f160w,spitzer-irac1]"   # good also
egg-gencat bands=[hst-f160w, spitzer-irac1]    # bad! cannot use a space in the array
egg-gencat bands="[hst-f160w, spitzer-irac1]" # good! with the quotes, it is fine
```

2.1.2 Control what is written to the standard output (terminal)

By default, all the tools in the EGG suite run in "silent" mode: unless an error happens, they will not print anything in the terminal. This is mostly useful for scripted usage. However, if you want to make sure you understand what is going on, it is recommended to set the `verbose` command line flag. It is available for all the tools, and will let each program print information about what it is doing; a short description of each step of their execution, or even progress bars for the longest steps. This is designed to alter the performances in a non-noticeable way, so do not hesitate to use it.

2.1.3 Rules for ASCII table formatting

Unfortunately, there are various ways an ASCII table can be defined, and building a parser to support them all is a challenging (if not impossible) task. For simplicity, the EGG tools only support one such definition. It reads as follows:

- The file may start with a header. It is purely descriptive and will not be read by the program. This header can span one or multiple lines, and each of these lines must start with the `'#'` character.
- Empty lines are allowed and ignored.
- Each column must be separated by spaces (or tabulations) only. Properly aligning the columns is recommended for human readability, but is not mandatory.
- Values in the table may not contain any space. Using quotes will not help you.
- Missing values (i.e., empty "cells" in the table) are forbidden and cannot be understood by the program. Use a placeholder value instead (for floating point numbers, `nan` is a good choice).
- All floating point formats are accepted: fixed point (`1.5`) and scientific (`1.5e15`). The special floating point values `+inf`, `-inf` and `nan` are accepted (case does not matter).
- All the values that contain non-numeric characters that are not covered with the above rules are considered as strings.
- A column may only contain values of a unique type: it is forbidden to mix numbers and strings.

2.1.4 Column-oriented FITS tables

For historical reasons tied to IDL, EGG and *phy*$_{++}$ (the underlying C++ library) exclusively support column-oriented FITS tables. Although this is a perfectly valid way of doing things (according to the FITS standard), this is not the standard way FITS tables are used in general. It has, however, a number of advantages that I will not describe here (take a look at the *phy*$_{++}$ documentation if you are interested). Instead, this section will tell you how you can read and write such kind of FITS tables.

In C++. Using the *phy*$_{++}$ library, reading and writing these tables is natural:

```
// First declare the columns you want to read (here: 1D columns of doubles)
vec1d m, z, ra, dec;

// Then read them from the file (order is irrelevant)
fits::read_table("the_file.fits", ftable(m, z, ra, dec));

// Now you can do whatever you want with these columns
m += 1+z; // some silly stuff

// Writing is as simple
fits::write_table("new_file.fits", ftable(m, z, ra, dec));
```

Only the columns you need are actually read from the file, and the type of the variables you declare in C++ has to match the type of the columns that are found inside the FITS table (conversions will be performed automatically only if they imply no potential loss of data). Diagnostics will be given if this is not the case. See the *phy*$_{++}$ documentation for more detail.

In IDL. Column-oriented FITS tables can be read using `mrdfits`, and written using `mwrfits`:

```
; Read the whole table
tbl = mrdfits('the_file.fits', 1, /silent)

; Now you can do whatever you want with these columns
tbl.m += 1+tbl.z ; some silly stuff

; Writing is as simple
mwrfits, tbl, 'new_file.fits', /create
```

Be careful not to forget the `/create` keyword, else if the file already exists the function will add a new extension to the table to write the data, and this is not what you want.

In Python. Column-oriented FITS tables are not well supported by the standard FITS I/O module from `astropy` (you *can* use `astropy.io.fits`, but it will be a bit cumbersome). To cope with this situation, I have written a small module that implements this in a user-friendly way; you can download it [here]. Usage is as simple as above:

```
import pycolfits

# Read the whole table
tbl = pycolfits.readfrom('the_file.fits')

# Now you can do whatever you want with these columns
tbl['m'] += 1+tbl['z'] # some silly stuff

# Writing is as simple
pycolfits.writeto('new_file.fits', tbl, clobber=True)
```

As for the IDL version, do not forget the `clobber=True` argument, else the function will throw an exception if the file already exists (I don't like this, but this seems to be the expected behavior in Python).

In Topcat. If you use a language that does not support column-oriented tables, you can always try to use Topcat to open these tables and convert them into a format of your choosing (would it be ASCII or some other FITS format). Be sure to load the FITS table as "colfit-basic". Unfortunately it does not support all the features of column-oriented tables, so all but the simplest files will be rejected. Consider instead investing time learning one of the above languages (and maybe not IDL).

2.2 `egg-gencat`

In this section, I first describe the most basic options and parameters of the tool. Then, I give you some clues on how to read and use the generated catalog. All the other sub sections are here to describe more advanced features, or smaller details that you do not need to worry about for your first contact with the tool.

2.2.1 Basic usage

This is the main program of the EGG suite. It will create a new mock catalog from scratch. You simply have to describe the parameters of the survey you want to simulate, i.e., the area of the sky that is covered (in square degrees), the depth (in AB magnitude) and the selection band. This is the standard way of using the software:

```
egg-gencat area=0.08 maglim=28 selection_band=hst-f160w
```

This will create a new catalog in a FITS table called `egg-[yyyymmdd].fits`, where `[yyyymmdd]` is the current date. The name of this file can also be chosen using the command line parameter `out`:

```
egg-gencat area=0.08 maglim=28 selection_band=hst-f160w out=some_folder/mycatalog.fits
```

The program generates the fluxes of each galaxies in an arbitrary number of bands simultaneously. By default, a standard set of broad bands (mostly from the *Hubble*, *Spitzer* and *Herschel* telescopes) is chosen, but most likely you will want to change that. The list of photometric bands must be given as an array of band names through the `bands` command line argument:

```
egg-gencat area=0.08 maglim=28 selection_band=hst-f160w \
    bands=[vimos-u,hst-f160w,spitzer-irac1,herschel-pacs160]
```

Here we ask for 4 bands: `vimos-u`, `hst-f160w`, `spitzer-irac1` and `herschel-pacs160`, but you can have as many as you want. Note that the selection band does not need to be part of this list.

The names of the bands is usually of the form `[instrument]-[band]`. They are case-sensitive, so `vimos-u` is good, but `vimos-U` is not. Because these names are somewhat arbitrary, you can see the list of all available bands by calling:

```
egg-gencat list_bands
```

This will print an alphabetically sorted list of bands, giving you the name, the reference wavelength and the width of all the filters. You can also filter this list by providing a value to `list_bands`:

```
# print all the JWST bands
egg-gencat list_bands="jwst-"

# Result:
# List of available bands (filter: 'jwst-'):
#  - jwst-f070w  ref-lam = 0.695353 um, FWHM = 0.168 um
#  - jwst-f090w  ref-lam = 0.902688 um, FWHM = 0.208 um
#  - jwst-f115w  ref-lam = 1.15124 um,  FWHM = 0.271 um
#  - jwst-f150w  ref-lam = 1.50168 um,  FWHM = 0.337 um
#  - jwst-f200w  ref-lam = 1.99057 um,  FWHM = 0.471 um
#  - jwst-f277w  ref-lam = 2.78408 um,  FWHM = 0.729 um
#  - jwst-f356w  ref-lam = 3.55939 um,  FWHM = 0.831 um
#  - jwst-f444w  ref-lam = 4.44572 um,  FWHM = 1.1526 um


# print all the Ks bands
egg-gencat list_band="-Ks"

# Result:
# List of available bands (filter: '-Ks'):
#  - 2mass-Ks      ref-lam = 2.16848 um, FWHM = 0.271 um
#  - flamingos-Ks  ref-lam = 2.15594 um, FWHM = 0.308 um
#  - fourstar-Ks   ref-lam = 2.15584 um, FWHM = 0.322 um
#  - hawki-Ks      ref-lam = 2.14845 um, FWHM = 0.3241 um
#  - isaac-Ks      ref-lam = 2.16813 um, FWHM = 0.266 um
#  - moircs-Ks     ref-lam = 2.15952 um, FWHM = 0.273 um
#  - sofi-Ks       ref-lam = 2.16798 um, FWHM = 0.259 um
#  - vista-Ks      ref-lam = 2.15276 um, FWHM = 0.301 um
#  - wircam-Ks     ref-lam = 2.15923 um, FWHM = 0.322 um
# The value of 'list_band' is a POSIX regular expression.
# So you can do some smart filtering, like showing all
# the J, H and K (Ks) bands:
egg-gencat list_bands="-(J|H|K)"
```

2.2.2 Reading and using the generated catalog

The main objective of this tool is to produce fluxes. These fluxes are stored in the output catalog in FITS format as two-dimensional columns. The order of the dimensions depends on the language you use to browse the FITS table (see below), but one dimension corresponds to each galaxy, while the other corresponds to each simulated band. In the examples below, I assume that you want to read the fluxes in the *Hubble* F160W broad band (name: `"hst-f160w"`).

In C++.

```
// Declare the arrays that we need
vec2f flux;
vec1s bands;
// Read the data
fits::read_table("egg-20151201.fits", ftable(flux, bands));

// Find the band ID in this catalog
uint_t bid = where(bands == "hst-f160w")[0];

// Here is the flux
// Note: dimensions of "flux" is [galaxy]x[band]
vec1f f160w = flux(_,bid);

// Print the flux of galaxy 54312
print(f160w[54312]);
```

In IDL.

```
; Read the catalog in memory
cat = mrdfits('egg-20151201.fits', 1, /silent)

; Find the band ID in this catalog
bid = (where(strpos(cat.bands, 'hst-f160w') eq 0))[0]

; Here is the flux
; Note: dimensions of "cat.flux" is [band]x[galaxy]
f160w = cat.flux[bid,*]

; Print the flux of galaxy 54312
print, f160w[54312]
```

In Python.

```
import pycolfits

# Read the catalog in memory
cat = pycolfits.readfrom('egg-20151201.fits', lower_case=True)

# Find the band ID in this catalog
bid = np.where(cat['bands'] == 'hst-f160w')[0][0]
```

```
# Here is the flux
# Note: dimensions of "cat['flux']" is [galaxy]x[band]
f160w = cat['flux'][:,bid]

# Print the flux of galaxy 54312
print f160w[54312]
```

The simulated catalog contains many other columns to describe the physical properties of each galaxy. These include the sky position, the stellar mass, the redshift, the dust temperature, and so on. You may find some use in these parameters, for example if you want to see the redshift distribution that is predicted by the simulation for your next generation survey. Below is the list of such parameters and their definition.

- `id`: Unique identifier of this galaxy.
- `ra`, `dec`: The position of the galaxy on the sky, given in degrees.
- `z`, `d`: The redshift and luminosity distance of the galaxy. The distance is computed assuming a cosmology where $H_0 = 70$, $\omega_L = 0.7$, $\omega_m = 0.3$ and $k = 0$. It is given in Mpc.
- `m`, `m_bulge`, `m_disk`: The stellar mass of the galaxy, either the sum of all components, or the stellar mass of each individual component. It is given in base-10 logarithm and in units of $M_\odot$. Salpeter IMF.
- `passive`: The quiescent flag: 1 (or `true`) for quiescent galaxies, 0 (or `false`) for star-forming galaxies.
- `sfr`, `rsb`: Star formation rate (SFR) and "starburstiness" of the galaxy. The SFR is given in units of $M_\odot$/yr. Salpeter IMF. The starburstiness is the ratio between the SFR of the galaxy and the SFR of the Main Sequence at the redshift and the stellar mass of this galaxy. Starburstiness values above 1 indicate galaxies with excess SFR compared to the average.
- `bulge_angle`, `disk_angle`: Position angle of each stellar component on the sky, in degrees.
- `bulge_radius`, `disk_radius`: The angular size of each stellar component, in arcseconds. This is the half-light radius for both.
- `bulge_ratio`, `disk_ratio`: The axis ratio of each stellar component: 1 is perfectly round, 0 (which never happens) is perfectly linear.
- `bt`: The bulge-to-total stellar mass ratio, i.e., M_{bulge}/M_*.
- `rfuv_bulge`, `rfvj_bulge`, `rfuv_disk`, `rfvj_disk`: The $U - V$ and $V - J$ rest-frame colors of the bulge and disk components. Given in differences of AB magnitudes.
- `opt_sed_bulge`, `opt_sed_disk`: Index of the SED in the stellar template library that was chosen for each component. This is a flattened index, since the stellar library is provided on the two-dimensional grid of colors. To get the actual 2D index, just take the modulo and integer division. In C++, you would have `iuv = opt_sed/30` and `ivj = opt_sed%30`.
- `irx`, `sfrir`, `sfruv`: Contribution of the IR and UV light to the total SFR of the galaxy. The IRX is simply the ratio of the two.
- `lir`: Infrared luminosity from 8 to 1000 μm, in units of $L_\odot$.
- `tdust`: The average dust temperature, in Kelvins.

- mdust: The total dust mass, in $M_\odot$ (note: not in logarithm...).
- fpah: The fraction of the dust mass that is contributed by PAH molecules (0: no PAH, 1: only PAH).
- ir_sed: Index of the SED in the dust template library that was chosen for this galaxy. Corresponds to a given value of T_dust.
- bands, lambda: Arrays containing the names and reference wavelengths of each photometric band used to produce the observed fluxes. Wavelength is in μm.
- flux, flux_bulge, flux_disk: Two dimensional columns containing the total observed flux of each galaxy in each band, or the flux in each component. See above. Given in μJy.
- rfbands, rflambda: Arrays containing the names and reference wavelengths of each photometric band used to produce the absolute magnitudes. Wavelength is in μm.
- rfmag, rfmag_bulge, rfmag_disk: Two dimensional columns containing the total absolute magnitude of each galaxy in each band (given in rfbands), or the magnitude in each component. See below for more detail about absolute magnitudes. Given in AB magnitudes at 10 pc.
- zb: Redshift slices used by the program (two dimensions: one is for each slice, the second contains two elements: the lower and upper bounds of the slice).
- cmd: Single string containing the command line arguments that were used to create this catalog.

2.2.3 Choosing a seed

The catalog is created using a random number generator and a set of recipes. However, even with the random number generator, running the above commands will always produce the same catalog. This is because the same random *seed* is used every time. The seed basically sets the starting point of the random number generator: choosing a slightly different seed will result in a completely different catalog. By default, this seed is chosen to be equal to the arbitrary value of 42. You can change this value using the command line argument seed:

```
egg-gencat area=0.08 maglim=28 selection_band=hst-f160w seed=1456518
```

The value of the seed can be any positive integer number (including 0) that your computer can handle (that is, a 32bit or 64bit integer depending on the architecture of your CPU). Nobody knows what is the seed of our own Universe, so feel free to use whatever value you like.

2.2.4 Starting from your own galaxies

The first step of the program is to generate the galaxies, with their position on the sky, their redshift, their stellar mass, and their star-forming classification (i.e., each galaxy is either star-forming or quiescent according to the *UVJ* diagram). This is done using the stellar-mass functions that I observed in the GOODS–South field combined with observations in the Local Universe, and this should be fairly realistic between $z = 0$ to $z = 3$. However, the program also gives you the opportunity to provide these parameters yourself. You could, for example, provide a true (observed) catalog of galaxies, and let the program run its recipes to predict the fluxes of each object.

To do so, you must first create an input catalog suitable for ingestion by egg-gencat. The program accepts two file formats: either a column-oriented FITS table or a plain ASCII table. The ASCII file must contain 6 columns in the following order: the ID of the galaxy, the RA and Dec position in degrees (double precision is advisable), the redshift, the base-10 logarithm of the stellar mass, and the quiescent flag (1: quiescent, 0: star-forming). The FITS file must contain at least the columns "ra", "dec", "z", "m", and "passive" with the same content as for the ASCII table, and the column "id" is optional (if you don't provide it, egg-gencat will create an ID for you, starting from zero and increasing by one for each galaxy).

I assume you know how to handle ASCII tables yourself. For tables in the FITS format, since column-oriented tables are not very well known, I give below a small example in three common languages.

In C++.

```
// Create a small catalog with two galaxies!
vec1u id      = {1,        2};
vec1d ra      = {53.006,   53.008};
vec1d dec     = {124.0507, 124.051};
vec1f z       = {1.05,     2.6};
vec1f m       = {11.2,     10.5};
vec1b passive = {true,     false};

// Write the file
fits::write_table("input_cat.fits", ftable(id, ra, dec, z, m, passive));
```

In IDL.

```
; Create a small catalog with two galaxies!
cat = { $
    id      : [1,         2], $
    ra      : [53.006d,   53.008d], $
    dec     : [124.0507d, 124.051d], $
    z       : [1.05,      2.6], $
    m       : [11.2,      10.5], $
    passive : [1,         0] $
}

; Write the file
mwrfits(cat, 'input_cat.fits', /create)
```

In Python.

```
# Create a small catalog with two galaxies!
cat = {
    'id'      : np.array([1,        2]),
    'ra'      : np.array([53.006,   53.008]),
    'dec'     : np.array([124.0507, 124.051]),
    'z'       : np.array([1.05,     2.6]),
    'm'       : np.array([11.2,     10.5]),
    'passive' : np.array([True,     False])
}

# Write the file
pycolfits.writeto('input_cat.fits', cat, clobber=True)
```

2.2.5 Choosing the survey position

By default, the program creates a survey centered at the position RA=53.558750 and Dec=−27.176001, which is approximately the center of the GOODS–*South* field. You can specify another position if you wish, although for now this choice has no consequence on the simulated catalog itself. However, when stars are included in a future release of the program, the number density of foreground stars will likely depend on the survey position. On top of this, you may wish to perform some post-processing on the catalog to include additional components (cirrus emission, or whatever) that depend on the absolute sky position.

The center of the field can be specified using the `ra0` and `dec0` command line arguments:

```
egg-gencat area=0.08 maglim=28 selection_band=hst-f160w ra0=2.45621 dec0=33.065632
```

The current definition of the survey area is a bit naive, and may yield weird results for surveys very close to (or including) the poles. Try to avoid this. If you have to, at least double check that the results make sense.

2.2.6 Choosing the stellar mass range

The standard behavior of the program is to generate galaxies in redshift slices, going down in stellar mass until more than 90% of the galaxies become fainter than the limiting magnitude specified in `maglim`. An alternative is to use a fixed stellar mass range at all redshifts. To simulate flux-limited surveys, this is obviously less efficient, since you may miss visible galaxies at low redshifts, and/or include many unobservable galaxies at high redshifts.

Still, in case you need this feature, you can specify the stellar mass interval using the `mmin` and `mmax` arguments (note: as always, the stellar mass must be specified in base-10 logarithm). By default, `mmax=12`, and `mmin` has no value. Providing a value for `mmin` disables the limiting magnitude.

```
egg-gencat area=0.08 maglim=28 selection_band=hst-f160w mmin=8.5 mmax=11.5
```

2.2.7 Choosing the redshift range

A realistic simulation should include galaxies from all redshifts. However, for practical implementation reasons, this program can only handle a finite range, excluding $z = 0$ (the Milky Way, as far as I known) and $z = \infty$ (the Big Bang, or whatever). By default, the program generates galaxies starting from $z = 10.5$ up to $z = 0.05$. These boundaries are well suited for deep cosmological surveys like GOODS, but may be inadequate for shallower surveys covering wider areas. Changing the highest simulated redshift will have a negligible impact on performances, so I would advise leaving it as it is. On the other hand, if you simulate areas larger than 1 deg^2, a minimum redshift of 0.05 may be too high. Conversely, if you are interested in pencil-beam surveys like the HUDF, you may want to increase the minimum redshift to avoid bright foreground galaxies.

This can be achieved by modifying the `zmin` and `zmax` arguments:

```
egg-gencat area=0.08 maglim=28 selection_band=hst-f160w zmin=0.01 zmax=5.0
```

There is no implicit restriction regarding the minimum redshift, except that it must be different from zero. Note that, since the redshift bins are defined so that $z_{i+1} = z_i\,(1 + z_{\rm min})$, the smaller the minimum redshift, the more redshift bins will be simulated. So you may start to degrade the performances if you go too low.

2.2.8 Adding new photometric bands

Most of the commonly used filters are in this data base. But the list is not complete. You may want to use a filter that is not provided there, or even create your own imaginary filter to test some hypothesis, or predict what a future instrument will see. There are several ways to do this.

The easiest way is to define the filter in place, in the band list. For each filter in the `bands` list, you can use an extended syntax: `[band-name]:range:[lmin]:[lmax]`, where `[band-name]` is the name that you chose for your new filter (avoid spaces), and `[lmin]` and `[lmax]` define the minimum an maximum wavelength of the filter. The filter is then constructed on the fly, using a simple top-hat function between the specified boundaries. For example, lets improve the previous band list by adding a new hypothetical filter "herschel-pacs40" that spans $\lambda = 32$ to $51\,\mu$m:

```
egg-gencat area=0.08 maglim=28 selection_band=hst-f160w \
    bands=[vimos-u,hst-f160w,spitzer-irac1,herschel-pacs40:range:32:51]
```

This is rather crude though, and you may prefer to provide a more detailed filter response curve. To do so, use the syntax `[band-name]:file:[path]`, where this time `[path]` is the path to a file containing the filter response curve. This file can be either a column-oriented FITS table or an ASCII table. If it is an ASCII table, it must contain two columns: the wavelength (in μm), and the filter response curve, normalized to unit integral. If it is a FITS table, it must contain the columns "lam", the wavelength, and "res", the response curve. Note that, to be accurate, the filter *must* be defined so that the flux in the corresponding band is obtained by integrating as a function of wavelength the product of the filter f and the spectrum of the galaxy s_ν in flux per unit frequency. In other words:

$$S_\nu = \int_0^\infty d\lambda\, [f(\lambda)\; s_\nu(\lambda)]\,.$$

If you see yourself using the same filter over and over again, and you are tired of re-defining it in the command line arguments, you may add this filter to your local EGG data base. This data base is located in the `$INSTALL_ROOT_DIR/share/egg/filter-db` folder (see the installation instructions if you don't know what `$INSTALL_ROOT_DIR` is). The first step is to copy your filter response curve into this directory (you can use sub-directories if you wish). Either FITS or ASCII format will do. Then, open the `db.dat` file with your text editor, and add a new line somewhere in the file (the order of the lines is irrelevant):

```
[band-name]=[path]
```

The `[band-name]` is defined as above, and may not contain spaces. The `[path]` must be relative to the location of the `db.dat` file. Once this is done, you can use your filter like any other, by just specifying its name in the `bands` list.

2.2.9 Computing absolute magnitudes

In addition to observed fluxes, the program can also compute for you the absolute magnitude of each galaxy in a set of bands. This works exactly as for observed fluxes, except that you must specify the bands in the `rfbands` command line argument instead.

2.2.10 Disabling simulation steps

Sometimes you only care about one particular aspect of the simulation. To save time, you can disable the steps you do not need. In particular:

- `no_pos`: do not generate sky positions
- `no_clust`: do not put clustering in sky positions
- `no_flux`: do not generate any flux, just galaxy parameters
- `no_dust`: do not include dust emission in the fluxes
- `no_stellar`: do not include stellar emission in the fluxes

2.2.11 Saving the full spectrum of each galaxy

The output catalog produced by `egg-gencat` can only contain broad-band fluxes, not spectra. Indeed, saving all the spectra occupies a lot of disk space, and it is usually not necessary. However, this can be a useful piece of information for some specific applications, therefore this program gives you a way to obtain it.

To do so, just set the `save_sed` command line flag. This will save the bulge and disk spectra separately. Because this can generate huge volumes of data, the spectra are not saved inside the generated catalog, but in another file. By default, the name of this file is the same as that of the generated catalog, but ending with `"-seds.dat"`. You can change that using the `seds_file` command line argument.

```
egg-gencat area=0.08 maglim=28 selection_band=hst-f160w save_sed
```

The size of the file is typically about 40 KB per galaxy. So the command above, which generates some 100 000 galaxies, will generate about 4 GB of spectra. You have been warned.

The spectra in this file are stored in a custom binary format, since neither standard FITS nor ASCII tables provide an efficient way to handle this. If you do not want to bother figuring out how to read this file, please use the `egg-getsed` program like so:

```
# get the spectrum of the bulge of galaxy 54231
egg-getsed seds=catalog-seds.dat id=54213 component=bulge
# create the FITS file catalogs-seds-54213-bulge.fits
# wavelength is column 'lambda', spectrum is in 'flux'

# if you don't like FITS tables, you can also ask for ASCII
egg-getsed seds=catalog-seds.dat id=54213 component=bulge ascii
# create the ASCII table catalogs-seds-54213-bulge.cat
# two columns: wavelength and flux
```

Wavelength is given in μm, while the unit of the spectrum is a flux in μJy.

If you need more performance and/or do not want to create a spectrum file for each galaxy with the above command, you will need to read directly from the binary file. Each spectrum is stored contiguously, first the wavelengths (in μm), then the flux (in μJy). Both are saved in single precision floating point numbers. To figure out the position of the spectrum of a given galaxy, a "lookup" file is created. Its name is the same as that of the spectrum file, except that it ends with `"-seds-lookup.fits"`. This is a column-oriented FITS table. It contains five columns: `"id"` is the ID of each galaxy, `"bulge_start"` gives the position of the first element of the bulge spectrum of this galaxy in the file (in bytes, starting from the beginning of the file), `"bulge_nbyte"` gives the total number of bytes occupied by this spectrum (including the wavelengths and the spectrum values), and finally `"disk_start"` and `"disk_nbyte"` provide the same informations for the disk component.

So, to read the bulge spectrum of galaxy 54312:

- Open the spectrum file in binary mode.
- Go to the position `bulge_start[54312]`.
- Read `bulge_nbyte[54321]/sizeof(float)/2` floating point numbers into the wavelength array.
- Read another `bulge_nbyte[54321]/sizeof(float)/2` floating point numbers into the spectrum array.
- Close the file.

This is exactly what is done by `egg-getset`. Look at the source code of this tool if you are unsure of how to implement it yourself.

2.3 `egg-2skymaker`

2.3.1 Basic usage

This tool is used to convert a mock catalog created by `egg-gencat` into input catalogs and configuration files for *SkyMaker*. Each call to the tool creates one *SkyMaker* setup for one band, and uses a "template" *SkyMaker* configuration file to define the PSF and the noise properties of the image:

```
# assume we want to create an image of the Hubble F160W band
egg-2skymaker cat=egg-20151201.fits band=hst-f160w \
    template=goodss-hst-f160w.conf out=egg-f160w.cat verbose
```

The `cat` parameter holds the path to the mock catalog, `band` is the name of the photometric band for which to generate a *SkyMaker* setup, and `template` gives the name of the *SkyMaker* configuration template. The resulting *SkyMaker* input catalog will be saved in `out`. The *SkyMaker* configuration file and FITS header will be written in `[out]-sky.conf` and `[out]-hdr.txt` (with the extension of `out` removed). This parameter also defines the name of the image file that *SkyMaker* will create (see below).

The template file must be a valid *SkyMaker* configuration file. You should provide there all the necessary parameters to simulate an image of this type (i.e., the right PSF, the right noise level, etc.). In particular, take special care with the following parameters:

- `PIXEL_SIZE`: This parameter is needed by `egg-2skymaker` to compute the size of the FITS image that *SkyMaker* will generate. Must be in arcseconds/pixel.

- `PSF_xxx`: If you want to use a PSF file, make sure that the path is either a) an absolute path, or b) relative to the location of the template configuration file. In the latter case, `egg-2skymaker` will take care of converting this path to an absolute path. For example, if the template is located in `"/home/cschreib/test/goods-hst-f160w.conf"` and the value of `PSF_NAME` is `"../psfs/hst-f160w.fits"`, then it will be automatically converted to `"/home/cschreib/psfs/hst-f160w.conf"`. This is to ensure that *SkyMaker* is able to find the PSF file wherever the template configuration file is located.

You do not have to provide the following parameters, since they will be replaced automatically by `egg-2skymaker`:

- `IMAGE_TYPE`: Set to `SKY` (only filled if missing).

- `LISTCOORD_TYPE`: Set to `PIXELS`.

- `IMAGE_NAME`: Set to `[img_dir]/[out]-sci.fits`. The value of `img_dir` is read from the command line arguments, and defaults to the current directory. In this context, only the base name of `out` is used, i.e., the extension (`".cat"`) and the directories are removed. For example, if `out="catalogs/egg-f160w.cat"`, then the image name will be `"[img_dir]/egg-f160w-sci.fits"`.

- `IMAGE_SIZE`: Computed automatically from `PIXEL_SIZE` and the area covered by the mock catalog (see below).

- `IMAGE_HEADER`: Set to `[out]-hdr.txt`.

Out of the box, EGG provides you with a set of pre-defined template files. These serve two purposes: provide realistic templates for the most common cases, and provide examples that you can build upon to create your own templates. These pre-defined templates are stored in the `$INSTALL_ROOT_DIR/share/egg/skymaker-templates` folder (see the installation instructions if you don't know what `$INSTALL_ROOT_DIR` is). The program will automatically look inside this directory if the template file name that you provided in the `template` argument cannot be found in the current directory. Use the `list_templates` command to display the list of available pre-defined templates.

2.3.2 Defining the image dimensions

By default the program will try to build one single image out of the provided mock catalog. Knowing the requested pixel size and the coordinates of each galaxy, it draws a rectangle bounding box around the projected *x* and *y* coordinates of the galaxies, taking into account their respective size so that no galaxy ends up being truncated at the border of the image. If this is not what you want, there are several ways to adjust this behavior.

First, you can add a fixed extra amount of empty space at the border of the image using the `inset` keyword (in arcseconds). This adds up with the padding computed from the sizes of the galaxies. If you want to disable the size-dependent padding, set the `strict_clip` keyword (this will not disable the effect of `inset` if both arguments are used at the same time).

Second, creating a single image for the whole field might prove to be impractical if the pixel size is small and the area is large. The resulting image may be too big to handle conveniently with other programs such as DS9. In addition, *SkyMaker* can only create images with total size smaller than the amount of available RAM memory on your computer. If you encounter either of these limitations, you can ask `egg-2skymaker` to split the image into multiple tiles that *SkyMaker* will create one by one. To do so, use the `size_cap` keyword. The value must be the maximum allowed size of the image (or of a single tile) in GigaBytes. For example, if you set this value to 1 GB and if `egg-2skymaker` realizes that the resulting image would be larger than this, it will create a number of tiles, each with a size less than 1 GB and covering a different region of the field.

If multiple tiles are created, they are named according to their position. For example, if `IMAGE_NAME` was chosen to `"egg-f160w-sci.fits"`, then the tiles will be named `"egg-f160w-sci-x-y.fits"` where `"x"` and `"y"` are the indices of the tile (starting from 1) corresponding to increasing right-ascension and decreasing declination, respectively. One *SkyMaker* configuration file and FITS header will be created for each tile according to the same naming scheme.

In this situation, if one wants to know in which tile lies a particular galaxy, one should set the `save_pixpos` keyword. This will create an additional column-oriented FITS table whose name is given by `save_pixpos`. This table will contain the `x` and `y` position of each galaxy on their respective tile (given in pixel units, the first pixel of the map being `{1,1}`), and the columns `tilex` and `tiley` provide the indices of the tile.

2.3.3 Other features

By default the tool will create a *SkyMaker* catalog including all the galaxies from the mock catalog. If you wish, you can ask it to only output the galaxies brighter than a given magnitude in this band. This can be done with the `maglim` keyword, which must be set to the requested limiting magnitude in the AB system. This may improve the performances, at the expense of correctness. However, note that *SkyMaker* takes into account the "observability" of a galaxy when it paints it on the image: if the galaxy is substantially fainter than the noise level, *SkyMaker* will not draw it at all. So the performance gain of having fewer galaxies in the input list may not be noticeable.

2.4 `egg-gennoise` and `egg-genmap`

2.4.1 Basic usage

These two tools can be used to create the images corresponding to a given mock catalog created by `egg-gencat`. They provide a simple alternative to *SkyMaker*, with the added limitation that galaxies are all considered as point sources. This is usually fine for long wavelength images (*Spitzer* MIPS and *Herschel*, typically) in cosmological deep fields. In addition, more control is given relative to the noise properties.

The typical call sequence consists of two steps: creating the noise map with `egg-gennoise`, then painting the galaxies on top of it with `egg-genmap`. The standard way to use `egg-gennoise` is the following:

```
# assume we want to create an image of the Herschel PACS 160um band
egg-gennoise cat=egg-20151201.fits out=pacs160-noise.fits \
    psf=herschel-pacs160.fits aspix=2.4 rms=1.68e-5 verbose
```

The argument `cat` gives the path to the mock catalog, while `out` gives the name of the file into which the noise map will be saved. Then, `psf` provides the FITS image of the point spread function of this instrument, `aspix` is the pixel size (in arcseconds/pixel) and `rms` is the desired noise level of the map (in map units, whatever they are). This latter value would be the standard deviation of the map pixel values without any galaxy painted on it. If you want to mimic the noise level of an existing image, simply measure the standard deviation of the pixels in an empty region and use this value for the `rms` keyword. The noise is assumed to be Gaussian with zero average.

Some common PSFs are provided with EGG. They are stored in the folder (see the installation instructions if you don't know what is). If the program cannot find the PSF you provided in the current directory, it will each among these PSFs for a matching name. Use the `list_psfs` argument to display a list of all the available PSFs.

One the process is complete, this will create a new image containing just noise. The dimensions of the image are computed automatically from the provided pixel size and the area covered by the mock catalog so that no galaxy is truncated close to the border of the image. A padding of 5 times the full-width at half-maximum of the PSF is included to ensure that this is true. The resulting noise map contains the necessary WCS astrometry.

Once this noise map is created, we can use `egg-genmap` to add the galaxies:

```
egg-genmap cat=egg-20151201.fits out=pacs160-sci.fits \
    noise_map=pacs160-noise.fits band=herschel-pacs160 \
    psf=herschel-pacs160.fits flux_factor=1.613e6 verbose
```

The values of `cat` and `psf` must be the same as in the previous step. This time, `out` gives the name of the file into which the final image will be saved, while `noise_map` must be set to the name of the noise map created by `egg-gennoise`. Finally, `band` sets the photometric band from which the flux of each galaxy will be drawn, and `flux_factor` is the conversion factor from fluxes to map units.

The flux conversion factor is strongly tied to the properties of the PSF file. If the PSF is normalized to unit peak flux, then it is the conversion between μJy/beam to map units. Else, if the PSF is normalized to unit total flux (the sum of all pixels is equal to one), then it is the conversion between μJy/pixel and map units. Be careful that, in this latter case, the aperture correction resulting from a truncated PSF need to be included in the conversion factor. In all cases, a galaxy is placed on the map following:

$$\texttt{map[i,j]} \mathrel{+}= \texttt{flux\_factor} \times \texttt{psf[i,j]} \times (S_\nu/\mu\mathrm{Jy})$$

where `psf[i,j]` is the pixel value of the PSF translated to the position of the galaxy and S_ν is the flux of this galaxy.

2.4.2 Using a custom astrometry and image dimensions

The tool `egg-gennoise` can compute automatically a suitable astrometry and image dimensions according the input catalog. However, you also have the possibility to copy the astrometry and image properties of an existing file. To do so, just use the `astro` command line argument and make it point to the FITS image you want to copy the data from:

```
egg-gennoise cat=egg-20151201.fits out=pacs160-noise.fits \
    psf=herschel-pacs160.fits rms=1.68e-5 verbose \
    astro=this_image_I_like_a_lot.fits
```

In this case, the value of `aspix` is not used and it can be omitted from the argument list.

2.4.3 Flag regions of the map with poor catalog coverage

By default the tools create a large map that encompasses the whole input catalog, with some margin. This means that, close to the borders of the image, the source density is very low and the image statistics are not realistic. This can be an issue for some applications. To cope for this, the `egg-gennoise` option `clip_borders` will identify the regions of the map with low source density, and flag them with "not-a-number" pixels so that no value can be read from there. This assumes that the input catalog does not contain holes, and will only affect the borders of the image.

2.4.4 Generate beam-smoothed images (sub-millimeter)

It is common practice in sub-millimeter astronomy to filter the observed maps with the PSF (or beam) of the instrument. Because these images usually have low signal-to-noise ratios, this allows easier identification of detections by eye or with blind source extraction tools. Note however that this is sub-optimal for prior-based source extraction or stacking because the size of the effective PSF is multiplied by $\sqrt{2}$, which increases the confusion. In addition, the gain is signal-to-noise is null since these methods already apply a sort of beam-smearing internally.

Nevertheless, in order to be able to reproduce such kind of images, both `egg-gennoise` and `egg-genmap` provide a set of command line arguments. In all cases, `beam_smoothed` must be set *for both programs* to notify them of your intention. By default, the map will be smoothed using the PSF as kernel. If you prefer to use a custom kernel, use the `smooth_fwhm` to specify the full-width at half-maximum of the kernel (in pixels).

If this option is used, the behavior of the programs is the following:

- Create the noise map, *unfiltered* (without smoothing), and with the requested noise level.
- Copy this map and apply the beam smoothing. Measure the new noise standard deviation (it is indeed modified by the smoothing process).
- Adjust the noise level of the *unfiltered* map so that the noise level in the previous step matches the one requested by the user.
- Feed the *unfiltered* map to `egg-genmap`, add the galaxies.
- Finally perform the beam smoothing.

This means that the value of the `rms` argument must be set to the expected noise standard deviation of the final, smoothed map.

2.4.5 Generate error and coverage maps

Without extra effort, the `egg-gennoise` program can generate for you the error and coverage map corresponding to your noise map and input catalog. To do so, simply set the `make_err` and `make_cov` arguments; this will generate the error map as `"[out]-err.fits"` and the coverage map as `"[out]-cov.fits"` (1: covered, 0 or not-a-number: not covered).

These maps carry very little information since the noise generated by `egg-gennoise` is uniform and covers the whole map by default. However, some external tools require the existence of either one or both of these.

2.4.6 Using a custom noise map

The whole point of separating the map-making process into two steps is that one can be used without the other. Indeed, `egg-genmap` can work with noise map, provided that it contains valid WCS astrometry. The tool can handle sources outside of the map, so it is fine if the provided noise map does not overlap perfectly with the input catalog. Just make sure that the PSF you provide is tabulated on the same pixel scale as your noise map. If you use the `beam_smoothed` option, note that the noise map must be provided *unfiltered* (i.e., before beam convolution).

For example, realistic *Herschel* noise maps can be obtained by jackknifing the observed data, and provide noise statistics (correlation, amplitude, etc.) that match very well the real images. Such kind of map can be fed naturally to `egg-genmap` through the `noise_map` argument.

2.4.7 Using a custom flux catalog

Similarly, both tools do not require that the input catalog be created by `egg-gencat`. They only need a limited set of columns: `"ra"` and `"dec"` to know the position of each galaxy on the sky, and `"flux"` and `"bands"` to know their flux. The format of these columns must be the same as that created by `egg-gencat`, but this can be easily achieved in any language. Alternatively, the program also supports a `"flux"` column containing a single band (i.e., a 1D vector column), in which case the `"bands"` column and the `band` command line argument are useless and can be omitted.