Generating mock catalogs with EGG

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 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 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 community. 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. 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
```

Contrary to the C++ version, these functions can only read the whole table at once, not specific columns. This can be convenient, but also inefficient. Be careful not to forget the `/create` keyword, else the function will create a new extension to the FITS file if it already exists.

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). However I have written a small module that implements this, 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 looking at the rest of the `astropy` library it seems to be the expected behavior).

In Topcat. If you use a language that does not support column-oriented tables, you can always 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`

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
```

2.2.2 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.

2.2.3 Providing 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 FITS table or a plain ASCII table. The FITS table must be a column-oriented FITS table.

2.3 `egg-getsed`

2.4 `egg-2skymaker`

2.5 `egg-gennoise`

2.6 `egg-genmap`