Clusters Documentation

Release 0.1.1

N. Chotard

Feb 21, 2018

Contents

Warning: Package under development

CHAPTER 1

General

1.1 Clusters

Python package wrapping up the ongoing cluster analysis of the LSST/DESC cluster group. For more info, see the two following github repositories:

- A collection of notebooks for LSST
- The ReprocessingTaskForce repository

See also the private Trello board that we use to share our work.

1.2 Installation

To install:

```
git clone https://github.com/nicolaschotard/Clusters.git
pip install Clusters/
```

To install in a local directory `mypath`, use:

```
pip install --prefix='mypath' Clusters/
```

and do not forget to add it to your PYTHONPATH.

To upgrade to a new version (after a `git pull` or a local modification), use:

```
pip install --upgrade (--prefix='mypath') Clusters/
```

To install a release version (no release version available yet):

```
pip install http://github.com/nicolaschotard/Cluster/archive/v0.1.tar.gz
```

Also works with the master:

```
pip install (--upgrade) https://github.com/nicolaschotard/Clusters/archive/master.zip
```

In the future, release versions will be listed at this location.

Package developers will want to run:

```
python setup.py develop
```

1.3 Dependencies

`Clusters` has for now the following dependencies (see the quick installs below):

- Python 2.7 and libraries listed in the requirements file
- The LSST DM stack.

Photometric redshift estimators:

- LEPHARE
- BPZ

1.3.1 Python

To install the python dependencies, simply do:

```
pip install -r requirements.txt
```

1.3.2 DM stack quick install

This four-step procedure should allow you to install and configure a light version of the DM stack, but complete enough to use the `Clusters` package. It should take ~10 minutes.

- Get and install miniconda, if you do not have it already:

```
wget https://repo.continuum.io/miniconda/Miniconda2-latest-Linux-x86_64.sh -O␣
↪miniconda.sh
bash miniconda.sh -b -p $HOME/miniconda
export PATH="$HOME/miniconda/bin:$PATH"
conda config --set always_yes yes --set changeps1 no
conda update -q conda
```

- Install the needed part of the DM stack (we do not need the entire stack):

```
conda config --add channels http://conda.lsst.codes/stack/0.13.0
conda create -q -n lsst python=2.7
source activate lsst
conda install -q gcc lsst-daf-persistence lsst-log lsst-afw lsst-skypix lsst-meas-
↪algorithms lsst-pipe-tasks lsst-obs-cfht
```

- To use this install of the DM stack, do not forget these following setups:

```
export PATH="$HOME/miniconda/bin:$PATH"
source activate lsst
source eups-setups.sh
setup daf_persistence
setup afw
setup obs_cfht
```

If these steps went well, you should be able to use `clusters_data.py` on one of the outputs of the DM stack (see below to get some data).

1.3.3 LEPHARE quick install

You can download and install a pre-configured version of LEPHARE as followed:

- for linux system:

```
wget https://lapp-owncloud.in2p3.fr/index.php/s/MDaXObLSD9IVQ1B/download -O␣
↪lephare.tar.gz
tar zxf lephare.tar.gz
```

- for mac:

```
wget https://lapp-owncloud.in2p3.fr/index.php/s/bMTLiwfGK1SpOqE/download -O␣
↪lephare.tar.gz
tar zxf lephare.tar.gz
```

When the download is complete, exctract the `lephare` directory where it suits you (`mypath` in this example), and set the following environment variables (use setenv if needed):

```
export LEPHAREWORK="mypath/lephare/lephare_work"
export LEPHAREDIR="mypath/lephare/lephare_dev"
export PATH="$PATH:mypath/lephare/lephare_dev/source"
```

You should now be able to run `clusters_zphot.py` (only tested on linux systems).

1.3.4 BPZ quick install

The following steps can be copied/pasted in order to install and test BPZ quickly. It supposes that LEPHARE has been installed following the procedure shown in the previous section (you need `$LEPHAREDIR/filt/cfht/megacam/\*.pb`). Here are the official install instructions for BPZ.

Get BPZ:

```
export MYDIR="an install dir" # change that line
cd MYDIR
wget http://www.stsci.edu/~dcoe/BPZ/bpz-1.99.3.tar.gz
tar -xvf bpz-1.99.3.tar.gz
```

Create needed enironment vairables:

```
export BPZPATH="$MYDIR/bpz-1.99.3"
export PYTHONPATH=$PYTHONPATH:$BPZPATH
export NUMERIX=numpy
```

Create the filter files using the LEPHARE install:

```
cd $BPZPATH/FILTER/
cp $LEPHAREDIR/filt/cfht/megacam/*.pb .
for f in *.pb; do mv "$f" "CFHT_megacam_${f%.pb}.res"; done
```

Test the install and the megacam filter:

```
wget https://lapp-owncloud.in2p3.fr/index.php/s/FP1vSMB7emLxwwg/download -O megacam_
↪bpz.columns
wget https://lapp-owncloud.in2p3.fr/index.php/s/HZbzCFLoy8Lcmwx/download -O megacam_
↪bpz.in
python $BPZPATH/bpz.py megacam_bpz.in -INTERP 2
```

1.4 Configuration file

All the scripts will take the same input YAML file, which contains necessary informations for the analysis or simply for plotting purpose, such as the name of the studied cluster. Keys are listed below and are case-sensitive. Additional keys are simply ignored. You can find examples of these configuration files in the config directory, or clicking here for MACSJ2243.3-0935.

General keys	Type	Description [units]
`"cluster"`	string	Name of the cluster
`"ra"`	float	RA coordinate of the cluster **[deg]**
`"dec"`	float	DEC coordinate of the cluster **[deg]**
`"redshift"`	float	Cluster redshift
`"butler"`	string	Absolute path to the intput data (butler)
`"filter"`	list	List of filters to be considered, e.g., 'ugriz' (Megacam filters)
`"patch"`	list	List of patches to study

The following list of optional keys can also be added to the configuration file. They correspond to specific configurations of the different steps of the analysis. While the previous list will most likely stay unchanged, the following one will be completed with new keys as this analysis will progress.

Optional keys	Type	Description [units]
`"keys"`	dict	Dictionary containing list of keys for the catalogs (see below)
`"zphot"`	dict	Dictionary containing a list dictionnaries whose names identify the photoz run configuration (code, zpara, etc.)
`"code"`	string	Name of the photoz code to run: "lephare" (default) or "bpz"
`"zpara"`	string	Paths to the photoz code parameter file (see below)
`"zspectro_file"`	string	File containing spectroz sample for LePhare training
`"mass"`	dict	Dictionary specifying options to run the mass code

- `keys` is a dictionary having the name of the different catalogs like **deepCoadd_meas**, **deepCoadd_forced_src** and **forced_src**. The list of keys for a given catalog can include:
  - "the_full_name_of_a_key";
  - "*_a_part_of_a_key_name" or "an_other_part_of_a_key_name*" preceded or followed by a *;
  - a combination of all the above: ["key1", "ke*", "*ey"];
  - or a "*" to get all keys available in a catalog, which is the default value for all catalogs.
- `zphot` is a dictionary whose keys are user-defined names to identify a given zphot configuration. These names will be used to identify each photoz output in the final astropy table. Each configuration is itself a dictionary

with optional keys (`code`, `zpara` and `zspectro_file`). If `zphot` is not specified the code will run using LePhare and a default parameter file. At the moment `"code":"lephare"` and `"code":"bpz"` are supported. More photoz code options might be added in the future.

- `mass` is a dictionary intended for user-defined options to run the mass code. At the moment, the only possible key is `zconfig` whose argument should be one of the keys of the `zphot` dictionary.

1.5 General usage

`Clusters` consists in several command-line executables that you have to run in the right order.

- Get the input data and dump them in a hdf5 file containing astropy tables (see the data format section of the documentation for detail):

```
clusters_data.py config.yaml (--output data.hdf5)
```

The memory you will need to load the data from the butler will for now depend on the number of catalogs (e.g. the `forced_src` catalog), patch, visits and CCD you will be loading. For instance, if you try to load ~10 patches for 5 filters, and want all the keys of several catalogs including the `forced_src` one (CCD-based), you could need up to 16GB of memory. The **best practice** would thus be to first check the list of existing keys of the catalogs you want to load (`--show` option), fill the configuration file with your selected list of keys using the `keys` parameter for each catalog, and finally run `clusters_data.py` using this configuration file. You can find an example for such cofiguration file there and some detail on how to use the keys in the previous section. This will allow you to adapt the content of the output file and work with lighter data files.

- Data validation plots can for now be found in the several notebooks available in:

```
https://github.com/nicolaschotard/Clusters/tree/master/notebooks
```

Once the main catalogue has been written in `data.hdf5` by `clusters_data.py`, the remaning steps of the pipeline may all be run using the same command line format:

```
clusters_xxx.py config.yaml data.hdf5
```

By default, the outputs of each step (extinction, photoz, galaxy selection) are stored as additional paths in `data.hdf5`. More details are given below.

- Correct the data for Milky Way extinction:

```
clusters_extinction.py config.yaml data.hdf5 (--output extinction.hdf5)
```

will save the extinction correction into path `extinction` of `data.hdf5` (if –output not specified) or `extinction.hdf5` (if specified).

- Get the photometric redshift using LEPHARE:

```
clusters_zphot.py config.yaml data.hdf5 (--extinction --dustmap sfd) (--output␣
↪zphot.hdf5)
```

This loops over the user-defined zphot configuration keys given under `zphot` in the `config.yaml` file. The results of each photoz run (point estimate and pdz distribution) is stored in `data.hdf5` (or `zphot.hdf5` if a different output is required) in a path whose name corresponds to the user-defined zphot configuration keys.

The `--extinction` option corrects the magnitudes according to what was previously computed by `clusters_extinction`, before running the photoz. You can select the dust map using the `--dustmap` option, which must have also been added in the previous step.

- Flag galaxies to be removed for the lensing analysis:

```
clusters_getbackground.py config.yaml data.hdf5 (--zdata zdata.hdf5) (--zmin z_
↪min)
                          (--zmax z_max) (--thresh_prob threshold) (--rs)
```

  will produce redshift-based flag for the selection of background galaxies.

  Each zphot user-defined configuration yields a new `flag_zphot_config_name` path in `data.hdf5` containing two columns:

  - one `flag_z_hard` corresponding to a hard redshift cut: all galaxies in [`z_min`, `z_max`] are flagged. Default is [0,z_cluster+0.1];
  - one `flag_z_pdz` corresponding to a pdz-based cut: if the probability of a galaxy to be located at z < z_cluster + 0.1 is larger than `thresh_prob` [%], the galaxy is flagged to be removed. Default is 1%.

  Galaxies belonging to the cluster red sequence may also be flagged using the `--rs` option. However, this option is not entirely reliable yet.

  Flags are set to `True` when the galaxy has passed the cut (i.e. is the be kept for analysis).

- Compute the shear:

```
clusters_shear config.yaml input.hdf5 output.hdf5
```

- A pipeline script which run all the above step in a raw with standard options:

```
clusters_pipeline config.yaml
```

With any command, you can run with `-h` or `--help` to see all the optional arguments, e.g., `clusters_data.py -h`.

1.6 Test the code

If you have installed all the dependencies previoulsy mentionned, download the following test data set:

```
wget https://lapp-owncloud.in2p3.fr/index.php/s/xG2AoS2jggbmP0k/download -O testdata.
↪tar.gz
tar zxf testdata.tar.gz
```

The `testdata` directory contains a subset of the reprocessing data available for MACSJ2243.3-0935. It can be used as a test set of the code, but is not complete enough to run the full analysis. Here is the full structure and content of this directory, which has the exact same structure as a regulare DM stack output directory:

```
testdata/
├── input
│   ├── _mapper
│   └── registry.sqlite3
├── output
│   ├── coadd_dir
│   │   ├── deepCoadd
│   │   │   ├── g
│   │   │   │   └── 0
│   │   │   │       ├── 1,5
│   │   │   │       └── 1,5.fits
│   │   │   └── skyMap.pickle
```

(continues on next page)

(continued from previous page)

```
    │   ├── deepCoadd-results
    │   │   └── g
    │   │       └── 0
    │   │           └── 1,5
    │   │               ├── bkgd-g-0-1,5.fits
    │   │               ├── calexp-g-0-1,5.fits
    │   │               ├── detectMD-g-0-1,5.boost
    │   │               ├── det-g-0-1,5.fits
    │   │               ├── forced_src-g-0-1,5.fits
    │   │               ├── meas-g-0-1,5.fits
    │   │               ├── measMD-g-0-1,5.boost
    │   │               └── srcMatch-g-0-1,5.fits
    │   ├── forced
    │   │   └── 08BO01
    │   │       └── SCL-2241_P1
    │   │           └── 2008-09-03
    │   │               └── g
    │   │                   └── 0
    │   │                       ├── FORCEDSRC-1022175-00.fits
    │   │                       ├── FORCEDSRC-1022175-09.fits
    │   │                       ├── FORCEDSRC-1022176-00.fits
    │   │                       ├── FORCEDSRC-1022176-09.fits
    │   │                       ├── FORCEDSRC-1022177-00.fits
    │   │                       ├── FORCEDSRC-1022177-09.fits
    │   │                       ├── FORCEDSRC-1022178-00.fits
    │   │                       ├── FORCEDSRC-1022178-09.fits
    │   │                       ├── FORCEDSRC-1022179-00.fits
    │   │                       ├── FORCEDSRC-1022179-09.fits
    │   │                       ├── FORCEDSRC-1022180-00.fits
    │   │                       └── FORCEDSRC-1022180-09.fits
    │   └── _parent -> ../
    └── _parent -> ../input/
└── travis_test.yaml
```

With this data set, you should be able to test most of the `Clusters` parts. You can start with the test suite available in the tests directory. To do so, use:

```
python setup.py test
```

It will use the testdata that you have downloaded previoulsy and run the tests. This is also usefull if your goal is to add new tests.

1.7 Get the data

1.7.1 Raw DM stack outputs

If you have installed `Clusters` but do not have any data to run it on, you can use one of our re-processing outputs for MACSJ2243.3-0935. The corresponding configuration file is stored under configs/. To use it, you either need to be connected at CC-IN2P3, or change the path to the butler inside the config file (if you already have a copy of this data). You could also mount sps on your personal computer (see this how to).

1.7.2 `clusters_data.py` output

The first step of the `Clusters` package is `clusters_data.py`, which will get the data from the DM butler, convert them into `astropy` tables and save them in a single `hdf5` file. To do so, you need the LSST DM stack to be installed. If you want to skip this part and try the code whithout having to install the DM stack, you could also use the outputs of this first step that you can download from this repository, which contains the following files:

```
|-- CL0016
|   |-- [4.4G]  CL0016_data.hdf5                      # full data set
|   |-- [334M]  CL0016_filtered_data.hdf5             # only quality-filtered galaxies
|   `-- [ 312]  CL0016.yaml                           # configuration file
|-- MACSJ224330935
|   |-- [5.6G]  MACSJ2243.3-0935_data.hdf5            # full data set
|   |-- [367M]  MACSJ2243.3-0935_filtered_data.hdf5   # only quality-filtered galaxies
|   |-- [ 329]  MACSJ2243.3-0935.yaml                 # configuration file
```

This short tutorial explains how to use these `hdf5` files to start an analysis.

1.8 Tests

All the tests are being run in a docker container containing a light install of the stack along with all the needed data and softs.

In order to build the container, you will need docker to be installed. If so, you can run:

```
./build_docker_image.sh
```

The containers are for now stored on the following depot:

```
https://hub.docker.com/r/nchotard/clusters-test/tags/
```

To push a new container, do:

```
docker ps  # to get the container ID that you want to save and push
docker login "docker.io" -u nchotard  # need the password
docker commit THEID docker.io/nchotard/clusters-test:NAME  # e.g, NAME = 'centos7-
↪stackv13'
docker push docker.io/nchotard/clusters-test:NAME
```

To use it, simply do:

```
docker run -itd --name clusterstest docker.io/nchotard/clusters-test:NAME
docker attach clusterstest
```

Use CTRL-P CTRL-Q to quit without stopping it, or CTRL-C to quit and stop.

CHAPTER 2

Data format

2.1 Overview

The data format used in all scripts is based on the Astropy Table format. In the *Build the table* section, we show how these Astropy Tables are created from the DM butler. This work is automatically done for the `deepCoadd_meas`, `deepCoadd_forced_src`, and `forced_src` catalogs, when avalaible, and then saved together in one single hdf5 file. The procedure to write and read these files and work with the loaded tables are described in the *Working with the table* section.

2.2 Build the table

The main table is built from the LSST DM butler as shown in the diagram[1] below:

For each filter `f`, an Astropy Table is created for all available patches `p1`, `p2`, etc. Since we have the same number of patches for all filters, which contain the exact same number of sources, all table (1,p), (2,p), etc., created from a patch will be of the same size for all filter. The Astropy Tables created from all individual filters/patches set will then be vertically stacked. This means that if we respectively have `N1`, `N2` and `N3` sources for the patches 1, 2 and 3, the output table will contains `N1` + `N2` + `N3` sources. After the first stack, we end up with one table per filter, all containing the same number of sources. These per-filter tables are finally stacked together to form the final table shown on the right-hand side of the diagram.

2.3 WCS

The WCS computed during the data processing is also stored in the `hdf5` file in the `wcs` path. If you load the data using the read_hdf5 function, the output will be a dictionary containing a `wcs` key, which refers to an astropy.wcs.WCS

[1] Diagram created using https://www.jgraph.com/. Use the https://www.draw.io/?demo=1 application and the last xml file from this repository to update the diagram if needed.

object. The skycoord_to_pixel and pixel_to_skycoord functions take this `wcs` object to convert the input coordinates into an output format (sky <-> pixel).

All the tables (correspondig to the catalogs listed previously) already contain three coordinates columns:

- `coord_ra` and `coord_dec`: they are the (ra, dec) coordinates in radian;
- `coord_ra_deg` and `coord_dec`: they are the (ra, dec) coordinates in degree;
- `x_Src` and `y_Src`: they are the (x, y) position in pixel.

2.4 Working with the table

Note: The corresponding Jupyter notebook can be found here. You can also reproduce these results in ipython.

`Astropy` tables are great to work with, and can be used for all kind of analysis in the context of our cluster study. You can apply filters, group by column, concatenate them, etc. For a detailed review on Astropy tables, see there.

2.4.1 Load the table

The `Astropy` tables created by the `clusters_data` step are saved in an `hdf5` file, and contains two main tables, corresponding to two output catalogs of the data processing using the DM stack. As an example, we will use here the `deepCoadd_forced_src` catalog, corresponding to the forced photometry processing (some details).

If you want to start an analysis with an existing `hdf5` file containing catalogs, you can use the one we have created for MACSJ2243.3-0935, which is saved at CC-IN2P3 under:

```
/sps/lsst/data/clusters/MACSJ2243.3-0935/analysis/output_v1/MACSJ2243.3-0935_data.hdf5
```

To load the `deepCoadd_forced_src` catalog, do:

```
from Clusters import data
f = "/sps/lsst/data/clusters/MACSJ2243.3-0935/analysis/output_v1/MACSJ2243.3-0935_
↪data.hdf5"
d = data.read_hdf5(f)
fc = d['deepCoadd_forced_src']
```

`d` is a dictionnary containing the 'deepCoadd_forced_src', the 'deepCoadd_meas' catalogs and the 'wcs' object.

```
print d.keys()
```

```
['deepCoadd_forced_src', 'wcs', 'deepCoadd_meas']
```

and `fc` is an astropy table

```
print fc
```

```
base_CircularApertureFlux_3_0_flag_sincCoeffsTruncated ... coord_dec_deg
                                                       ...      deg
------------------------------------------------------ ... --------------
                                                 False ... -9.50417299504
                                                  True ... -9.50631091083
                                                  True ... -9.50631273401
```

(continues on next page)

(continued from previous page)

```
                                                    True ... -9.50632589495
                                                    True ...  -9.5063327395
                                                   False ...  -9.5062460577
                                                    True ... -9.50629874096
                                                    True ... -9.50635437897
                                                   False ... -9.50600120865
                                                   False ... -9.50549567214
                                                     ... ...              ...
                                                   False ... -9.73333093082
                                                   False ... -9.73388006895
                                                   False ...  -9.7302761071
                                                   False ... -9.73010079525
                                                   False ... -9.72701283749
                                                   False ...  -9.7273114286
                                                   False ... -9.91085559972
                                                   False ... -9.91084514606
                                                   False ...  -9.8851539436
                                                   False ... -9.88578472829
Length = 1050500 rows
```

As you can see, there are

```
N = len(fc)
print N, "rows"
```

```
1050500 rows
```

in this table. This number correspond to the number of sources (ns) times the number of filters (nf): N = ns x nf. In this table, we have the following number of filter:

```
filters = set(fc['filter'])
nf = len(filters)
print nf, "filters:", filters
```

```
5 filters: set(['i', 'r', 'u', 'z', 'g'])
```

The number of sources in this catalog if thus:

```
ns = N / nf
print ns, "sources"
```

```
210100 sources
```

The number of columns corresponding to the number of keys available in the catalog is:

```
print "%i columns" % len(fc.keys())
for k in sorted(fc.keys())[:10]:
    print k
```

```
207 columns
base_CircularApertureFlux_12_0_flag
base_CircularApertureFlux_12_0_flag_apertureTruncated
base_CircularApertureFlux_12_0_flux
base_CircularApertureFlux_12_0_fluxSigma
base_CircularApertureFlux_12_0_mag
```

(continues on next page)

(continued from previous page)

```
base_CircularApertureFlux_12_0_magSigma
base_CircularApertureFlux_17_0_flag
base_CircularApertureFlux_17_0_flag_apertureTruncated
base_CircularApertureFlux_17_0_flux
base_CircularApertureFlux_17_0_fluxSigma
```

2.4.2 Apply filters

You can filter this table to, for example, only keep the `i` and `r` magnitude of the `modelfit_CModel_mag` for all sources:

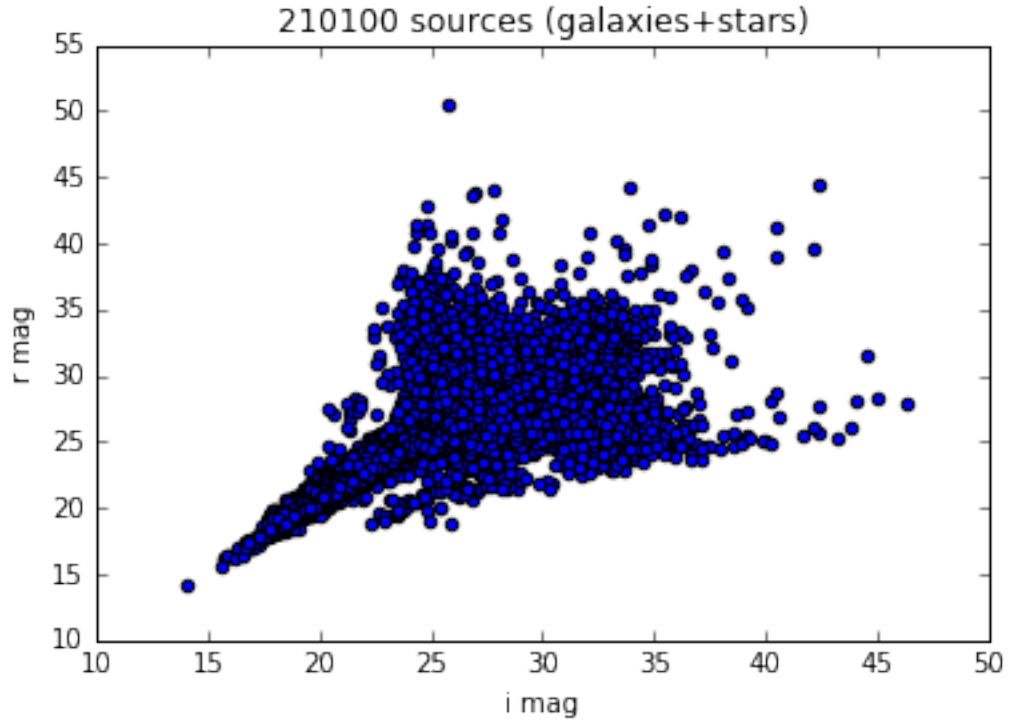
```
magi = fc['modelfit_CModel_mag'][fc['filter'] == 'i']
magr = fc['modelfit_CModel_mag'][fc['filter'] == 'r']
```

and plot them against each other

```
# ignore the following line
%matplotlib inline
```

```
import pylab
pylab.scatter(magi, magr)
pylab.xlabel('i mag')
pylab.ylabel('r mag')
pylab.title('%i sources (galaxies+stars)' % len(magi))
```

```
<matplotlib.text.Text at 0x7f55453994d0>
```



A few standard filters have been implemented in `data` and can be used directly to get a clean sample of galaxies:

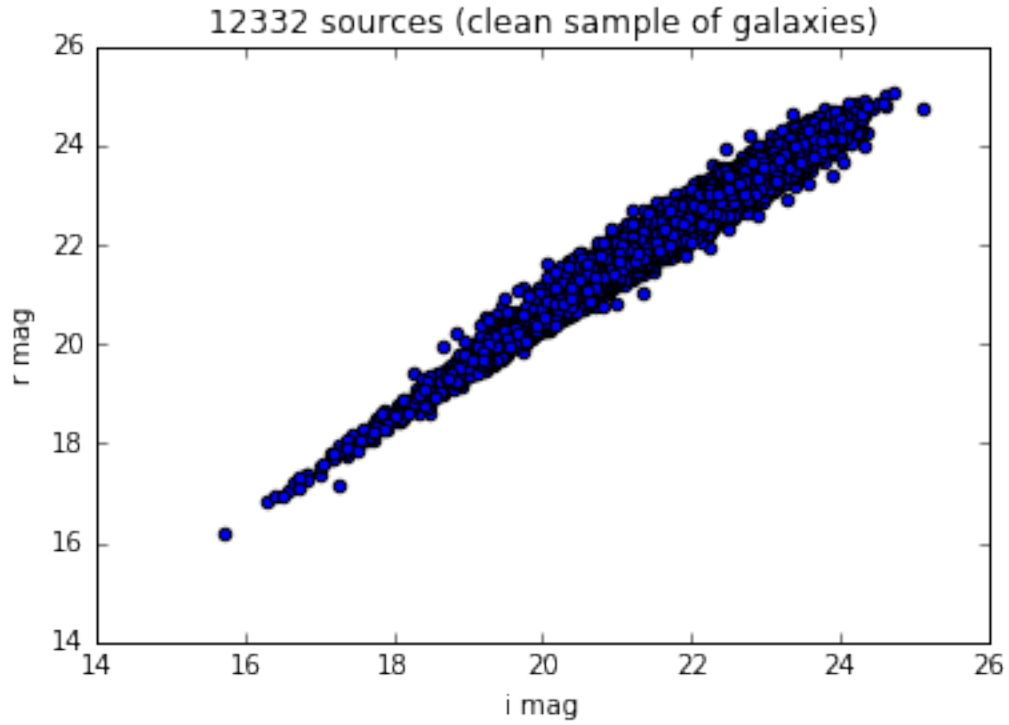
```
# ignore the following line
import warnings; warnings.filterwarnings("ignore")
```

```
data_filtered = data.filter_table(d)
fc_filtered = data_filtered['deepCoadd_forced_src']
```

The same plot as in the above example now looks like

```
magi_filtered = fc_filtered['modelfit_CModel_mag'][fc_filtered['filter'] == 'i']
magr_filtered = fc_filtered['modelfit_CModel_mag'][fc_filtered['filter'] == 'r']
pylab.scatter(magi_filtered, magr_filtered)
pylab.xlabel('i mag')
pylab.ylabel('r mag')
pylab.title('%i sources (clean sample of galaxies)' % len(magi_filtered))
```

```
<matplotlib.text.Text at 0x7f55451f92d0>
```



See the code for a few other examples on how to use filters.

2.4.3 Add a new column

You can also add a new column to the table (examples here)

```
from astropy.table import Column
```

Create a simple shifted magnitude array

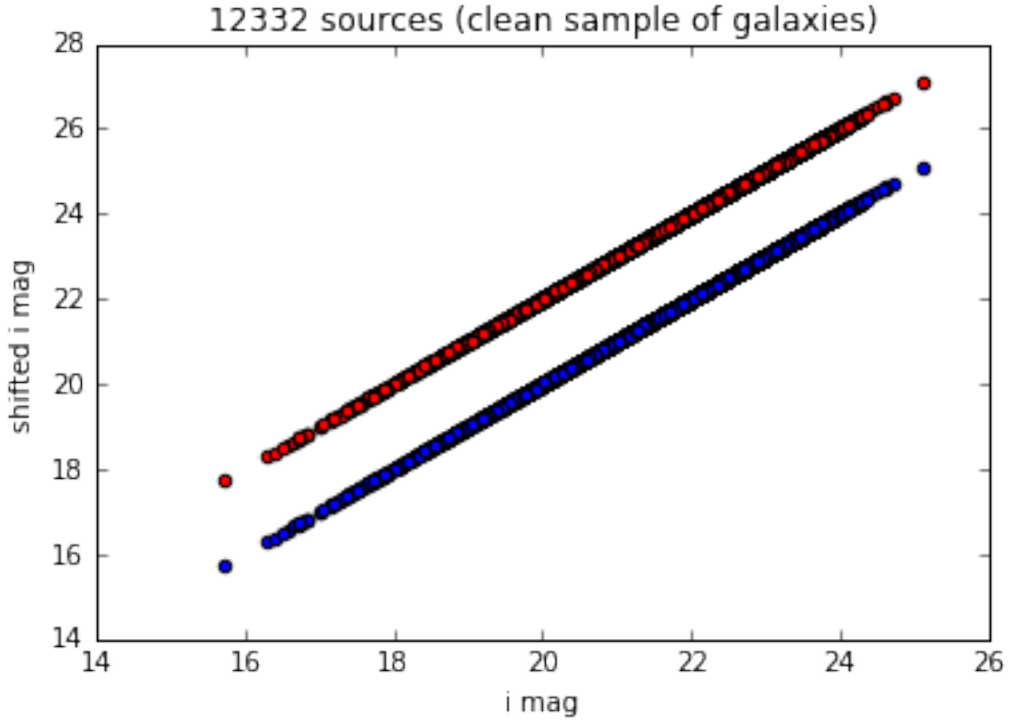
```
shifted_mags = fc_filtered['modelfit_CModel_mag'] + 2
```

Add it to the initial table and plot it against the initial magnitude (for the *i* filter here)

```
fc_filtered.add_column(Column(name='shifted_mag', data=shifted_mags,
                              description='Shifted magnitude', unit='mag'))
```

```
magi_filtered = fc_filtered['modelfit_CModel_mag'][fc_filtered['filter'] == 'i']
magi_shifted =  fc_filtered['shifted_mag'][fc_filtered['filter'] == 'i']
pylab.scatter(magi_filtered, magi_filtered)
pylab.scatter(magi_filtered, magi_shifted, c='r')
pylab.xlabel('i mag')
pylab.ylabel('shifted i mag')
pylab.title('%i sources (clean sample of galaxies)' % len(magi_filtered))
```

```
<matplotlib.text.Text at 0x7f55449552d0>
```

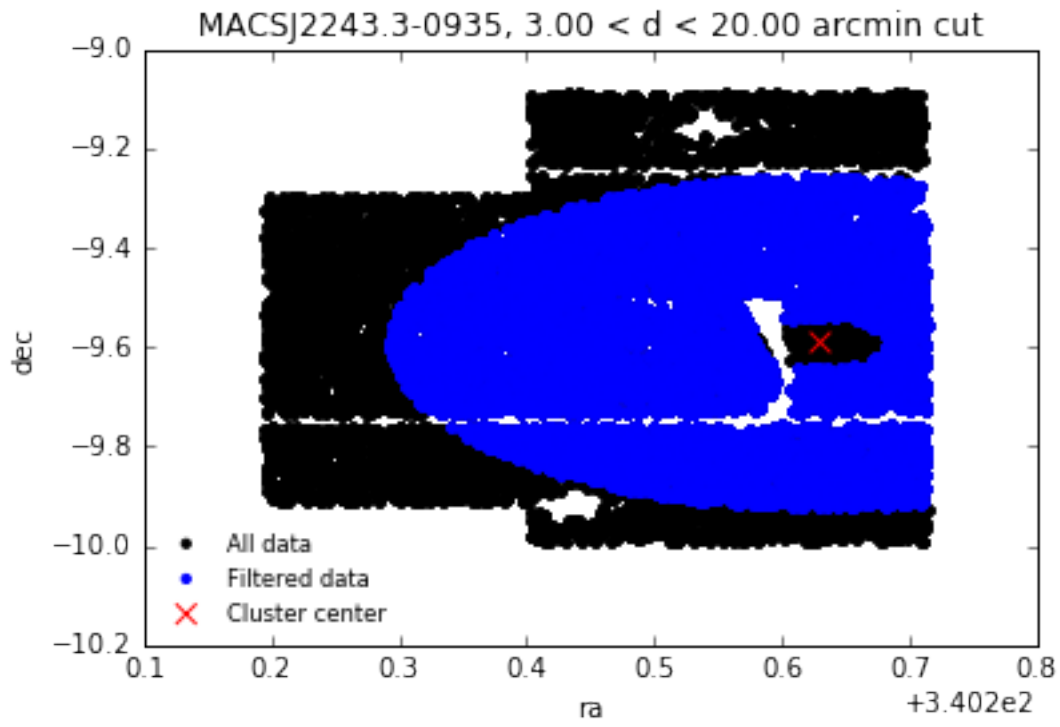


You can also add several columns using `fc.add_columns([Columns(...), Columns(...), etc])`.

2.4.4 Filter around the cluster center

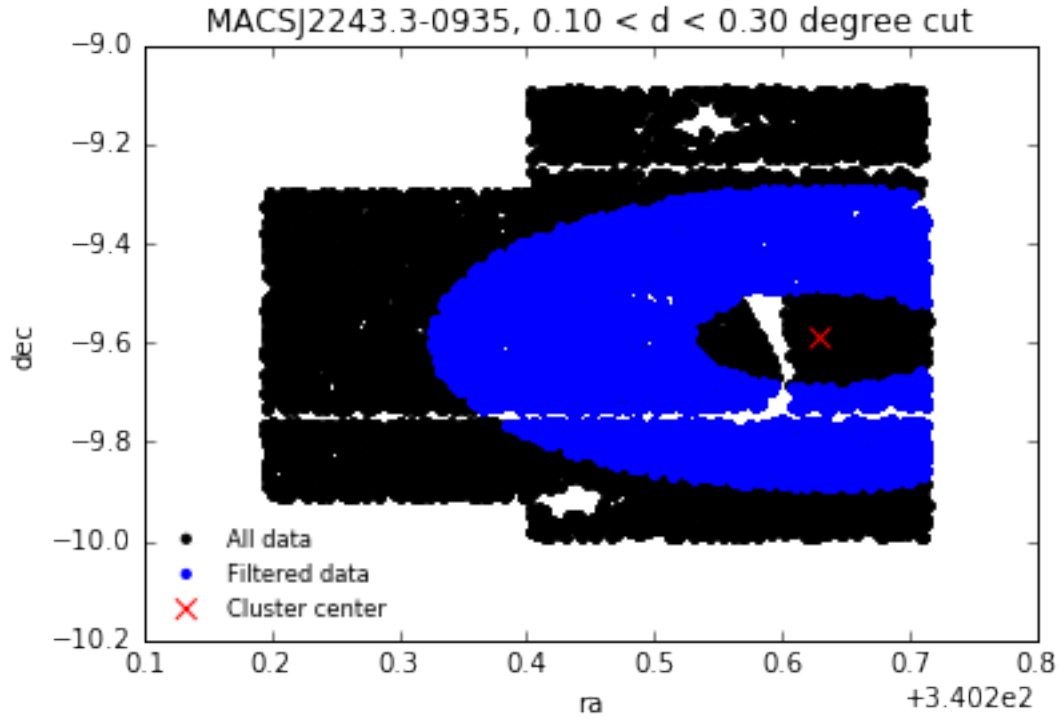
If you only want to work on a sample of galaxies center around the cluster at a certain radius, do:

```
confile = '/sps/lsst/data/clusters/MACSJ2243.3-0935/analysis/output_v1/MACSJ2243.3-
↪0935.yaml'
config = data.load_config(confile)
output = data.filter_around(fc_filtered, config, exclude_outer=20, exclude_inner=3,␣
↪unit='arcmin', plot=True)
```



The output of `filter_around` is a filtered data table. You can also choose a different unit:

```
output = data.filter_around(fc_filtered, config, exclude_outer=0.3, exclude_inner=0.1,
↪ unit='degree', plot=True)
```



CHAPTER 3

Data validation

Write the doc

CHAPTER 4

Extinction

Write the doc

CHAPTER 5

Photometric redshift

Write the doc

CHAPTER 6

Galaxy selection

Write the doc

CHAPTER 7

Shear

Write the doc

CHAPTER 8

Mass

Write the doc

CHAPTER 9

Code description

9.1 clusters package

9.1.1 Submodules

9.1.2 clusters.background module

Tools to fit the red sequence and extract background galaxies around a cluster.

clusters.background.**color_histo**(*mags*)
Plot color histograms.

clusters.background.**color_mag_plot**(*mags*)
Plot color / mag diagrams.

clusters.background.**fit_red_sequence**(*color*, *mag*, ***kwargs*)
Fit red sequence (RS) band in galaxy color plots, i.e m(i)-m(j) vs. m(k).

Parameters

- **color** (*list*) – A list of color mag_i - mag_j (ordinate)
- **mag** (*list*) – List of magnitude (abciss)
- ****kwargs** –
  - minc (float): lower cut on the color axis: color > minc (1.0)
  - maxc (float): upper cut of the color axis: color < maxc (2.0)
  - minm (float): lower cut on the mag axis: mag > minm (20.0)
  - maxm (float): upper cut of the mag axis: mag < maxm (23.5)
  - islope (float): first guess for the red sequence band slope (-0.04)
  - nbins (int): Number of bins used in the fits (40)
  - plot (bool): if True plot stuff

– verbose (bool): if True print information to screen

Returns

- slope of the red sequence band,
- ordinate at the origin of the red sequence band +/- 1.5 sigma

fitRedSequence is also producing some control plots

clusters.background.**get_rs_background**(*config*, *data*)
Return flag based on RS criterion for galaxy selection.

clusters.background.**get_zphot_background**(*config*, *zdata*, *zspec=None*, *z_config=None*, *thresh=None*, *zmin=None*, *zmax=None*, *plot=None*)
Return flag based on zphot criterion for galaxy selection.

clusters.background.**red_sequence_cut**(*config*, *data*, ***kwargs*)
Identify RS galaxies using color-magnitude diagram.

First do a radial cut on catalogue and identify the RS from the inner galaxies –> increase the contrast of the RS Then go back and apply the cut on the entire catalogue as some RS galaxies are located far away from the centre

Returns bool array, where False means the object does not pass the cut

List of available kwargs:

Parameters

- **mag_cut** (*float*) – rband magnitude cut - default is 25
- **plot** (*float*) – if keywords exists, plot stuff for visual inspection

clusters.background.**zphot_cut**(*zclust*, *zdata*, ***kwargs*)
Redshif selection of the galaxies used for analysis, using both: - hard cut, z_cl+0.1 < z_best < 1.25 (cf WtGIII) - cut from pdz. int_0^z_cl p(z) dz < x%

Parameters

- **plot** (*float*) – if keywords exists, plot stuff for visual inspection
- **thresh** (*float*) – tolerance x% for the pdz cut method.

Returns bool arrays, where False means the object does not pass the cut

9.1.3 clusters.data module

Data builder and parser for the Clusters package.

class clusters.data.**Catalogs**(*path*, *load_butler=True*)
Bases: object

Load data from a LSST stack butler path.

load_catalogs(*catalogs*, ***kwargs*)
Load a list of catalogs.

Parameters

- **catalogs** (*str/list*) – A catalog name, or a list of catalogs (see below)
- **keys** (*dict*) – A dictionnary of keys to load for each catalog

Available kwargs are:

Parameters

- **update** (*bool*) – Set to True if you want to update an already loaded catalog
- **show** (*bool*) – Set to True to get all available keys of a (list of) catalog(s)
- **matchid** (*bool*) – Will only keep objects which are in the deepCoad catalogs (to be used when loading the forced_src and deepCoadd catalogs)

Examples of catalogs that you can load:

- 'deepCoadd_ref',
- 'deepCoadd_meas',
- 'deepCoadd_forced_src',
- 'deepCoadd_calexp',
- 'forced_src'
- 'src'

save_catalogs(*output_name*, *catalog=None*, *overwrite=False*, *delete_catalog=False*)
Save the catalogs into an hdf5 file.

show_keys(*catalogs=None*)
Show all the available keys.

clusters.data.**get_astropy_table**(*cat*, ***kwargs*)
Convert an afw data table into a simple astropy table.

Parameters **cat** – an afw data table

Returns the corresponding astropy.table.Table

clusters.data.**save_wcs**(*wcs*, *output*)
Save the wcs dictionnary into a valid astropy Table format.

clusters.data.**shorten**(*doc*)
Hack to go around an astropy/hdf5 bug. Cut in half words longer than 18 chars.

9.1.4 clusters.extinction module

Converting tools for extinction.

clusters.extinction.**from_ebv_sfd_to_megacam_albd**(*ebv*)
Return A(lbd) for the 6 Megacam filters: u, g, r, i, z.

clusters.extinction.**from_ebv_sfd_to_sdss_albd**(*ebv*)
Return A(lbd) for the 5 SDSS filters: u, g, r, i, z.

clusters.extinction.**from_sdss_albd_to_megacam_albd**(*sdss*)
Return A(lbd) for the 6 Megecam filters: u, g, r, i_old, i_new, z.

clusters.extinction.**plots**(*ra*, *dec*, *ebv*, *albd*, *title=None*, *figname=''*)
Plot the extinction sky-map.

9.1.5 clusters.main module

9.1.6 clusters.shear module

Shear analysis.

clusters.shear.**analysis**(*table*, *xclust*, *yclust*, *e1='ext_shapeHSM_HsmShapeRegauss_e1'*, *e2='ext_shapeHSM_HsmShapeRegauss_e2'*, *config=None*, *datafile=None*, *step=200*)

Computethe shear.

Parameters

- **data_file** (*string*) – Name of the hdf5 file to load
- **path** (*string*) – Path (key) of the table to load

Returns

A dictionnary containing the following keys and values:

- meas: the 'deepCoadd_meas' catalog (an astropy table)
- forced: the 'deepCoad_forced_src' catalog (an astropy table)
- wcs: the 'wcs' of these catalogs (an `astropy.wcs.WCS` object)

clusters.shear.**compare_shear**(*catalogs*, *xclust*, *yclust*, *qcut=None*, *param='Tshear'*)

Compare shear mesured on the coadd and shear measured on indivial ccd.

For now, do: from clusters import data from clusters import shear config = data.load_config('MACSJ2243.3-0935.yaml') catalogs = data.read_hdf5('test_data2.hdf5') xc, yc = shear.xy_clust(config, data.load_wcs(catalogs['wcs'])) tables = shear.compare_shear([catalogs['deepCoadd_meas'], catalogs['forced_src']], xc, yc)

clusters.shear.**compute_shear**(*e1*, *e2*, *distx*, *disty*)

Compute the shear.

clusters.shear.**kappa_plot**(*x*, *y*, *e1*, *e2*)

clusters.shear.**plot_hist**(*xs*, *labels*, *nbins=200*, *xarange=(-2*, *2)*)

Plot multiple histograms in subplots.

clusters.shear.**plot_scatter**(*xs*, *ys*, *xlabels*, *ylabels*, ***kwargs*)

Plot multiple scatter plots in subplots.

Parameters

- **xs** (*list*) – List of arrays for x axis
- **ys** (*list*) – List of arrays for y axis
- **xlabels** (*str*) – List of x labels
- **ylabels** (*str*) – List of y labels

List of available kwargs: :param list yerrs: List of arrays, error on the y axis :param list xarange: Range for x axis (min,max) :param list yarange: Range for y axis (min,max)

clusters.shear.**plot_shear**(*gamt*, *gamc*, *dist*, *drange=(0*, *8500)*, *nbins=8*)

Plot shear.

clusters.shear.**quiver_plot**(*meas*)

clusters.shear.**xy_clust**(*config*, *wcs*)

Return xy coordinate (pixel).

9.1.7 clusters.validation module

Data validation utilisites and plots.

clusters.validation.**check_star_elipticities**(*d*, *cat='deepCoadd_meas'*, *oid='id'*)
Compute star elipticities from second momments and check if psf correction is valid.

Also check magnitude vss radius

clusters.validation.**compute_elipticities**(*xx*, *yy*, *xy*)
Compute star elipticities from second momments.

clusters.validation.**define_selection_filter**(*d*, *cat*)
Define and return a standard quality selection filter.

clusters.validation.**get_filter_list**(*table*)
Get the filter list and number of filter in a table.

clusters.validation.**load_cluster**(*cluster='MACSJ2243.3-0935'*, *ifilt='i_new'*)
Load the data for a given cluster.

clusters.validation.**separate_star_gal**(*d*, *cat*, *oid*, *nfilters*, *filt=None*)
Return two clean tables: one for the stars, the other for the galaxies.

clusters.validation.**stellarLocus**(*d*, *mag_type='modelfit_CModel_mag_extcorr'*, *ifilt='i_new'*, *cat='deepCoadd_forced_src'*)
Check colors by plotting stellar loci and comparing with analytical fits.

First a few color-color (and one mag-color) plots are plotted based on the input magnitudes. Since analytical fits are based on SDSS data, the given magnitudes are then converted to SDSS mags. Fits are overplotted with the derived SDSS magnitudes, and then residuals are calculated and plotted. The analytical plots are plotted as an intermediary as well.

Three plots are saved. Nothing is returned.

9.1.8 clusters.zphot module

Photometric redshift analysis. Includes a wrapper to LEPHARE and BPZ.

- LEPHARE: http://www.cfht.hawaii.edu/~arnouts/LEPHARE/lephare.html
- BPZ: http://www.stsci.edu/~dcoe/BPZ

class clusters.zphot.**BPZ**(*magnitudes*, *errors*, *zpara=None*, *spectro_file=None*, ***kwargs*)
Bases: object

Wrapper to the BPZ photometric redshift code.

http://www.stsci.edu/~dcoe/BPZ

build_columns_file(*prefix='CFHT_megacam_'*, *sufix='p'*, *filters=None*, *ref='i'*, *z_s=False*)
Build and write the 'columns' file.

Hardcoded for test purpose.

run()
Run BPZ.

Configuration file must exist in the current directory.

Todo: Build the configuration file on the fly (the .columns)

write_input()
Create and write files needed to run BPZ.

- the input data file for BPZ

- a similar file containing the sources ID along with their RA DEC.

class clusters.zphot.**LEPHARE**(*magnitudes*, *errors*, *zpara=None*, *spectro_file=None*, ***kwargs*)

Bases: object

Wrapper to the LEPHARE photometric redshift code.

http://www.cfht.hawaii.edu/~arnouts/LEPHARE/lephare.html

check_config(*config=None*)

Check that the SED and filters requested for the LePhare run do exist.

If not: explains where the problem is and aborts.

run(*config=None*)

Run LEPHARE.

Default config file is $LEPHAREDIR/config/zphot_megacam.para. Can be overwritten with the config argument

write_input()

Create and write files needed to run LEPHARE.

- the input data file for LEPHARE
- a similar file containing the sources ID along with their RA DEC.

class clusters.zphot.**ZPHOTO**(*zphot_output*, *zphot_pdz_output*, *zcode_name=None*, *all_input=None*, ***kwargs*)

Bases: object

Read photoz code (LePhare, BPZ) output file and creates/saves astropy tables.

hist(*param*, ***kwargs*)

Plot histograms.

Possible kwargs

Params float minv Lower value of the histogram

Params float maxv Upper value of the histogram

Params int nbins Number of bins. Default is 10.

Params string xlabel An xlbal for the figure

Params string title A title for the figure

Params float zclust Redshift of the studies cluster

plot(*px*, *py*, ***kwargs*)

Plot x vs. y.

Possible kwargs are: :params float minx: lower limit of the x axis :params float maxx: upper limit of the x axis :params float miny: lower limit of the y axis :params float maxy: upper limit of the y axis :params string xlabel: label of the x axis :params string ylabel: label of the y axis :params string title: title of the figure

plot_map(*title=None*, *zmin=0*, *zmax=999*)

Plot the redshift sky-map.

read()

Read the output.

read_input()

Read the input.

save_zphot(*file_out*, *path_output*, *overwrite=False*)
Save the output of photoz code (z_best, chi^2, pdz) into astropy table.

class clusters.zphot.**ZSPEC**(*sfile*, *names*, *unit='deg'*)
Bases: object

Compare spectroscopic and photometric redshifts.

load_zphot(*ra*, *dec*, *zphot*, *unit='deg'*)
Load the photometric informations and match them to the spectro ones.

Parameters

- **ra** (*list*) – List of RA coordinates
- **dec** (*list*) – List of DEC coordinates
- **zphot** (*list*) – List of photometric redshift
- **unit** (*list*) – List of RA coordinates

All lists must have the same length.

plot(*cut=300*, *path_to_png=None*)
Plot a sky-map of the matches.

scatter(*zclust*, *cluster=None*, *cut=0.1*, *stability=False*)
Redshift scatter in the cluster.

Plot the spectroscopic redshift distribution and apply a gaussian fit.

clusters.zphot.**dict_to_array**(*d*, *filters='ugriz'*)
Transform a dictionnary into a list of arrays.

clusters.zphot.**gauss**(*x*, **p*)
Model function to be used to fit a gaussian distribution.

9.1.9 Module contents

Cluster analysis on the LSST DM stack.

9.2 Indices and tables

- genindex
- modindex
- search

CHAPTER 10

Todo

10.1 General todo list

- Data
  - Complete the doc
- Extinction
  - Plug the Extinction package in and remove all internal extinction-related code

10.2 Code oriented todo list

Todo: Build the configuration file on the fly (the .columns)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/clusters/checkouts/latest/clusters/zphot.py:docstring of clusters.zphot.BPZ.run, line 5.)

Python Module Index

C

Index

S

W

X

Z