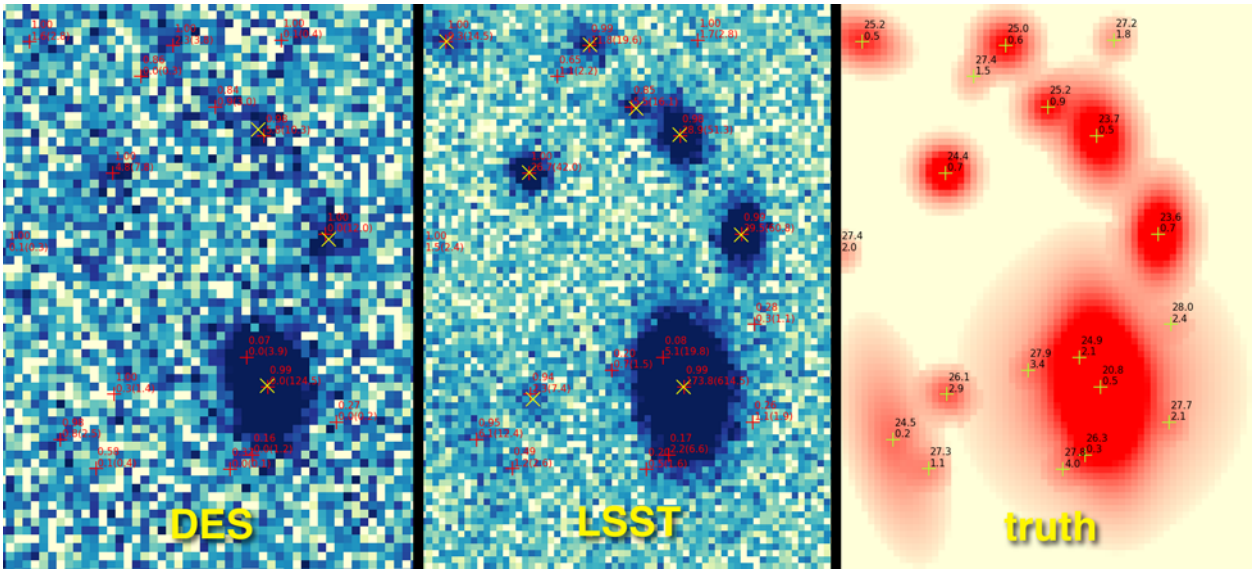
WeakLensingDeblending Documentation

Release 0.4

David Kirkby

August 09, 2016

Contents



Fast simulations and analysis for the Weak Lensing Working Group of the LSST Dark Energy Science Collaboration.

This software was primarily developed to study the effects of overlapping sources on shear estimation, photometric redshift algorithms, and deblending algorithms. Users can run their own simulations (of LSST and other surveys) or, else download the galaxy catalog and simulation outputs to use with their own code or analyze with the tools provided here.

The code is hosted on github. Please use the issue tracker to let us know about any issues you have with installing or running this code, or to request new features.

CHAPTER 1

Quickstart Tutorial

The tutorial is in the form of a jupyter notebook. You can either:

- Browse the tutorial online
- Install this package and run the notebook locally.

The first method is the easiest since it only requires a web browser. The second method requires that you have python and jupyter installed, but allows you to make changes to the tutorial and experiment.

CHAPTER 2

Frequently Asked Questions

2.1 How do I...

2.1.1 . . . get started with this package?

Start with the quickstart tutorial to get an overview of this package, then select one of the installation methods.

2.1.2 . . . simulate my own source catalogs?

If your source objects are galaxy-like, you can format you catalog with the columns *described here*. Otherwise, *create an issue* describing your catalog and what you are trying to do.

2.1.3 . . . simulate other instruments?

If the instrument you would like to simulate is similar to one of the defaults (LSST, DES, CFHT), you can simply modify a few parameters. See the quickstart tutorial for examples. You can also define a new default survey configuration by modifying the file `descwl/survey.py`. Instruments are described with a simple model that may not be sufficient for your needs: in this case, please *create an issue* describing what you are trying to do.

2.1.4 . . . add my own pixel-level analysis?

In case the existing output catalog does not already calculate the quantities you need, you can either write your own post-processor using the skeleton code provided, or else modify the `descwl/analysis.py` file that generates the output catalog. In either case, feel free to *create an issue* to let us know what you are trying to do and get feedback.

2.1.5 . . . ask questions or report bugs?

The code is hosted on github. Please use the issue tracker to let us know about any issues you have with installing or running this code, or to request new features.

2.1.6 . . . contribute to the code?

This software is open source and your contributions are welcome! General information for package developers is here. If you would like to add a new feature, please start by *creating a new issue* to describe it.

CHAPTER 3

Installation

3.1 Install with GIT

The code is hosted on github so the easiest method to perform an initial installation is with git:

```
git clone https://github.com/DarkEnergyScienceCollaboration/WeakLensingDeblending.git
```

This will create a new subdirectory *WeakLensingDeblending* containing the latest stable version of the complete package.

Experts who already have a correctly configured github account might prefer this alternative:

```
git clone git@github.com:DarkEnergyScienceCollaboration/WeakLensingDeblending.git
```

3.2 Update with GIT

You can update your local copy of the package at any time using:

```
cd WeakLensingDeblending
git update
```

3.3 Getting Started

Programs can be run directly from the top-level directory without needing to set *PYTHONPATH* as long as you have the required packages already installed, e.g.:

```
cd WeakLensingDeblending
./simulate.py --help
```

For an introduction to the available programs, see here and for examples of running these programs see here.

3.4 Required Packages

The following python packages are required by this package:

- numpy (version >= 1.9)
- astropy (version >= 0.4)

- fitsio (version >= 0.9.6)
- galsim (version >= 1.2)
- lmfit (version >= 0.8.3)

Note that *numpy* and *astropy* are both available in recent anaconda or enthought canopy distributions. The *fitsio* package is required for performance reasons, since the similar pure-python functionality in the `astropy.io.fits` module is too slow for this application. Installing GalSim is a more involved process, but well worth the effort. The *lmfit* package is only required if you will be running your own simulations.

You can check your astropy version using:

```
import astropy.version
print astropy.version.version
```

A version of at least 0.4 is required due to recent changes in tables. If you have a pre-0.4 version of astropy via anaconda, you can update using:

```
sudo conda update conda
sudo conda update anaconda
```

Note that some additional packages are required to *query the LSST DM catalog*, but you will not normally need to do this.

The psutil package is required if you use the *–memory-trace* command-line argument to the *simulate* program, but you normally would not need to do this.

CHAPTER 4

Source Catalogs

4.1 Galaxy Catalog Format

The input catalog is a text file (or an *equivalent FITS table*) consisting of an initial header line and followed by one line of numbers per catalog object, for example a catalog with two objects might look like this:

```
galtileid ra dec redshift fluxnorm_bulge fluxnorm_disk fluxnorm_agn a_b a_d b_b b_d pa_bulge pa_disk
2200871446 0.418319702147 -0.000148399994941 0.496377289295 0.0 1.4144730572e-17 0.0 0.0 0.278649687
2205921112 0.420028448104 -0.00100259995088 1.89508104324 0.0 1.91501907101e-18 0.0 0.0 0.3580636978
```

The header line specifies a list of names, separated by white space, associated with each catalog parameter. Each catalog entry is represented by a a list of numbers, separated by white space, giving the corresponding parameter values.

The parameter names below are required by the *simulate* program, but other parameters may also be present in the file. Parameters can be listed in any order as long as the order of values matches the header line. The names in the table below are a bit idiosyncratic (e.g., mixing under_scores with CamelCase and using different schemes to denote bulge vs. disk) but are chosen to match the names used in the LSST DM galaxy catalog schema.

Name	Description
galtileid	Unique integer identifier for this object
ra	Object centroid right ascension (degrees)
dec	Object centroid declination (degrees)
redshift	Object cosmological redshift
fluxnorm_bulge	Multiplicative scaling factor to apply to the bulge SED
fluxnorm_disk	Multiplicative scaling factor to apply to the disk SED
fluxnorm_agn	Multiplicative scaling factor to apply to the AGN SED
a_b	Semi-major axis of bulge 50% isophote (arcseconds)
b_b	Semi-minor axis of bulge 50% isophote (arcseconds)
a_d	Semi-major axis of disk 50% isophote (arcseconds)
b_d	Semi-minor axis of disk 50% isophote (arcseconds)
pa_bulge	Position angle of bulge (degrees)
pa_disk	Position angle of disk (degrees)
u_ab	Apparent AB magnitude in the LSST u-band, including extinction effects
g_ab	Apparent AB magnitude in the LSST g-band, including extinction effects
r_ab	Apparent AB magnitude in the LSST r-band, including extinction effects
i_ab	Apparent AB magnitude in the LSST i-band, including extinction effects
z_ab	Apparent AB magnitude in the LSST z-band, including extinction effects
y_ab	Apparent AB magnitude in the LSST y-band, including extinction effects

The catalog file is read using a `astropy.io.ascii.Basic` reader (created with default options) so can be embellished with comments and blank lines for readability, as supported by that class.

4.2 Create Galaxy Catalog From LSST Catalog Database

This section documents the process for creating an input catalog derived from the LSST Data Management (DM) simulation database, described in these SPIE proceedings and being documented here. This is not something you will normally need (or want) to do since the resulting catalog is already available to download. However, if you want to know exactly how the catalog was created or want to create your own, read on.

> **Warning:** In case you are using the DM galaxy database directly, you should **avoid using the DiskHalfLightRadius and BulgeHalfLightRadius columns** since these do not have the usual definition where *hlr == sqrt(a*b)* and instead satisfy *hlr == a*. To avoid this confusion, use the *a,b* parameters directly and calculate *hlr == sqrt(a*b)*.

The *dbquery.py* program automates the process of connecting to the database, extracting the relevant data, and writing a catalog file. The program uses the pymsql python interface to Microsoft SQL Server (which LSST DM uses), so you will need to install that and its dependencies (FreeTDS and Cython) in order to run *dbquery.py*.

The *OneDegSq.dat* catalog file was created using:

```
dbquery.py -o OneDegSq.dat --dec-min -0.5 --dec-max +0.5 --ra-min -0.5 --ra-max +0.5 --verbose
```

with FreeTDS v0.91, Cython v0.21, pymsql v2.1.1 under OS-X 10.10.1. The program takes about 2 minutes to run and saves 858502 rows to the 200 Mbyte output text file. The set of rows returned by a query is invariant, but they are returned in an arbitrary order so the output files obtained by running this program twice can be different (but only by a permutation of lines). Note that the "1 sq.deg." here is defined by RA and DEC intervals of 1 degree centered on (RA,DEC) = (0,0), which is not exactly 1 sq.deg. of 2D imaging because of projection effects.

The query uses a custom stored procedure that tiles the full sky by repeating the same 4deg x 4deg patch of about 14M galaxies (the catalog actually contains a 4.5deg x 4.5deg patch, but only the smaller patch is tiled). The stored routine synthesizes the returned *ra*, *dec*, and *galtileid* values, where:

```
galtileid = TTTTGGGGGGGG
```

is a 64-bit integer that combines the tile number *TTTT* (0-4049) with the unique galaxy ID *GGGGGGGG* (0-17,428,283). The ID that *dbquery* writes to its output file is *galtileid*. For the *OneDegSq.dat* example above, all galaxies are unique and located in tileid = 22 or 4027.

Note that the LSST database uses standard port assignments for its Microsoft SQL Server. However, since these ports are frequently targets of network attacks, many organizations block outbound packets to these ports from internal IP addresses. In addition, the UW hosts of the database only allow inbound packets to their SQL server from IP address ranges included in their whitelist. Therefore, if you are unable to connect, the most likely reason is packet filtering on one or both ends of your connection. One option is to use a host that has already been configured for access to the LSST catalog database (on the NCSA cluster, for example).

You might find the following interactive python snippet useful for debugging connection problems:

```
import _mssql
conn = _mssql.connect(server='fatboy.npl.washington.edu', user='LSST-2', password='L$$TUser', databas
print conn.tds_version
conn.execute_query("Select name from sysobjects where type like 'u'")
for row in conn: print row['name']
conn.execute_query("select COLUMN_NAME from INFORMATION_SCHEMA.COLUMNS where TABLE_NAME = 'galaxy'")
for row in conn: print row[0]
conn.execute_scalar("select count(*) from galaxy")
conn.close()
```

The second line will fail with a connection error after about 30 seconds if your packets are being filtered on either end:

```
MSSQLDatabaseException: (20009, 'DB-Lib error message 20009, severity 9:\nUnable to connect: Adaptive
```

4.3 Convert ASCII Catalog to FITS Table

The catalog file can also be read as a FITS file containing a single table. A catalog in text format can be converted to a FITS file using, for example (this will not overwrite an existing output file):

```
import astropy.table
catalog = astropy.table.Table.read('OneDegSq.dat',format='ascii.basic')
catalog.write('OneDegSq.fits')
```

The resulting FITS file will be somewhat smaller (by about 30%) than the text original and significantly faster for the *simulate* program to read, but may be less convenient for other programs to read or generate.

CHAPTER 5

Examples

5.1 Command-line Options

Print out usage info for command-line options:

```
./simulate.py --help
./display.py --help
./fisher.py --help
```

5.2 Survey Parameters

Print default camera and observing condition parameters for each supported (survey,filter) combination:

```
./simulate.py --survey-defaults
```

5.3 Quick Simulation Demo

Simulate an LSST i-band stack for a small (512x512) field:

```
./simulate.py --catalog-name OneDegSq.fits --image-width 512 --image-height 512 --survey-name LSST --
```

Make a finder chart for overlapping groups:

```
./display.py --input-name demo --info '%(grp_id)ld' --select 'grp_size>1' --select 'grp_rank==0' --ma
```

Display a single blended group:

```
./display.py --input-name demo --info 'z=%(z).1f' --crop --group 2202242785 --magnification 16
```

Plot the Fisher matrix calculations for this group:

```
./fisher.py --input-name demo --galaxy 2202242785 --verbose
./fisher.py --input-name demo --group 2202242785 --verbose
./fisher.py --input-name demo --group 2202242785 --correlation
```

5.4 Data Products Demo

Download the *LSST_i.fits* data product, then display the location of galaxies that are unresolved due to blending:

```
./display.py --input-name LSST_i --magnification 0.25 --select 'snr_grpf<5' --select 'snr_sky>10' --c
```

Compare with SExtractor detected objects for one cluster:

```
./display.py --input-name LSST_i --crop --group 402700079754 --magnification 8 --match-catalog LSST_
```

Compare simulations of the same region in different surveys:

```
./display.py -i LSST_i --magnification 10 --select-region '[-10.20,2.80,-30.40,-12.80]' --view-regio

./display.py -i LSST_i --match-catalog LSST_i_noise.cat --magnification 10 --select-region '[-10.20,2

./display.py -i DES_i --match-catalog DES_i_noise.cat --magnification 13.15 --select-region '[-10.20,
```

CHAPTER 6

Programs

All available programs are contained in the top-level directory where this package is installed. Programs are written in python and use the '.py' file extension.

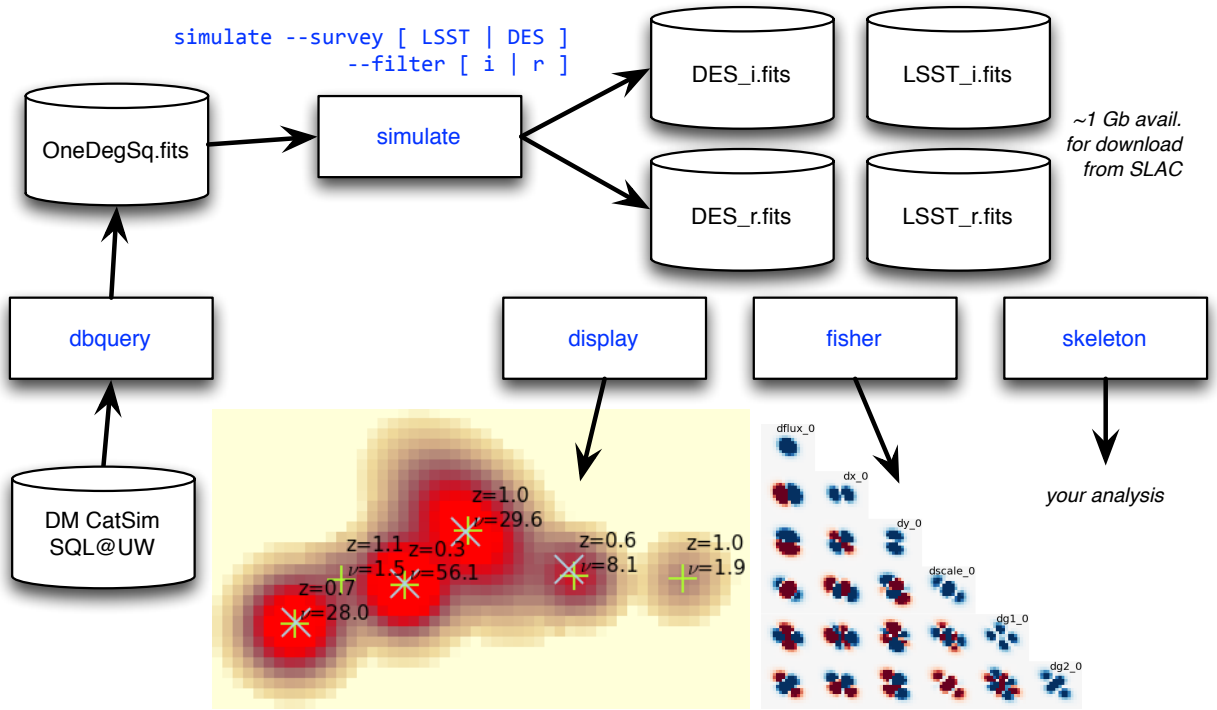
All programs are configured by passing command-line options. Pass the *–help* option to view documentation for these options, e.g.:

```
./simulate.py --help
```

Note that some program arguments normally contain characters that your shell may interpret specially but you can generally avoid this by quoting these arguments. For example, the square brackets [ ] and parentheses ( ) in the following command line are problematic and should be quoted as shown:

```
./display.py -i demo --crop --select-region '[-10.20,2.80,-30.40,-12.80]' --info '%(grp_size)d' --mag
```

This document provides an introduction to each of the available programs. Examples of running the programs are documented elsewhere. The flowchart below shows the main processing steps and relationships between these programs.



6.1 simulate

The *simulate* program simulates and analyzes a weak-lensing survey by performing fast image rendering with galsim.

The program reads a catalog of astronomical sources and produces a set of output data products. Simulation outputs can be displayed with the *display* and *fisher* programs, or you can modify the *skeleton* program to perform your own analysis of the outputs.

By default, the program will simulate and analyze an area corresponding to 1 LSST chip (about 13 x 13 arcmin**2) assuming a full-depth LSST i-band exposure with nominal PSF, sky background, extinction, zero points, etc. To simulate a different nominal survey use, e.g.:

```
./simulate.py --survey LSST --filter r
./simulate.py --survey DES --filter i
```

Use the following command to see all the defaults associated with each survey configuration:

```
./simulate.py --survey-defaults
```

You can override any of these defaults on the command line, e.g.:

```
./simulate.py --survey CFHT --sky-brightness 20.7 --airmass 1.3 --atmospheric-psf-e1 0.01
```

You can also create new default survey configurations by editing the `descwl.survey` module.

The main simulation parameter is the pixel signal-to-noise threshold, *min-snr*:

```
./simulate.py --min-snr 0.1 ...
```

Sources are simulated over a footprint that includes exactly those pixels whose noise-free mean signal is above this threshold. Any source with at least one pixel meeting this criterion is considered 'visible' and others are discarded from the subsequent analysis and output. The threshold is in units of the full-depth mean sky noise level and the default value is 0.05.

For a full list of the available options, use:

```
./simulate.py --help
```

For details on the simulation output format, see this page.

6.2 display

The *display* program displays simulated images and analysis results stored in the FITS file generated by the *simulate* program.

Certain objects can be selected for highlighting and annotating. Use the *–galaxy* command-line arg to select a single galaxy using its unique database identifier (*db_id* in the *Analysis Results*). Similarly, use *–group* to select all galaxies in an overlapping group identified by its *grp_id*. The *–galaxy* and *–group* args can be repeated and combined and the resulting selection is the logical OR of each specification. More generally, objects can be selected using any expression of the variables in the *Analysis Results* with the *–select* option. For example, the following args select objects that are visible and pass a minimum signal-to-noise ratio cut:

```
./display.py --select 'visible' --select 'snr_iso > 10' ...
```

The *–select* arg can be repeated and the results will be combined with logical AND, followed by the logical OR of any *–galaxy* or *–group* args.

Displayed pixel values are clipped using an upper limits set at a fixed percentile of the histogram of non-zero pixel values for the selected objects, and a lower limit set a a fixed fraction of the mean sky noise level. Use the *–clip-hi-percentile* and *–clip-lo-noise-fraction* command-line args to change the defaults of 90% and 0.1, respectively. Clipped pixel values are rescaled from zero to one and displayed using the DS9 sqrt scale.

Scaled pixel values are displayed using two colormaps: one for selected objects and another for background objects. Use the *–highlight* and *–colormap* command-line args to change the defaults using any named matplotlib colormap.

Selected objects can be annotated using any information stored in the *Analysis Results*. Annotations are specified using python format strings where all catalog variables are available. For example, to display redshifts with 2 digits of precision, use:

```
./display.py --info '%(z).2f' ...
```

More complex formats with custom styling options are also possible, e.g.:

```
./display.py --info '$\\nu$ = %(snr_isof).1f,%(snr_grpf).1f' --info-size large --info-color red ...
```

The results of running an independent object detection pipeline can be superimposed in a displayed image. The *match-catalog* option specifies the detections to use in SExtractor compatible format. The matching algorithm is described *here*. Matches can also be annoted, e.g.:

```
./display.py --match-catalog SE.cat --match-info '%(FLUX_AUTO).1f'
```

For a full list of available options, use:

```
./display.py --help
```

6.3 fisher

The *fisher* program creates plots to illustrate galaxy parameter error estimation using Fisher matrices.

The program either calculates the Fisher matrix for a single galaxy (*–galaxy*) as if it were isolated, or else for an overlapping group of galaxies (*–group*). The displayed image consists of the lower-triangular part of a symmetric *npar x npar* matrix, where:

```
npar = (num_partials=6) * num_galaxies
```

By default, the program displays the Fisher-matrix images whose sums (over pixels) give the Fisher matrix element values. Alternatively, you can display the partial derivative images (*–partials*), Fisher matrix elements (*–matrix*), covariance matrix elements (*–covariance*) or correlation coefficients matrix (*–correlation*).

Use the *–colormap* option to select the color map. The vertical color scales are optimized independently for each partial-derivative or Fisher-matrix image, but are guaranteed to use ranges that are symmetric about zero (so diverging colormaps are usually the best choice). When Fisher or covariance matrix elements are being displayed, their relative values are somewhat arbitrary since they generally have different units. However, the dimensionless correlation coefficient matrix is always displayed using a scale range of [-1,+1].

6.4 skeleton

The *skeleton* program provides a simple demonstration of reading and analyzing simulation output that you can adapt to your own analysis. See the comments in the code for details.

6.5 dbquery

The *dbquery* program queries the official LSST simulation galaxy catalog and writes a text catalog file in the format expected by the *simulate* program. You do not normally need to run this program since suitable catalog files are already provided.

CHAPTER 7

Simulation Output

This document describes the content and format of the output file produced by the *simulate* program.

The output consists of a FITS file containing several header/data units (HDUs).

You can inspect an output file's contents from an interactive python session, e.g.:

```
import fitso
fits = fitsio.FITS('demo.fits')
print fits[0] # simulated survey image
print fits[1] # analysis results
fits.close()
```

You can reconstruct the analysis `descwl.analysis.OverlapResults` using the `descwl.output.Reader` class, e.g.:

```
import descwl
results = descwl.output.Reader('demo.fits').results
print results.survey.description()
```

You will need to ensure that the *descwl* package directory is in your *$PYTHONPATH* for this example to work. See the *skeleton* program for a more complete example of using analysis results from python.

7.1 Simulated Survey Image

The primary HDU contains the final simulated image using double precision floats. All sources are superimposed in this source and fluxes are given in units of detected electrons during the full exposure time.

All of the `descwl.survey.Survey` constructor args are saved as header keywords in the primary HDU, using only the last eight characters in upper case for the corresponding keys. In addition, the class attribute variables listed below are saved to the header.

Key	Class	Attribute	Description
NSLICES	`descwl.analysis.OverlapResults`	*num_slices*	Number of slices for each per-object datacube
PSF_SIGM	`descwl.survey.Survey`	*psf_sigma_m*	PSF size |Q|**0.25 in arcsecs (unweighted Q)
PSF_SIGP	`descwl.survey.Survey`	*psf_sigma_m*	PSF size (0.5*trQ)**0.5 in arcsecs (unweighted Q)
PSF_HSM	`descwl.survey.Survey`	*psf_size_hsm*	PSF size |Q|**0.25 in arcsecs (weighted Q)

To write a survey image with Poisson sky noise added to a new file, use e.g.:

```
import galsim,descwl
results = descwl.output.Reader('LSST_i.fits').results
results.add_noise(noise_seed=1)
galsim.fits.write(results.survey.image,'LSST_i_noise.fits')
```

7.2 Analysis Results

HDU[1] contains a binary table where each row represents one simulated source and the columns are described in the table below. Q refers to the second-moment tensor of the galaxy's combined (bulge + disk + AGN) 50% isophote, including any cosmic shear but not the PSF. M refers to the second-moment tensor estimated using adaptive pixel moments and including the PSF. Note that the *e1,e2* and *hsm_e1,hsm_e2* parameters use different ellipticity conventions. HSM below refers to algorithms described in Hirata & Seljak (2003; MNRAS, 343, 459) and tested/characterized using real data in Mandelbaum et al. (2005; MNRAS, 361, 1287).

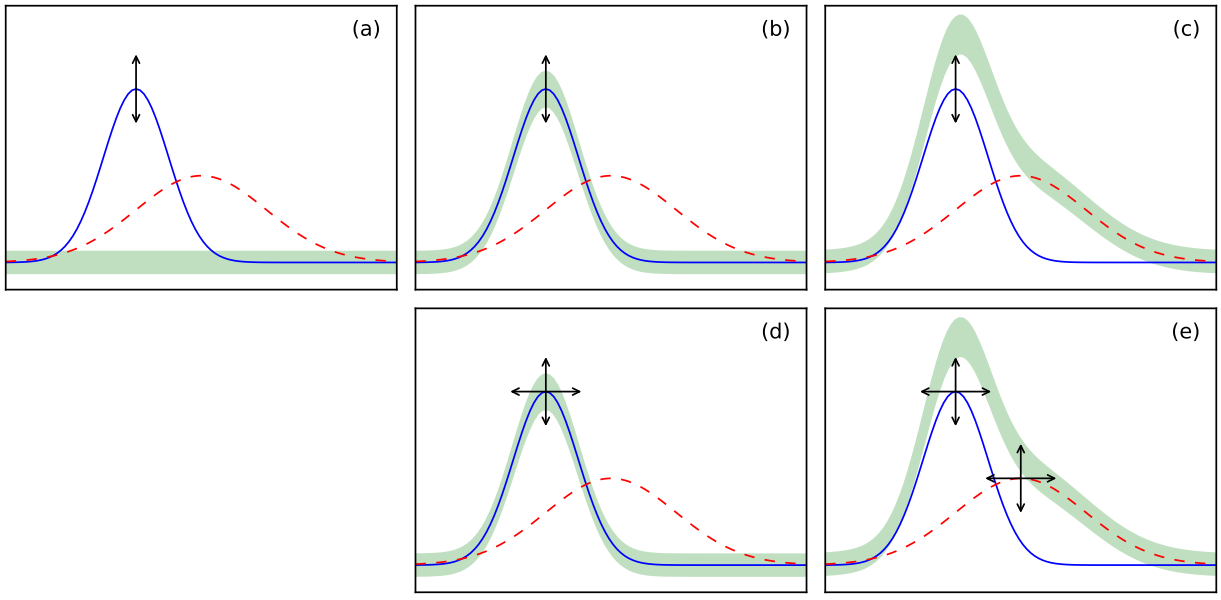
Name	Type	Description
db_id	int64	Unique identifier for this source in the LSST DM catalog database
grp_id	int64	Group identifier (db_id of group member with largest snr_grp)
grp_size	int16	Number of sources in this group (equal to 1 for isolated sources)
grp_rank	int16	Rank position of this source in its group based on decreasing snr_iso
visible	int16	Is this source's centroid within (1) our outside (0) the simulated image bounds?
Stamp Bounding Box		
xmin	int32	Pixel offset of left edge of bounding box relative to left edge of survey image
xmax	int32	Pixel offset of right edge of bounding box relative to left edge of survey image
ymin	int32	Pixel offset of bottom edge of bounding box relative to bottom edge of survey image
ymax	int32	Pixel offset of top edge of bounding box relative to bottom edge of survey image
Source Properties		
f_disk	float32	Fraction of total galaxy flux to due a Sersic n=1 disk component
f_bulge	float32	Fraction of total galaxy flux to due a Sersic n=4 bulge component
dx	float32	Source centroid in x relative to image center in arcseconds
dy	float32	Source centroid in y relative to image center in arcseconds
z	float32	Catalog source redshift
ab_mag	float32	Catalog source AB magnitude in the simulated filter band
ri_color	float32	Catalog source color calculated as (r-i) AB magnitude difference
flux	float32	Total detected flux in electrons
sigma_m	float32	Galaxy size arcseconds calculated as |Q|**0.25
sigma_p	float32	Galaxy size in arcseconds calculated as (0.5*trQ)**0.5
e1	float32	Real part (+) of galaxy ellipticity spinor (Q11-Q22)/(Q11+Q22+2|Q|**0.5)
e2	float32	Imaginary part (x) of galaxy ellipticity spinor (2*Q12)/(Q11+Q22+2|Q|**0.5)
a	float32	Semi-major axis of 50% isophote ellipse in arcseconds, derived from Q
b	float32	Semi-minor axis of 50% isophote ellipse in arcseconds, derived from Q
beta	float32	Position angle of second-moment ellipse in radians, or zero when a = b
psf_sigm	float32	PSF-convolved half-light radius in arcseconds calculated as |Q|**0.25
Pixel-Level Properties		
purity	float32	Purity of this source in the range 0-1 (equals 1 when grp_size is 1)
snr_sky	float32	S/N ratio calculated by ignoring any overlaps in the sky-dominated limit (a)
snr_iso	float32	Same as snr_sky but including signal variance (b)
snr_grp	float32	Same as snr_sky but including signal+overlap variance (c)
snr_isof	float32	Same as snr_grp but including correlations with fit parameters for this source (d)
snr_grpf	float32	Same as snr_grp but including correlations with fit parameters for all sources (e)
Error on parameters (Square root of covariance matrix elements calculated from the Fisher Formalism)		

Continued on next page

Table 7.1 – continued from previous page

Name	Type	Description
ds	float32	Error on scale dilation factor (nominal s=1) marginalized over flux,x,y,g1,g2 (d)
dg1	float32	Error on shear + component (nominal g1=0) marginalized over flux,x,y,scale,g2 (d)
dg2	float32	Error on shear x component (nominal g2=0) marginalized over flux,x,y,scale,g1 (d)
ds_grp	float32	Same as ds but also marginalizing over parameters of any overlapping sources (e)
dg1_grp	float32	Same as dg1 but also marginalizing over parameters of any overlapping sources (e)
dg2_grp	float32	Same as dg2 but also marginalizing over parameters of any overlapping sources (e)
Bias on parameters (Calculated from Fisher Formalism)		
bias_f	float32	Bias on galaxy's flux marginalized over scale,x,y,g1,g2.
bias_s	float32	Bias on scale dilation factor (nominal s=1) marginalized over flux,x,y,g1,g2
bias_g1	float32	Bias on shear + component (nominal g1=0) marginalized over flux,x,y,scale,g2
bias_g2	float32	Bias on shear x component (nominal g2=0) marginalized over flux,x,y,scale,g1
bias_x	float32	Bias on source's centroid in x relative to image center in arcseconds marginalized over flux,y,scale,g1,g2
bias_y	float32	Bias on source's centroid in y relative to image center in arcseconds marginalized over flux,x,scale,g1,g2
bias_f_grp	float32	Same as bias_f but also marginalizing over parameters of any overlapping sources.
bias_s_grp	float32	Same as bias_s but also marginalizing over parameters of any overlapping sources.
bias_g1_grp	float32	Same as bias_g1 but also marginalizing over parameters of any overlapping sources.
bias_g2_grp	float32	Same as bias_g2 but also marginalizing over parameters of any overlapping sources.
bias_x_grp	float32	Same as bias_x but also marginalizing over parameters of any overlapping sources.
bias_y_grp	float32	Same as bias_y but also marginalizing over parameters of any overlapping sources.
HSM Analysis Results (ignoring overlaps)		
hsm_sigm	float32	Galaxy size |M|**0.25 in arcseconds from PSF-convolved adaptive second moments
hsm_e1	float32	Galaxy shape e1=(M11-M22)/(M11+M22) from PSF-convolved adaptive second moments
hsm_e2	float32	Galaxy shape e1=(2*M12)/(M11+M22) from PSF-convolved adaptive second moments
Systematics Fit Results		
g1_fit	float32	Best-fit value of g1 from simultaneous fit to noise-free image
g2_fit	float32	Best-fit value of g2 from simultaneous fit to noise-free image

The figure below illustrates the different Fisher-matrix error-estimation models (a-e) used to define the pixel-level properties and referred to in the table above. The green bands show the variance used in the Fisher-matrix denominator and the arrows indicate the parameters that are considered floating for calculating marginalized parameter errors. Vertical arrows denote flux parameters and horizontal arrows denote the size and shape parameters (dx,dy,ds,dg1,dg2).



If any Fisher matrix is not invertible or yields non-positive variances, galaxies are iteratively dropped (in order of increasing snr_iso) until a valid covariance is obtained for the remaining galaxies. The corresponding values in the analysis results table will be zero for signal-to-noise ratios and infinite (*numpy.inf*) for errors on s,g1,g2.

You can load just the analysis results catalog from the output file using, e.g.:

```
import astropy.table
catalog = astropy.table.Table.read('demo.fits',hdu=1)
```

To scroll through the table in an interactive python session, use:

```
catalog.more()
```

To browse the catalog interactively (including seaching and sorting), use:

```
catalog.show_in_browser(jsviewer=True)
```

To plot a histogram of signal-to-noise ratios for all visible galaxies (assuming that *matplotlib* is configured):

```
plt.hist(catalog['snr'][catalog['visible']])
```

7.3 Rendered Galaxy Stamps

HDU[n+1] contains an image data cube for stamp n = 0,1,... Each data cube HDU has header keywords *X_MIN* and *Y_MIN* that give the pixel offset of the stamp's lower-left corner from the lower-left corner of the full simulated survey image. Note that stamps may be partially outside of the survey image, but will always have some pixels above threshold within the image.

7.4 DS9 Usage

If you open an output file with the DS9 program you will normally only see the full simulated survey image in the primary HDU. You can also use the *File* > *Open As* > *Multiple Extension Cube...* to view the nominal rendered stamp for each visible galaxy (but not any partial derivative images).

CHAPTER 8

Data Products

The following data products related to this package are available for download from SLAC:

Filename	Size (Mb)	Description
OneDegSq.fits.gz	81.4	Input LSST DM galaxy catalog covering 1 sq.deg.
LSST_i.fits.gz	235.0	Simulated LSST i-band at full depth covering 1 chip
LSST_r.fits.gz	282.9	Simulated LSST r-band at full depth covering 1 chip
DES_i.fits.gz	77.4	Simulated DES i-band at full depth covering same area
DES_r.fits.gz	85.0	Simulated DES r-band at full depth covering same area
LSST_i_trimmed.fits.gz	27.3	Trimmed simulation results without per-object datacubes
LSST_r_trimmed.fits.gz	30.8	Trimmed simulation results without per-object datacubes
DES_i_trimmed.fits.gz	11.9	Trimmed simulation results without per-object datacubes
DES_r_trimmed.fits.gz	12.9	Trimmed simulation results without per-object datacubes

All files are in compressed FITS format and should be uncompressed after downloading:

```
gunzip *.fits.gz
```

The commands used to generate the *OneDegSq.fits* catalog are *described here*. The following commands were used to run the LSST and DES simulations:

```
nohup ./simulate.py --catalog-name OneDegSq.fits --survey-name LSST --filter-band i --output-name LSS
nohup ./simulate.py --catalog-name OneDegSq.fits --survey-name LSST --filter-band r --output-name LSS
nohup ./simulate.py --catalog-name OneDegSq.fits --survey-name DES --filter-band i --output-name DES_
nohup ./simulate.py --catalog-name OneDegSq.fits --survey-name DES --filter-band r --output-name DES_
```

The resulting FITS files were then trimmed to remove the individual object datacubes using:

```
import astropy.io.fits
for name in ('LSST_i','LSST_r','DES_i','DES_r'):
        hdus = astropy.io.fits.open(name+'.fits')
        del hdus[2:]
        hdus.writeto(name+'_trimmed.fits')
        hdus.close()
```

CHAPTER 9

IPython Notebooks

The following notebooks are included in the *notebooks/* directory and can be viewed online:

- GalaxyCatalogPlots: Plots of galaxy catalog quantities.
- OutputCatalogPlots: Plots of simulation output catalog quantities.
- ShearEstimatorPlots: Plots related to shear estimation.
- Figures: Figures for documenting the package.
- Source Extractor Comparisons: Comparisons with Source Extractor object detection.
- BiasEstimationPlots: Plots related to bias estimation.

You can also edit and run these notebooks locally if you have jupyter installed following these instructions.

CHAPTER 10

To Do List

- Formatted survey options printout.
- Use latest fitsio for writing simulation output.
- Use better 8-char names and add comments to FITS headers.
- Is the optical PSF making any difference? How much does it slow down simulate?
- Implement a 'exposure-time calculator' to calculate quantities for a profile specified on the command line (idea from Josh Meyers).
- Include a prior on dilation scale parameter for error estimation?
- Add ra,dec pointing and WCS to FITS primary HDU written by simulate.
- More flexible handling of the ra,dec catalog query window (we currently assume the query is centered near the origin).
- python 2-to-3 migration: from __future__ import absolute_import, division, print_function, unicode_literals
- Add zscale color gradient and angular scale legend options to display.py
- Add option to specify display view bounds independently of selected objects.
- Increase simulated area (from 1 LSST chip) for data products?
- Add option to add arbitrary fraction of 2*pi to all position angles (for ring tests).

10.1 Longer-Term Projects

- Validate galaxy input catalog quantities against existing datasets.
- Add stars from some catalog and estimate their contributions to blending statistics.
- Integrate over survey observing conditions (seeing, sky, ...) using operations simulator output.
- Test toy deblender performance on simulated images (Josh Myers).
- Compare sextractor object detection with snr_grp detection threshold (Mandeep Gill).
- Compare DM pipeline object detection with snr_grp detection threshold.
- Use PhoSim to generate realistic bright-star templates that could be used in GalSim (Chris Walter).
- Compare GalSim and PhoSim images generated for the same catalog footprint (Matt Wiesner).
- Characterize galaxies that look like stars (Alex Abate + Elliott Cheu + Arizona undergrad).

- Compare CFHTLS predictions with published results (Dominique Boutigny).
- Compare DES predictions with actual DES imaging.
- Add survey parameters for HSC to Survey class and compare predictions with actual HSC imaging (Rachel Mandelbaum).
- Include CatSim SEDs to implement band-dependent bulge/disk ratios and synthesize non-LSST magnitudes.
- Study dependence of full-depth neff on assumed mean sky noise and seeing.

CHAPTER 11

Information for Developers

11.1 Build the documentation

To build a local copy of the HTML documentation, use:

```
cd .../descwl/docs
make html
```

To view the most recent build of the HTML documentation, point your browser to *.../docs/_build/html/index.html*

To create a tarball snapshot *.../descwl/docs/_build/descwl.tgz* that can be installed on a web server, use:

```
cd .../descwl/docs/_build/html
tar -zcf ../descwl.tgz .
```

11.2 Add a new package module

Create a new file *descwl/xxx.py* for module *descwl.xxx* with an initial descriptive docstring.

Add the line *import xxx* to *descwl/__init__.py*.

Add the line *descwl.xxx* to the list of submodules in *docs/src/descwl.rst*.

Create a new documentation file *docs/src/descwl.xxx.rst* containing (replace *xxx* in two places):

```
descwl.xxx module
=================

.. automodule:: descwl.xxx
    :members:
    :undoc-members:
    :show-inheritance:
```

11.3 Update the version

Update the *version* and *release* values in the sphinx configuration file *docs/conf.py*, then commit, tag, and push the new version, e.g.:

```
git tag -l v0.2
```

11.4 Profiling

The *simulate* program has a *–memory-trace* option that displays the memory usage at frequent checkpoints during the execution, based on the `descwl.trace` module.

Profile CPU usage with the standard recipe, for example:

```
python -m cProfile -o profile.out ./simulate.py --catalog-name OneDegSq.fits --output-name profile
```

Examine the results using, for example:

```
import pstats
p = pstats.Stats('profile.out')
p.sort_stats('time').print_stats(10)
```

Here is a sample profiling output using the above examples:

```
Sat Jan  3 12:16:55 2015    profile.out

         217334792 function calls (216127873 primitive calls) in 813.478 seconds

   Ordered by: internal time
   List reduced from 2518 to 10 due to restriction <10>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    50963  520.836    0.010  569.392    0.011 /Users/david/anaconda/lib/python2.7/site-packages/galsi
        1   41.916   41.916   48.871   48.871 ./descwl/analysis.py:142(finalize)
   593789   40.541    0.000   42.049    0.000 /Users/david/anaconda/lib/python2.7/site-packages/galsi
   281014   15.681    0.000   15.681    0.000 {method 'reduce' of 'numpy.ufunc' objects}
    51547   15.326    0.000  611.955    0.012 ./descwl/render.py:71(render_galaxy)
  2946402    9.266    0.000   37.495    0.000 /Users/david/anaconda/lib/python2.7/site-packages/astro
 41011977    8.557    0.000    8.977    0.000 {isinstance}
  3457423    8.052    0.000   13.553    0.000 /Users/david/anaconda/lib/python2.7/site-packages/astro
  1178324    4.791    0.000   21.044    0.000 /Users/david/anaconda/lib/python2.7/site-packages/astro
   367098    4.396    0.000    5.389    0.000 /Users/david/anaconda/lib/python2.7/site-packages/galsi
```

CHAPTER 12

Revision History

12.1 v0.4

- Implement Fisher-matrix bias calculations.
- Add quickstart and FAQ documentation pages.
- Update notebook links (nbviewer -> github) and docs (ipython -> jupyter).

12.2 v0.3

- Add tutorial notebook.
- Add HSM size and shape analysis for each galaxy (ignoring overlaps).

12.3 v0.2

- Refactor code and add sphinx documentation.
- Rename *galsimcat.py* as *simulate.py* and *lsst2wl.py* as *dbquery.py*.
- Add new programs *display.py*, *fisher.py*.
- Fix catalog half-light radius bug: use sqrt(a*b) instead of catalog HLR value.
- Include AGN component when present in the catalog.
- Use Airy optical model instead of increasing atmospheric FWHM by 0.4 arcsec.
- Implement new blending figures of merit.
- Version used for DESC highlight project data products and paper draft v0.2

12.4 v0.1

- Version used to prepare results for the Dec 2013 LSST DESC meeting in Pittsburgh.

CHAPTER 13

Modules API Reference

13.1 descwl package

13.1.1 Submodules

descwl.catalog module

Load source parameters from catalog files.

There is a separate catalog page with details on the expected catalog contents and formatting.

class descwl.catalog.**Reader**(*catalog_name*, *ra_center=0.0*, *dec_center=0.0*, *only_id=[]*, *skip_id=[]*)
Bases: object

Read source parameters from a catalog to simulate an image.

The entire catalog is read when the object is constructed, independently of the only_id and skip_id parameter values. Note that constructor parameter defaults are specified in the add_args() function.

Details on the catalog contents and format are documented on the catalog page.

Parameters

- **catalog_name** (*str*) – Name of catalog file, which must exist and be readable. The catalog will be read as a FITS table if catalog_name ends with '.fits'. Otherwise, it will be read as an ASCII file.
- **ra_center** (*float*) – Right ascension of image center in degrees.
- **dec_center** (*float*) – Declination of image center in degrees.
- **only_id** (*array*) – Only read ids in this array of integer ids.
- **skip_id** (*array*) – Skip ids in this array of integer ids.

Raises RuntimeError – Missing required catalog_name arg.

static add_args(*parser*)
Add command-line arguments for constructing a new *Reader*.

The added arguments are our constructor parameters with '_' replaced by '-' in the names. Note that constructor parameter defaults are specified here rather than in the constructor, so that they are included in command-line help.

Parameters **parser** (*argparse.ArgumentParser*) – Arguments will be added to this parser object using its add_argument method.

classmethod **from_args**(*args*)
Create a new *Reader* object from a set of arguments.

Parameters **args** (*object*) – A set of arguments accessed as a dict using the built-in vars() function. Any extra arguments beyond those defined in *add_args()* will be silently ignored.

Returns A newly constructed Reader object.

Return type *Reader*

potentially_visible_entries(*survey*, *render_options*)
Iterate over potentially visible catalog entries.

Potential visibility is determined by the combined survey parameters and rendering options and implemented so that all actually visible entries are included. Returned entries might not be actually visible, for example, if they are too faint. If only_id has any entries, then only the specified ids will be considered. Otherwise, all ids are considered. Any ids in skip_id will be silently ignored.

Filtering on (ra,dec) to determine visibility is currently not very robust and assumes that ra_center is close to zero and that subtracting 360 from any ra > 180 is sufficient for interval tests.

Parameters

- **survey** (*descwl.survey.Survey*) – Survey parameters used to determine which entries are visible.
- **render_options** (descwl.render.Options) – Rendering options used to determine which entries are visible.

Yields *tuple* –

Tuple (entry,dx,dy) of the current catalog entry (astropy.table.Row) and the x,y offsets (float) of this entry's centroid from the image center in pixels.

descwl.survey module

Manage the parameters that define a simulated survey's camera design and observing conditions.

class descwl.survey.**Survey**(***args*)
Bases: object

Survey camera and observing parameters.

No default values are assigned to constructor args since these are handled at run time based on a requested (survey_name,filter_band) combination using *get_defaults()*.

Parameters

- **survey_name** (*str*) – Use default camera and observing parameters appropriate for this survey.
- **filter_band** (*str*) – LSST imaging band to simulate. Must be one of 'u','g','r','i','z','y'.
- **image_width** (*int*) – Simulated camera image width in pixels.
- **image_height** (*int*) – Simulated camera image height in pixels.
- **pixel_scale** (*float*) – Simulated camera pixel scale in arcseconds per pixel.
- **exposure_time** (*float*) – Simulated camera total exposure time seconds.

- **zero_point** (*float*) – Simulated camera zero point in electrons per second at 24th magnitude.
- **mirror_diameter** (*float*) – Size of the primary mirror's clear aperture in meters to use for the optical PSF, or zero if no optical PSF should be simulated.
- **effective_area** (*float*) – Effective total light collecting area in square meters. Used to determine the obscuration fraction in the simulated optical PSF. Ignored if mirror_diameter is zero.
- **zenith_psf_fwhm** (*float*) – FWHM of the atmospheric PSF at zenith in arcseconds.
- **atmospheric_psf_beta** (*float*) – Moffat beta parameter of the atmospheric PSF, or use a Kolmogorov PSF if beta <= 0.
- **atmospheric_psf_e1** (*float*) – Atmospheric ellipticity component e1 (+) with |e| = (a-b)/(a+b).
- **atmospheric_psf_e2** (*float*) – Atmospheric ellipticity component e2 (x) with |e| = (a-b)/(a+b).
- **sky_brightness** (*float*) – Sky brightness in mags/sq.arcsec during the observation.
- **airmass** (*float*) – Optical path length through the atmosphere relative to the zenith path length.
- **extinction** (*float*) – Exponential exctinction coefficient for atmospheric absorption.
- **cosmic_shear_g1** (*float*) – Cosmic shear ellipticity component g1 (+) with |g| = (a-b)/(a+b).
- **cosmic_shear_g2** (*float*) – Cosmic shear ellipticity component g2 (x) with |g| = (a-b)/(a+b).

Raises `RuntimeError` – Missing or extra arguments provided or unable to calculate PSF size.

static add_args(*parser*)

Add command-line arguments for constructing a new *Survey*.

The added arguments are our constructor parameters with '_' replaced by '-' in the names. No default values are assigned to the added args since these are handled at run time based on a requested (survey_name,filter_band) combination using *get_defaults()*.

Parameters **parser** (*argparse.ArgumentParser*) – Arguments will be added to this parser object using its add_argument method.

description()

Describe the survey we simulate.

Returns Description of the camera design and observing conditions we simulate.

Return type str

classmethod from_args(*args*)

Create a new *Survey* object from a set of arguments.

Parameters **args** (*object*) – A set of arguments accessed as a `dict` using the built-in `vars()` function. The argument set must include those defined in *add_args()*. Two additional arguments are also required: survey_name and filter_band. These establish the constructor parameter defaults via :func:get_defaults. Any extra arguments beyond those defined in *add_args()*, survey_name, and filter_band will be silently ignored.

Returns A newly constructed Survey object.

Return type *Survey*

Raises RuntimeError – Defaults not yet defined for requested combination.

static get_defaults(*survey_name*, *filter_band*)
Get survey camera and observing parameter default values.

Parameters

- **survey_name** (*str*) – Use defaults for this survey. Case is significant.
- **filter_band** (*str*) – Use defaults for this filter band. Case is significant.

Returns Dictionary of parameter (name,value) pairs.

Return type dict

Raises RuntimeError – Defaults not yet defined for requested combination.

get_flux(*ab_magnitude*)
Convert source magnitude to flux.

The calculation includes the effects of atmospheric extinction.

Parameters **ab_magnitude** (*float*) – AB magnitude of source.

Returns Flux in detected electrons.

Return type float

get_image_coordinates(*dx_arcsecs*, *dy_arcsecs*)
Convert a physical offset from the image center into image coordinates.

Parameters

- **dx_arcsecs** (*float*) – Offset from the image center in arcseconds.
- **dy_arcsecs** (*float*) – Offset from the image center in arcseconds.

Returns

Corresponding floating-point image coordinates (x_pixels,y_pixels) whose math.floor() value gives pixel indices and whose...

Return type tuple

static print_defaults()
Print parameters for all available (survey,filter band) combinations.

descwl.model module

Model astronomical sources.

class descwl.model.**Galaxy**(*identifier*, *redshift*, *ab_magnitude*, *ri_color*, *cosmic_shear_g1*, *cosmic_shear_g2*, *dx_arcsecs*, *dy_arcsecs*, *beta_radians*, *disk_flux*, *disk_hlr_arcsecs*, *disk_q*, *bulge_flux*, *bulge_hlr_arcsecs*, *bulge_q*, *agn_flux*)
Bases: object

Source model for a galaxy.

Galaxies are modeled using up to three components: a disk (Sersic n=1), a bulge (Sersic n=4), and an AGN (PSF-like). Not all components are required. All components are assumed to have the same centroid and the extended (disk,bulge) components are assumed to have the same position angle.

Parameters

- **identifier** (*int*) – Unique integer identifier for this galaxy in the source catalog.
- **redshift** (*float*) – Catalog redshift of this galaxy.
- **ab_magnitude** (*float*) – Catalog AB magnitude of this galaxy in the filter band being simulated.
- **ri_color** (*float*) – Catalog source color calculated as (r-i) AB magnitude difference.
- **cosmic_shear_g1** (*float*) – Cosmic shear ellipticity component g1 (+) with |g| = (a-b)/(a+b).
- **cosmic_shear_g2** (*float*) – Cosmic shear ellipticity component g2 (x) with |g| = (a-b)/(a+b).
- **dx_arcsecs** (*float*) – Horizontal offset of catalog entry's centroid from image center in arcseconds.
- **dy_arcsecs** (*float*) – Vertical offset of catalog entry's centroid from image center in arcseconds.
- **beta_radians** (*float*) – Position angle beta of Sersic components in radians, measured anti-clockwise from the positive x-axis. Ignored if disk_flux and bulge_flux are both zero.
- **disk_flux** (*float*) – Total flux in detected electrons of Sersic n=1 component.
- **disk_hlr_arcsecs** (*float*) – Half-light radius sqrt(a*b) of circularized 50% isophote for Sersic n=1 component, in arcseconds. Ignored if disk_flux is zero.
- **disk_q** (*float*) – Ratio b/a of 50% isophote semi-minor (b) to semi-major (a) axis lengths for Sersic n=1 component. Ignored if disk_flux is zero.
- **bulge_flux** (*float*) – Total flux in detected electrons of Sersic n=4 component.
- **bulge_hlr_arcsecs** (*float*) – Half-light radius sqrt(a*b) of circularized 50% isophote for Sersic n=4 component, in arcseconds. Ignored if bulge_flux is zero.
- **bulge_q** (*float*) – Ratio b/a of 50% isophote semi-minor (b) to semi-major (a) axis lengths for Sersic n=4 component. Ignored if bulge_flux is zero.
- **agn_flux** (*float*) – Total flux in detected electrons of PSF-like component.

get_transformed_model(*dx=0.0*, *dy=0.0*, *ds=0.0*, *dg1=0.0*, *dg2=0.0*)

Apply transforms to our model.

The nominal model returned by *get_transformed_model()* is available via the *model* attribute.

Parameters

- **dx** (*float*) – Amount to shift centroid in x, in arcseconds.
- **dy** (*float*) – Amount to shift centroid in y, in arcseconds.
- **ds** (*float*) – Relative amount to scale the galaxy profile in the radial direction while conserving flux, before applying shear or convolving with the PSF.
- **dg1** (*float*) – Amount to adjust the + shear applied to the galaxy profile, with |g| = (a-b)/(a+b), before convolving with the PSF.
- **dg2** (*float*) – Amount to adjust the x shear applied to the galaxy profile, with |g| = (a-b)/(a+b), before convolving with the PSF.

Returns

New model constructed using our source profile with the requested transforms applied.

Return type galsim.GSObject

class descwl.model.**GalaxyBuilder**(*survey*, *no_disk*, *no_bulge*, *no_agn*, *verbose_model*)
Bases: object

Build galaxy source models.

Parameters

- **survey** (descwl.survey.Survey) – Survey to use for flux normalization and cosmic shear.
- **no_disk** (*bool*) – Ignore any Sersic n=1 component in the model if it is present in the catalog.
- **no_bulge** (*bool*) – Ignore any Sersic n=4 component in the model if it is present in the catalog.
- **no_agn** (*bool*) – Ignore any PSF-like component in the model if it is present in the catalog.
- **verbose_model** (*bool*) – Provide verbose output from model building process.

static add_args(*parser*)
Add command-line arguments for constructing a new *GalaxyBuilder*.

The added arguments are our constructor parameters with '_' replaced by '-' in the names.

Parameters **parser** (*argparse.ArgumentParser*) – Arguments will be added to this parser object using its add_argument method.

classmethod from_args(*survey*, *args*)
Create a new *GalaxyBuilder* object from a set of arguments.

Parameters

- **survey** (descwl.survey.Survey) – Survey to build source models for.
- **args** (*object*) – A set of arguments accessed as a dict using the built-in vars() function. Any extra arguments beyond those defined in *add_args()* will be silently ignored.

Returns A newly constructed Reader object.

Return type *GalaxyBuilder*

from_catalog(*entry*, *dx_arcsecs*, *dy_arcsecs*, *filter_band*)
Build a :class:Galaxy object from a catalog entry.

Fluxes are distributed between the three possible components (disk,bulge,AGN) assuming that each component has the same spectral energy distribution, so that the resulting proportions are independent of the filter band.

Parameters

- **entry** (*astropy.table.Row*) – A single row from a galaxy *descwl.catalog*.
- **dx_arcsecs** (*float*) – Horizontal offset of catalog entry's centroid from image center in arcseconds.
- **dy_arcsecs** (*float*) – Vertical offset of catalog entry's centroid from image center in arcseconds.
- **filter_band** (*str*) – The LSST filter band to use for calculating flux, which must be one of 'u','g','r','i','z','y'.

Returns A newly created galaxy source model.

Return type *Galaxy*

Raises

- *SourceNotVisible* – All of the galaxy's components are being ignored.
- RuntimeError – Catalog entry is missing AB flux value in requested filter band.

exception descwl.model.**SourceNotVisible**
Bases: exceptions.Exception

Custom exception to indicate that a source has no visible model components.

descwl.model.**moments_size_and_shape**(*Q*)
Calculate size and shape parameters from a second-moment tensor.

If the input is an array of second-moment tensors, the calculation is vectorized and returns a tuple of output arrays with the same leading dimensions (...).

Parameters **Q** (*numpy.ndarray*) – Array of shape (...,2,2) containing second-moment tensors, which are assumed to be symmetric (only the [0,1] component is used).

Returns

Tuple (sigma_m,sigma_p,a,b,beta,e1,e2) of numpy.ndarray objects with shape (...). Refer to *Analysis Results* for details on how each of these vectors is defined.

Return type tuple

descwl.model.**sersic_second_moments**(*n*, *hlr*, *q*, *beta*)
Calculate the second-moment tensor of a sheared Sersic radial profile.

Parameters

- **n** (*int*) – Sersic index of radial profile. Only n = 1 and n = 4 are supported.
- **hlr** (*float*) – Radius of 50% isophote before shearing, in arcseconds.
- **q** (*float*) – Ratio b/a of Sersic isophotes after shearing.
- **beta** (*float*) – Position angle of sheared isophotes in radians, measured anti-clockwise from the positive x-axis.

Returns

Array of shape (2,2) with values of the second-moments tensor matrix, in units of square arcseconds.

Return type numpy.ndarray

Raises RuntimeError – Invalid Sersic index n.

descwl.model.**sheared_second_moments**(*Q*, *g1*, *g2*)
Apply shear to a second moments matrix.

The shear M = ((1-g1,-g2),(-g2,1+g1)) is applied to each Q by calculating Q' = (M**-1).Q.(M**-1)^t where M**-1 = ((1+g1,g2),(g2,1-g1))/|M|. If the input is an array of second-moment tensors, the calculation is vectorized and returns a tuple of output arrays with the same leading dimensions (...).

Parameters

- **Q** (*numpy.ndarray*) – Array of shape (...,2,2) containing second-moment tensors, which are assumed to be symmetric.
- **g1** (*float*) – Shear ellipticity component g1 (+) with |g| = (a-b)/(a+b).
- **g2** (*float*) – Shear ellipticity component g2 (x) with |g| = (a-b)/(a+b).

Returns

Array with the same shape as the input Q with the shear (g1,g2) applied to each 2x2 second-moments submatrix.

Return type numpy.ndarray

descwl.render module

Render source models as simulated survey observations.

class descwl.render.**Engine**(*survey*, *min_snr*, *truncate_radius*, *no_margin*, *verbose_render*)
Bases: object

Rendering engine to simulate survey observations.

Any pixels outside of the truncation radius or below the minimum S/N cut will have their flux set to zero in the rendered image. As a result the total rendered flux may be below the total model flux.

Parameters

- **survey** (descwl.survey.Survey) – Survey that rendered images will simulate.
- **min_snr** (*float*) – Simulate signals from individual sources down to this S/N threshold, where the signal N is calculated for the full exposure time and the noise N is set by the expected fluctuations in the sky background during a full exposure.
- **truncate_radius** (*float*) – All extended sources are truncated at this radius in arcseconds.
- **no_margin** (*bool*) – Do not simulate the tails of objects just outside the field.
- **verbose_render** (*bool*) – Provide verbose output on rendering process.

static add_args(*parser*)
Add command-line arguments for constructing a new *Engine*.

The added arguments are our constructor parameters with '_' replaced by '-' in the names. Note that constructor parameter defaults are specified here rather than in the constructor, so that they are included in command-line help.

Parameters **parser** (*argparse.ArgumentParser*) – Arguments will be added to this parser object using its add_argument method.

description()
Describe our rendering configuration.

Returns

Description of the rendering configuration that we be used to simulate the survey.

Return type str

classmethod from_args(*survey*, *args*)
Create a new *Engine* object from a set of arguments.

Parameters

- **survey** (descwl.survey.Survey) – Survey that rendered images will simulate.
- **args** (*object*) – A set of arguments accessed as a dict using the built-in vars() function. Any extra arguments beyond those defined in *add_args()* will be silently ignored.

Returns A newly constructed Engine object.

Return type *Engine*

render_galaxy(*galaxy*, *no_partials=False*, *calculate_bias=False*)
Render a galaxy model for a simulated survey.

Parameters

- **galaxy** (descwl.model.Galaxy) – Model of the galaxy to render.
- **no_partials** (*bool*) – Do not calculate partial derivative images.

Returns

(stamps,bounds) **where *stamps* is a numpy.ndarray of shape** (nstamp,width,height) pixel values that represents nstamp postage-stamp images with the same dimensions (width,height) determined by the rendering options provided. The returned *bounds* give the position of these stamps within the full simulated survey image as a *galsim.BoundsI* object. Note that these bounds might extend beyond the survey image, but will always have some overlap where the source is above threshold.

Return type tuple

Raises *SourceNotVisible* – Galaxy has no pixels above threshold that are visible in the simulated survey.

class descwl.render.**GalaxyRenderer**(*galaxy*, *stamp*, *survey*)
Bases: object

Rendering engine for a single galaxy.

Parameters

- **galaxy** (descwl.model.Galaxy) – Model of the galaxy we will render.
- **stamp** (*galsim.Image*) – Previously rendered image of this galaxy with no transforms applied. Zero pixels in this image are used to define a mask where all subsequently rendered images will have pixels values set to zero. The input stamp is copied and not modified. The stamp bounds determine the region of the full survey image that we will render into.
- **survey** (descwl.survey.Survey) – Survey that rendered images will simulate. Used to determine the mapping from survey positions to stamp positions and specify the PSF to use.

draw(*df=0.0*, *dx=0.0*, *dy=0.0*, *ds=0.0*, *dg1=0.0*, *dg2=0.0*)
Draw the galaxy with the specified transforms applied.

We use descwl.galaxy.Galaxy.get_transformed_model() to apply all transforms except for *df*, which we implement internally using rescaling. The transformed model is convolved with the survey PSF, rendered into the stamp specified in our constructor, and masked (zero pixels in the untransformed rendering are forced to zero). Repeated calls with the same transform parameters return a cached image immediately. The same is true if only the *df* parameter is changed from the last call.

Parameters

- **df** (*float*) – Relative amount to scale the total galaxy flux.
- **dx** (*float*) – Amount to shift centroid in x, in arcseconds.
- **dy** (*float*) – Amount to shift centroid in y, in arcseconds.
- **ds** (*float*) – Relative amount to scale the galaxy profile in the radial direction while conserving flux, before applying shear or convolving with the PSF.
- **dg1** (*float*) – Amount to adjust the + shear applied to the galaxy profile, with |g| = (a-b)/(a+b), before convolving with the PSF.

- **dg2** (*float*) – Amount to adjust the x shear applied to the galaxy profile, with |g| = (a-b)/(a+b), before convolving with the PSF.

Returns Rendering of the transformed galaxy.

Return type galsim.Image

exception descwl.render.**SourceNotVisible**
Bases: exceptions.Exception

Custom exception to indicate that a source has no visible pixels above threshold.

descwl.analysis module

Perform weak-lensing analysis of simulated sources.

class descwl.analysis.**OverlapAnalyzer**(*survey*, *no_hsm*, *alpha=1*)
Bases: object

Analyze impact of overlapping sources on weak lensing.

Parameters **survey** (descwl.survey.Survey) – Simulated survey to describe with FITS header keywords.

static add_args(*parser*)
Add command-line arguments for constructing a new Reader.

The added arguments are our constructor parameters with '_' replaced by '-' in the names. Note that constructor parameter defaults are specified here rather than in the constructor, so that they are included in command-line help.

Parameters **parser** (*argparse.ArgumentParser*) – Arguments will be added to this parser object using its add_argument method.

add_galaxy(*model*, *stamps*, *bounds*)
Add one galaxy to be analyzed.

Parameters

- **model** (*descwl.model.Galaxy*) – The galaxy model used for rendering.
- **stamps** (numpy.ndarray) – Array of shape (nstamp,width,height) containing pixel values for nstamp stamps of dimensions (width,height).
- **bounds** (*galsim.BoundsI*) – Bounds of the stamps in the full simulated survey image.

finalize(*verbose*, *trace*, *calculate_bias*)
Finalize analysis of all added galaxies.

Parameters

- **verbose** (*bool*) – Print a summary of analysis results.
- **trace** (*callable*) – Function to call for tracing resource usage. Will be called with a brief str description of each checkpoint.

Returns Overlap analysis results.

Return type *OverlapResults*

fit_galaxies(*indices*, *observed_image*, *fixed_parameters=None*)
Simultaneously fit a set of galaxy parameters to an observed image.

Fits are performed on noise-free images, so there are no meaningful errors to report and only best-fit parameter values are returned. This method is intended to support systematics studies where shifts in best-fit parameters are the quantities of interest. See the *descwl.render.GalaxyRenderer.draw()* method for details on how the fit parameters are defined and how each galaxy model is rendered as these parameters are varied.

Parameters

- **indices** (*iterable*) – List of num_galaxies integer galaxy indices to include in the fit. Index values correspond to the order in which galaxies were added using *add_galaxy()*.
- **observed_image** (*galsim.Image*) – A GalSim image that defines the observed pixels that will be fit and the region into which galaxy models will be renderered.
- **fixed_parameters** (*dict*) – An optional dictionary of parameter values to fix in the fit, or None if all parameters should be floating.

Returns

Array with shape (num_galaxies,6) containing the best-fit values of the following six parameters: df,dx,dy,ds,dg1,dg2. See the *descwl.render.GalaxyRenderer.draw()* method for details on how these parameters are defined.

Return type numpy.ndarray

Raises `RuntimeError` – Invalid galaxy index or fit did not find a minimum.

classmethod from_args(*args*)

Create a new `Reader` object from a set of arguments.

Parameters **args** (*object*) – A set of arguments accessed as a `dict` using the built-in `vars()` function. Any extra arguments beyond those defined in *add_args()* will be silently ignored.

Returns A newly constructed Reader object.

Return type `Reader`

class descwl.analysis.**OverlapResults**(*survey*, *table*, *stamps*, *bounds*, *num_slices*)

Bases: `object`

Results of analyzing effects of overlapping sources on weak lensing.

Results are normally obtained by calling *OverlapAnalyzer.finalize()* after simulating an image or using *descwl.output.Reader()* to load saved simulation results.

Indices returned by the *find* methods below can be used to lookup analysis results via *table[index]* and the corresponding simulated datacube using *stamps[index]* and *bounds[index]*.

Parameters

- **survey** (`descwl.survey.Survey`) – Simulated survey that results are based on.
- **table** (*astropy.table.Table*) – Table of analysis results with one row per galaxy.
- **stamps** (*array*) – Array of `numpy.ndarray` postage-stamp datacubes. Might be None.
- **bounds** (*array*) – Array of galsim.BoundsI objects giving pixel coordinates of each datacube in the full simulated image. Might be None.
- **num_slices** (*int*) – Number of datacube slices for each stamp. Ignored if stamps is None.

Raises `RuntimeError` – Image datacubes have unexpected number of slices.

add_noise(*noise_seed*)
Add Poisson noise to the simulated survey image.

Parameters **noise_seed** (*int*) – Random seed to use.

Raises `RuntimeError` – Noise has already been added.

get_bias(*selected*, *covariance*)
Return bias of the 6 parameters in vector form.

The bias is obtained from contracting the bias tensor with the appropiate covariance matrix elements.

Parameters

- **selected** (*iterable*) – Array of integer indices for the sources to include in the calculated matrices.
- **covariance** (*array*) – An array containing the covariance matrix of the selected galaxies.

Returns

bias which is a vector with dimensions (nbias) containing the biases of the selected galaxies.

Return type array

get_bias_tensor(*selected*, *covariance*)
Return bias tensor from the selected galaxies.

Uses the function get_bias_tensor_images() and then contracts these images to obtain the actual bias tensor.

Parameters

- **selected** (*iterable*) – Array of integer indices for the sources to include in the calculated matrices.
- **covariance** (*array*) – An array containing the covariance matrix of the selected galaxies.

Returns

bias_tensor with dimensions (ntensor,ntensor,npar) containing bias_tensor elements.

Return type array

get_bias_tensor_images(*index1*, *index2*, *background*)
Return bias tensor images for a pair of galaxies.

The bias tensor images are required in the calculation of the bias of parameters of the galaxies. Bias tensor images are derived from the first partials and second partials of the six parameters.

Parameters

- **index1** (*int*) – Index of the first galaxy to use.
- **index2** (*int*) – Index of the second galaxy to use, which might the same as index1.
- **background** (*galsim.Image*) – Background image that combines all sources that overlap the two galaxies and completely contains them.

Returns

Tuple (images,overlap) where images is a `numpy.ndarray` with shape (npar,npar,npar,height,width), where npar=6 and (height,width) are the dimensions of the overlap between the two galaxies, and overlap gives the bounding box of the overlap in the full survey image. Returns None,None if the two galaxies do not overlap.

Return type tuple

Raises `RuntimeError` – Invalid index1 or index2, or galaxies are not contained with the background image, or no partial derivative images are available.

`get_fisher_images`(*index1*, *index2*, *background*)

Return Fisher-matrix images for a pair of galaxies.

Fisher matrix images are derived from the partial derivatives with respect to six parameters: total flux in detected electrons, centroid positions in x and y in arcseconds, flux-preserving radial scale (dimensionless), and shear g1,g2 with |g| = (a-b)/(a+b). Use the *fisher* program to display Fisher matrix images.

Parameters

- **`index1`** (*`int`*) – Index of the first galaxy to use.
- **`index2`** (*`int`*) – Index of the second galaxy to use, which might the same as index1.
- **`background`** (*`galsim.Image`*) – Background image that combines all sources that overlap the two galaxies and completely contains them.

Returns

Tuple (images,overlap) where images is a `numpy.ndarray` with shape (npar,npar,height,width), where npar=6 and (height,width) are the dimensions of the overlap between the two galaxies, and overlap gives the bounding box of the overlap in the full survey image. Returns None,None if the two galaxies do not overlap.

Return type tuple

Raises `RuntimeError` – Invalid index1 or index2, or galaxies are not contained with the background image, or no partial derivative images are available.

`get_matrices`(*selected*)

Return matrices derived the from Fisher-matrix images for a set of sources.

If the Fisher matrix is not invertible or any variances are <= 0, we will drop the selected source with the lowest value of snr_iso and try again. This procedure is iterated until we get a valid covariance matrix. Matrix elements for all parameters of any sources that get dropped by this procedure will be set to zero and variances will be set to np.inf so that 1/np.sqrt(variance) = 0. Invalid covariances are generally associated with sources that are barely above the pixel SNR threshold, so this procedure should normally provide sensible values for the largest possible subset of the input selected sources. Use the *fisher* program to visualize matrix elements and to further study examples of invalid covariances.

Parameters **`selected`** (*`iterable`*) – Array of integer indices for the sources to include in the calculated matrices.

Returns

Tuple *(fisher,covariance,variance,correlation)* of `numpy.ndarray` where *variance* has shape (npar,) and all other arrays are symmetric with shape (npar,npar), where npar = 6*len(selected). Matrix elements will be zero for any parameters associated with dropped sources, as described above.

Return type tuple

`get_stamp`(*index*, *datacube_index=0*)

Return the simulated postage stamp for a single galaxy.

Parameters

- **index** (*int*) – Index of the requested galaxy in our results. Use meth:*find_galaxy* to get the index for an identifier.
- **datacube_index** (*int*) – Which slice of the datacube to return.

Returns

A GalSim image for the requested galaxy. Note that the return value is a read-only view of our internal stamps array.

Return type galsim.ImageView

Raises RuntimeError – No such galaxy in the results.

get_subimage(*indices*, *datacube_index=0*)

Return simulated subimage of a set of objects.

Parameters **indices** (*iterable*) – Indices of the objects to include in the subimage. Our *select()* method returns a suitable argument to use here by default (format = 'index').

Returns Image of the selected objects or None if indices is empty.

Return type galsim.Image

Raises RuntimeError – An index is out of range.

match_sextractor(*catalog_name*, *column_name='match'*)

Match detected objects to simulated sources.

Parameters

- **catalog_name** (*str*) – Name of an ASCII catalog in SExtractor-compatible format, and containing X_IMAGE, Y_IMAGE columns and *num_found* rows.
- **column_name** (*str*) – Name of a column in our *table* data member to fill with the detected object index matched to each simulated source, or -1 if no match is found. Overwrites any exisiting column or creates a new one. Does nothing if column_name is None or ''.

Returns

Tuple *detected,matched,indices,distance* where *detected* is the detected catalog as a astropy.table.Table, *matched* is an array of *num_found* booleans indicating whether each object was matched with a simulated object, *indices* is an array of *num_matched = np.count_nonzero(matched)* integers giving row numbers in our *table* attribute for each simulated match, and *distance* is an array of *num_matched* separation distances in arcseconds between matched and simulated objects.

Return type tuple

select(**selectors*, ***options*)

Select objects.

This function is implemented using eval() so should only be used when the source of the selector strings is trusted.

Parameters

- **selectors** (*str,...*) – A sequence of one or more selection strings, each consisting of a boolean expression of column table names, e.g. 'snr_iso > 5'. The special strings 'ALL' and 'NONE' do what you would expect.

- **mode** (*str*) – The string 'and' or 'or' to specify how multiple selectors should be combined.
- **format** (*str*) – The string 'mask' or 'index' to specify the format of the returned array. A mask is an array of booleans with one entry per source. The alternative index array of integers lists the selected indices in increasing order and might be empty. The mask format is useful for performing your own logical operations on the returned array. The index format is useful for iterating over the selected objects. Both formats can be used to slice the catalog table to extract only the selected rows.

Returns

A numpy array of booleans or integers, depending on the value of the format input argument.

Return type numpy.ndarray

Raises RuntimeError – A selector uses an undefined variable name, the method was passed an unexpected mode or format string, or these results have no catalog table.

slice_labels = ['dflux', 'dx', 'dy', 'ds', 'dg1', 'dg2']

descwl.analysis.**make_inv_positions**()

descwl.analysis.**make_positions**()

descwl.output module

Configure and handle simulation output.

There is a separate output page with details on what goes into the output and how it is formatted.

class descwl.output.**Reader**(*input_name*, *defer_stamp_loading=True*)

Bases: object

Simulation output reader.

The reader loads the file's contents into memory in the constructor and makes them available via a *results* data member of type *descwl.analysis.OverlapResults*. Loading of stamp datacubes can be defered until stamps are acutally accessed using the defer_stamp_loading argument below.

We use fitsio to read files since it performs significantly faster than astropy.io.fits for files with many (~100K) relatively small HDUs (although astropy.io.fits outperforms *fitsio* for writing these files).

Parameters

- **input_name** (*str*) – Base name of FITS output files to write. The ".fits" extension can be omitted.
- **defer_stamp_loading** (*bool*) – Defer the loading of stamp datacubes until they are actually accessed via descwl.OverlapResults.get_stamp() of our *results* data member. This speeds up the constructor substantially and reduces memory requirements when relatively few stamps are actually needed.

Raises RuntimeError – Unable to initialize FITS input file.

static add_args(*parser*)

Add command-line arguments for constructing a new *Reader*.

The added arguments are our constructor arguments with '_' replaced by '-' in the names. The defer_stamp_loading constructor argument is not added here.

Parameters **parser** (*argparse.ArgumentParser*) – Arguments will be added to this parser object using its add_argument method.

classmethod from_args(*defer_stamp_loading*, *args*)
Create a new *Reader* object from a set of arguments.

Parameters

- **defer_stamp_loading** (*bool*) – Defer the loading of stamp datacubes until they are actually accessed via `descwl.OverlapResults.get_stamp()` of our *results* data member. This speeds up the constructor substantially and reduces memory requirements when relatively few stamps are actually needed.
- **args** (*object*) – A set of arguments accessed as a `dict` using the built-in `vars()` function. Any extra arguments beyond those defined in *add_args()* will be silently ignored.

Returns A newly constructed Reader object.

Return type *Reader*

class `descwl.output.`**Writer**(*survey*, *output_name*, *no_stamps*, *no_catalog*, *output_no_clobber*)
Bases: `object`

Simulation output writer.

See the output page for details on the output contents and formatting.

We use `astropy.io.fits` to write files since it performs significanly faster than fitsio for files with many (~100K) relatively small HDUs (although *fitsio* outperforms `astropy.io.fits` for reading these files). See this issue for details and updates.

Parameters

- **survey** (`descwl.survey.Survey`) – Simulated survey to describe with FITS header keywords.
- **output_name** (*str*) – Base name of FITS output files to write. The ".fits" extension can be omitted.
- **no_catalog** (*bool*) – Do not save the results catalog.
- **no_stamps** (*bool*) – Do not save postage stamp datacubes for each source.
- **output_no_clobber** (*bool*) – Do not overwrite any existing output file with the same name.

Raises `RuntimeError` – Unable to initialize FITS output file.

static add_args(*parser*)
Add command-line arguments for constructing a new *Writer*.

The added arguments are our constructor parameters with '_' replaced by '-' in the names.

Parameters **parser** (*argparse.ArgumentParser*) – Arguments will be added to this parser object using its add_argument method.

description()
Describe our output configuration.

Returns Description of the the rendering configuration.

Return type str

finalize (*results*, *trace*)
Save analysis results and close the output file, if any.

This call builds the entire FITS file in memory then writes it to disk.

:param *descwl.analysis.OverlapResults*: Overlap analysis results. :param trace: Function to call for tracing resource usage. Will be

called with a brief `str` description of each checkpoint.

classmethod from_args (*survey*, *args*)
Create a new *Writer* object from a set of arguments.

Parameters

- **survey** (`descwl.survey.Survey`) – Simulated survey to describe with FITS header keywords.
- **args** (*object*) – A set of arguments accessed as a `dict` using the built-in `vars()` function. Any extra arguments beyond those defined in *add_args()* will be silently ignored.

Returns A newly constructed Reader object.

Return type *Writer*

descwl.trace module

Trace program resource usage.

class `descwl.trace.`**Memory** (*enabled*)
Bases: `object`

Trace memory usage for the current program.

Parameters **enabled** (*bool*) – Enable memory tracing.

13.1.2 Module contents

Weak lensing fast simulations and analysis for the LSST Dark Energy Science Collaboration.

This code was primarily developed to study the effects of overlapping sources on shear estimation, photometric redshift algorithms, and deblending algorithms.

d

O

P

R

S

W