Balrog

An astronomical imaging simulation d(a)emon[1]

for those who dig too deeply

and too greedily into

their data...

Eric Suchyta and Eric Huff

[1]Technically, the package is not a daemon. Please forgive our attempts at naming humor.

Contents

1. What Is Balrog?

1.1 Introduction

Balrog is a package of Python code, intended for use with astronomical imaging data. Strictly speaking, Balrog is a simulation tool. However, its ambition is derived from the aspiration to better characterize and understand real data. By performing a set of simulations, Balrog's intent is to allow observers to infer properties of their images by directly testing on the images themselves.

The core functionality driving Balrog's design is rather straightforward. Galaxies are simulated, trivially writing their simulated properties to a *truth catalog*. Noisy images of the galaxies are then inserted into real data. Source detection software runs over the image, whose measurements for the simulated galaxies can be directly compared to the truth catalog. Accordingly, one is able to answer the question of how the measured properties of the image are related to the true properties.

Instead of reinventing the wheel, the Balrog pipeline wraps around existing codes, well known within the Astronomy community. All galaxy simulations are implemented via GalSim[1] and source extraction and measurement occurs using SExtractor[2]. Balrog facilitates the ease of running these codes en masse over many images, filling in many of the bookkeeping steps in an automated way.

Since different users will have different needs, Balrog strives to be as flexible as possible. It includes a well defined framework capable of implementing a wide variety of simulation possibilites. The framework allows users to define their own arguments and functions to plug into Balrog when generating simulated galaxies.

An often overlooked feature of scientific software is its usability. Balrog has been written making our best attempts at user-friendliness. Example files have been packaged with the code so that following installation, the pipeline is able to run out of the box. Users can run over these example files while they become familiar with the Balrog environment. To preserve an intuitive feel, Balrog's simulation framework mimics ordinary Python syntax. Files and directories are given understandable names. Log files are automatically written, useful in cases where something does go wrong and debugging must occur. Numerous errors and warning are handled, printing useful messages about why the exception was raised.

In this brief introduction, we have merely scratched the surface explaining Balrog's uses and capabilites. The remainder of the documentation elaborates further. Section 1.2 enumerates the components of the Balrog pipeline, illustrating its algorithm. Until now, we have been rather vague about practical applications for Balrog. Chapter 2 addresses just that. Chapter 3 discusses what is necessary for installation. Beyond, the remainder of the document concerns how to configure and run Balrog. Chapter 4 presents an approach to hit the ground running and quickly get started with some key features of Balrog.

[1] https://github.com/GalSim-developers/GalSim

[2] https://www.astromatic.net/software/sextractor

Chapter 5 builds upon Chapter 4 spelling out all the usage details, including commad line arguments and the framework for specifying galaxy simulations. The format of BALROG's outputs are explained and some debugging hints are offered.

1.2 The Balrog Algorithm

BALROG operates using two initally unrelated pieces of information: real imaging data and simulated galaxies. The central idea of BALROG is to place the simulated galaxies into the images themselves. Figure 1.1.

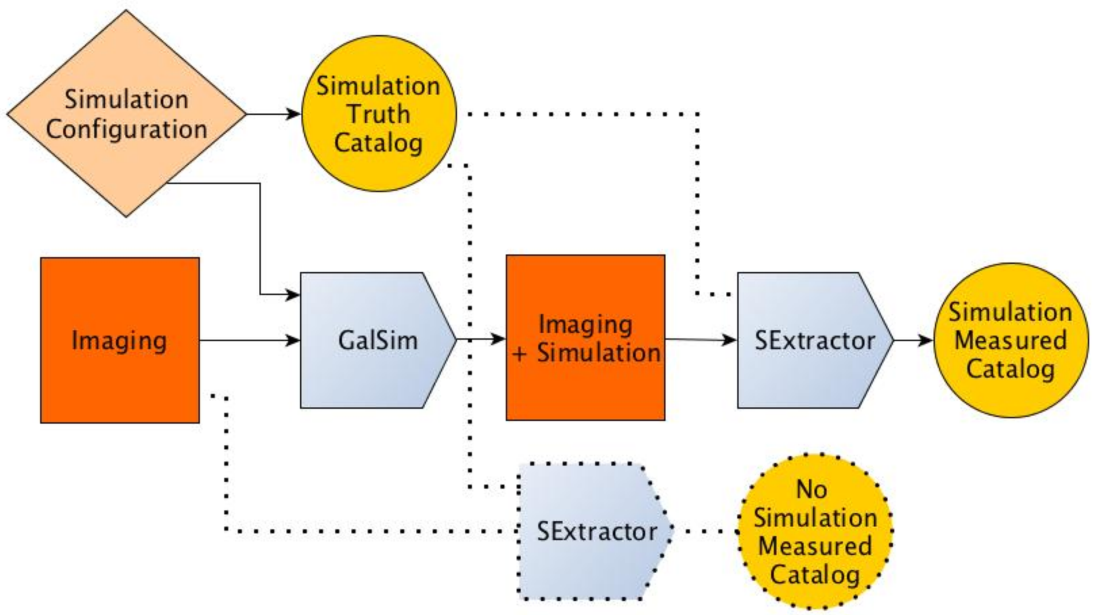

Figure 1.1: Visualization of BALROG's data flow. Optional configurations are represented as dotted connections.

1.2.1 Input Data

BALROG begins by reading in an astronomical image of pixellated flux values. Required with each image is an associated weight map and PSF model. The only file type BALROG accepts for these three inputs is FITS; any other file type will trigger an error. Furthermore, BALROG requires the PSF model be given in the format generated by PSFEx[3]. Readers should be aware that BALROG itself does not run PSFEx. Thus, generating PSFEx models constitues a prerequisite users must complete prior to running BALROG. BALROG simulations will be done in WCS coordinates as opposed to image coordinates, and hence BALROG requires each

[3] www.astromatic.net/software/psfex

image contain a WCS solution. BALROG reads this solution from the image's header. If no WCS exists in the header, a constant default pixel scale of 0.263 arcsec/pixel is adopted.

BALROG supports subsampling the input images if desired, meaning galaxies will only be inserted into a portion of the image. Once galaxies are simulated, the input image's flux values change by adding the galaxies on top of the original images. Algorithmically, this is the only change applied to the input by the entire BALROG pipeline. The weight map and PSF remain unchanged. Conceptually, the bulk of the undertaking goes into simulationg the galaxies, which is described in the following section.

EDS: Is it worth adding some kind of support to generate the PSFEX model? This introduces some issues doing so, but it might be worthwhile.

1.2.2 Galaxy Simulation

To simulate galaxies, BALROG makes use of GalSim. It calls the `Sersic` class to define the galaxies' light distriubtions as Sérsic profiles. By adding `Sersic` objects together, BALROG allows galaxies to be composed of as many superimposed Sérsic components as desired. Each of these components has its own Sérsic index, half light radius, flux, minor to major axis ratio, and orientation angle. The half light radius is measured along the major axis, and the orienation angle is measured as the major axis' counter-clockwise rotation away from the x-direction. In addtion to its Sérsic components, each galaxy shares five parameters which are identical among each Sérsic component: two centroid coordinate positions (x, y), two components of reduced shear (g_1, g_2), and magnfication. The reduced shear follows the usual lensing notation convention, with positive g_1 along the x-axis, negative g_1 along the y-axis, and positive and negative g_2 rotated 45° from the respective g_1 counterparts. Magnification is the usual $\mu = 1 + \kappa$.

Postage Stamp Size

To get the postage stamps ...

1.2.3 Image Processing

SEXTRACTOR, etc.

2. What Is Balrog Good For?

What is BALROG good for?

3. Installation

Installation is a bitch. Let your system administrator worry about it. But if you have to do it, here are some steps that *might* work...

4. Quick Start

BALROG has been designed with flexibility of use in mind. As a necessary consequence, a number of different configuration possiblities exist, each of which must be adequately explained, which quickly expands the length of the documentation. We realize parsing the entirity of this manual requires some time. Hence, the intent of the this section is to offer a short primer for a few of the most import features BALROG users will want to become familiar with. The comprehensive usage instructions are saved for Chapter 5. The following start up sections will refer readers to the relevant sections of Chapter 5 for more details.

The fastest way to get started understanding how to configure and use BALROG is to run it using the example files which come packaged with the software, and then examine the input and output of the run. BALROG has been set up such that when the executable Python file is called without any command line arguments, it will run over the example files, filling in defaults as necessary. Thus, this inital call is as simple as:

```
% runbalrog
```

Referring to Table 4.1, `runbalrog` is equivalent to an alias to the file `balrog.py` located within the installation directory, labelled like an environment variable as `$INSTALLDIR`. We will use these conventions throughout the documentation. Section 4.1 briefly addresses the input which was read in for the `runbalrog` command and Section 4.2 introduces the output generated during the run.

4.1 Input

BALROG's input comes in two forms, command line arguments and Python statements. The command line arguments can be printed, along with brief help string by running:

```
% runbalrog help
```

Furthermore, a complete description of BALROG's command line arguments can be found in Section 5.1. Each comes with a default BALROG will assume if the user does not supply one. In brief, the command line parameters are used to specify input images and their properties, as well as configuration files to use with SEXTRACTOR, and a few other variables every BALROG run will need to define. The default example's image data and PSF live in `$INSTALLDIR/default_example/`. The SEXTRACTOR configuration files live in `$INSTALLDIR/astro_config`. Command line argument names and default file names are intended to be transparent.

In order to flexibly define how galaxies will be simulated, BALROG accepts defined blocks of Python code within the file specified by command line option `--config`. The example's `--config` defaults to `$INSTALLDIR/config.py`. Included within this BALROG configuration file is support for implementing custom user-defined functions and command line arguments.

The core functions defined in the file have a strictly structured syntax which will be fully described in Section 5.1 and Section 5.2. The syntax is designed to be as Pythonesque as possible, so many users will likely be able to extrapolate directly from the examples without reading lengthy documentation. The file also includes comment lines as a guide. A slightly more sophisticated configuration file can be found in `$INSTALLDIR/config2.py`. These two examples are designed to demonstrate the range of statements available to BALROG's Python configuration files.

4.2 Output

All BALROG output is saved in subdirectories under the directory chosen by command line option `--outdir`. If no `--outdir` is given, it defaults to `$INSTALLDIR/default_example/output`. The complete set of output generated by BALROG is detailed in Section 5.3; most relevant for getting started are the `balrog_cat` subdirectory and the `balrog_log` subdirectory. `balrog_cat` contains `example.truthcat.sim.fits`, the truth parameters assigned to simulated galaxies and `example.measuredcat.sim.fits`, the simulated galaxies' properties as measured in the image by SEXTRACTOR. `balrog_log` saves log files useful for debugging BALROG runs, including dumps of how the the command line arguments were parsed, SEXTRACTOR's command line output, and any BALROG specific run errors or warning which occured. Section 5.4 offers some debugging hints.

Table 4.1: Definitions used throughout this manual.

Name	Meaning
`$INSTALLDIR`	Directory where BALROG was installed
`runbalrog`	`$INSTALLDIR/balrog.py`

5. Detailed

5.1 Command Line Options

BALROG runs can be configured via command line options. Two types of options exist. First are the built-in ones, native to BALROG. In addition, BALROG supports a mechanism for users to define their own command line options. To print a list of all BALROG's command line options, both native and user-defined, along with brief help strings, run:

```
% runbalrog --help
```

Section 5.1.1 further details each of the native options and Section 5.1.2 explains how to create custom options.

5.1.1 Built-in Options

BALROG includes a number of built-in optional arguments for each run, defining a variety of parameters such as the input image, the number of galaxies to simulate, the flux calibration, etc. Any options which are not specified assume a default value. The options are intended to be named intuitvely in order to faciltate ease of use. Table 5.1 lists each option, with a description of what it means, including its default. Abbreviations for each option also exist, trading clarity for brevity.

Table 5.1: Command line arguments natively built-in to BALROG.

--Full Name **-Short Name**	**Description**
`--outdir` `-od`	Toplevel directory for BALROG output files. Files will be organized into intuitively named directories under `--outdir`. **DEFAULT**: `$INSTALLDIR/default_example/output/`
`--imagein` `-ii`	Image to insert simulated galaxies into. Must be in FITS format. **DEFAULT**: `$INSTALLDIR/default_example/input/example.fits`
`--imageext` `-ie`	Index of the FITS extension where the image flux data lives. Indexing begins at 0. **DEFAULT**: 0

`--weightin` `-wi`	File containing the weight map associated with `--imagein`. This can be a separate file from `--imagein` or the same file, where the flux image and weigh map live in different extensions. **DEFAULT**: `$INSTALLDIR/default_example/input/example.fits`

`--weightext`
`-we`
	Index of the FITS extension where the weight map data lives. Indexing begins at 0.
	DEFAULT:
	if `--imagein` != `--weightin`:
	`--weighext` = 0
	else:
	`--weightext` = `--imageext` + 1

`--psfin` `-pi`	File containing the PSFEx PSF model for `--imagein`. This is a FITS file, but the convention uses `.psf` as the extension. **DEFAULT**: `$INSTALLDIR/default_example/input/example.psf`
`--xmin` `-xmin`	x-coordinate pixel for the lower bound of the subimage (if subsampling). $x \in [1, N_{\mathrm{cols}}]$. **DEFAULT**: 1
`--xmax` `-xmax`	x-coordinate pixel for the upper bound of the subimage (if subsampling). $x \in [1, N_{\mathrm{cols}}]$. **DEFAULT**: N_{cols}
`--ymin` `-ymin`	y-coordinate pixel for the lower bound of the subimage (if subsampling). $y \in [1, N_{\mathrm{rows}}]$. **DEFAULT**: 1
`--ymax` `-ymax`	y-coordinate pixel for the upper bound of the subimage (if subsampling). $y \in [1, N_{\mathrm{rows}}]$. **DEFAULT**: N_{rows}
`--ngal` `-ngal`	Number of galaxies to simulate. **DEFAULT**: 40

`--gain`
`-gain`
	Gain [e^-/ADU] for adding CCD noise to the simulated galaxies. Refer to galsim.CCDNoise documentation for further details. `--gain` can take two types of values: a float explicitly defining the gain, or a string referring to a keword written in the header of `--imagein[--imageext]`. If neither of these is successfully found, BALROG uses the defaults.
	DEFAULT:
	try:
	`--gain` = `--imagein[--imageext].header['GAIN']`
	except:
	`--gain` = 1.0

`--zeropoint`
`-zp`
	Zeropoint for converting sampled simulation magnitudes to simulated fluxes. SEXTRACTOR will run using this zeropoint when reporting magnitudes. `--zeropoint` can take two types values: a float explicitly defining the zeropoint, or a string referring to a keword written in the header of `--imagein[--imageext]`. If neither of these is successfully found, BALROG uses the defaults.
	DEFAULT:
	try:
	`--zeropoint` = `--imagein[--imageext].header['SEXMGZPT']`
	except:
	`--zeropoint` = 30.0

`--seed` `-s`	Seed to give random number generator for any sampling which requires it, except noise realizations which are always different. **DEFAULT**: Current time
`--fluxthresh` `-ft`	Flux threshold below which the simulated galaxy's profile must fall before drawing the postage stamp. **DEFAULT**: 0.01
`--clean` `-c`	Delete image files after catalogs have been written. **DEFAULT**: Unflagged, i.e. effectively false
`--sexpath` `-spp`	Full path to SEXTRACTOR executable. **DEFAULT**: sex, i.e. system default
`--sexconfig` `-sc`	Configuration file for running SEXTRACTOR. Refer to the SEXTRACTOR user manual or Source Extractor for Dummies for more help. **DEFAULT**: `$INSTALLDIR/astro_config/sex.config`

`--sexparam` `-sp`
: Parameter file specifying which measurements SExtractor outputs. Refer to the SExtractor user manual or Source Extractor for Dummies for more help.
DEFAULT: `$INSTALLDIR/astro_config/bulge.param`. This performs Sérsic profile model fits to each galaxy with free Sérsic index.

`--sexnnw` `-sn`
: SExtractor neural network file for star-galaxy separation. Refer to the SExtractor user manual or Source Extractor for Dummies for more help.
DEFAULT: `$INSTALLDIR/astro_config/sex.nnw`

`--sexconv` `-sf`
: SExtractor filter convolution file when making detections. Refer to the SExtractor user manual or Source Extractor for Dummies for more help.
DEFAULT:

`--noassoc` `-na`
: Do not run SExtractor in association mode. Association mode is SExtractor speak to only look for sources at certain positions; here, the simulated galaxy positions. Using association mode is signficantly faster when simulating into an image consisting of many objects prior to any simulation.
DEFAULT: Unflagged, i.e. use association mode.

`--noempty` `-ne`
: Skip SExtractor run over original image, prior to any simulation. One usage for such a run is to identify cases where a galaxy is simulated in the same position as something originally there. Depending on how the objects' properties conspire, SExtractor may not know any blending happened.
DEFAULT: Unflagged, i.e. perform the SExtractor run

`--sexemptyparam` `-sep`
: Parameter file specifying which measurements SExtractor outputs during run over original image, prior to any simulation. If only interested in the run for 'deblending' issues, the file's contents are mostly irrelevant. The default file does not do model fitting to be faster.
DEFAULT: `$INSTALLDIR/astro_config/sex.param`

5.1.2 User-defined Options

Within the `config.py` file, user's are able to define their own command line options. This occurs within the function `CustomArgs`. Passed to `CustomArgs` as an argument is `parser`, an object made by `python`'s `argparse.ArgumentParser()`. Arguments can be added to parser according to the usual `argparse` syntax. For those unfamilar with `argparse`, this tutorial contains many useful examples. A simple example of `CustomArgs` is copied below.

```
def CustomArgs(parser):
    parser.add_argument("-cs", "--catalogsample", help="Catalog used to
                        sample simulated galaxy parameter distriubtions
                        from", type=str, default=None)
```

User-defined options are parsed within the function `CustomParseArgs`, also part of `config.py`. Passed as an argument to `CustomParseArgs` is `args`, equivalent to an object returned by `parser.parse_args()`. Each one of the user's command line options becomes an attribute of `args`. A simple version of `CustomParseArgs` has been included below.

```
def CustomParseArgs(args):
    thisdir = os.path.dirname( os.path.realpath(__file__) )
    if args.catalogsample==None:
        args.catalogsample = os.path.join(thisdir, 'cosmos.fits')
```

The ability to define and parse one's own command line arguments is intended to make Balrog flexible to conveniently running a wide variety of different simulation scenarios. These parameters are available to the user when setting up the galaxy simulations. How to define these simulations is described in Section 5.2 below.

5.2 Defining How to Simulate Galaxies

Defining how galaxies should be simulated is controlled within `config.py` in the function `SimulationRules`. Passed into the function are three arguments: `args`, `rules`, and `sampled`. `args` refers to the parsed command line arguments, both native Balrog and user-defined. `rules` is an object whose components are overwritten in order to specify how simulated galaxies are sampled. `sampled` gives access to simulated galaxy parameters after they have been sampled. `rules` and `sampled` will become clearer to follow.

Balrog simulates galaxies as N component Sérsic profiles, where N ranges from 1 to as many as desired. Associated with each of these components are five attributes: a Sérsic index, half light radius, magnitude, axis ratio (b/a), and orientation angle (β). In addition, each simulated galaxy has five attributes common among each Sérsic profile: x and y-coordinates, two components of reduced shear (g_1, g_2), and magnification. `SimulationRules`'s argument `rules` is comprised of attributes for these different galaxy characteristics. Users overwrite each of `rules`'s attributes to define their simulations. For example the statement to set each galaxy's magnification to 1 would read:

```
magnification = 1
```

`rules` has 11 attributes in total, whose names are intended to be simple to understand. These are printed and desribed in Table 5.2 below.

Table 5.2: Attributes of the rules defining the simulated galxies.

Attribute	Meaning
`rules.x`	Galaxy centroid x-coordinate [pixels], first pixel = 1
`rules.y`	Galaxy centroid y-coordinate [pixels], first pixel = 1
`rules.g1`	Reduced shear, g_1 component
`rules.g2`	Reduded shear, g_2 component
`rules.magnification`	Magnification, $1 + \kappa$
`rules.nProfiles`	Number of superimposed Sérsic profiles
`rules.sersicindex`	Sérsic index
`rules.halflightradus`	Sérsic half light radius
`rules.magnitude`	Galaxy brightness in magnitudes
`rules.axisratio`	Minor to major axis ratio, b/a
`rules.beta`	Orientaiton angle of major axis (measured from x-axis)

`rules.sersicindex`, `rules.halflightradius`, `rules.magnitude`, `rules.axisratio`, and `rules.beta` must be arrays, whose length is equal to `rules.nProfiles`. For example, simulating galaxies with both an expontential and a de Vaucouleurs component would read:

```
rules.nProfiles = 2
rules.sersicindex = [1,4]
```

Simulation rules can assume four types. The first is a constant, meaning each of the galaxies in the simulated galaxy sample will have the same value for the selected parameter. Rules can also be assigned as an array, equal in length to the number of simulated galaxies. Simulated galaxy i for the chosen parameter then assumes the value in element i of the array. Additionally, sampling can be drawn from a catlog. Multiple parameters selected from the same data table are automatically jointly sampled. Finally a function can be used. Users write their own `python` function, then feed this function and its necessary arguments as the arguments to BALROG's `Function` command. The defined function must return an array equal in length to the number of simulated galaxies. Like the array sampling type, galaxy i will use element i of the returned array. Currently only positional arguments are supported within the user-defined functions, but support for keyword arguments will be added. `Function` affords both flexiblity and convenience when deciding how to sample the simulated galaxies. See `config.py` for examples using `Function` as well as other sampling types. One simple example of how to implement each of the four different types is shown in Table 5.3.

Table 5.3: Syntax examples for each of the simulation types BALROG understands.

Type	Example
Constant	`rules.g1 = 0`
Array	`rules.axisratio = [np.ones(args.ngal)/np.arange(1,args.ngal+1), sampled.axisratio[0]]`

Catalog	`rules.magnitude = [Catalog('cosmos.fits', 0, 'IMAG'), Catalog('cosmos.fits', 0, 'IMAG')]`
Function	`rules.g2 = Function(function=NFW2, args=(sampled.x,sampled.y))`

The second and fourth examples in Table 5.3 makes use of `sampled`, the third argument passed to `SimulationRules`. `sampled` is an object allowing users to refer to the properties of the simulated galaxies after they have been sampled. Referring to the second example above, the first element of `rules.axisratio` is an array decreasing incrementally from 1 to 0. The second elemment of `rules.axisratio` then says to use whatever the sampled values of the first element turn out to be. Here, this equates to setting the second element of `rules.axisratio` to the same array as the first element of `rules.axisratio`, so the use of `sampled` is not really necessary. However, return to the fourth example in Table 5.3. Image `NFW2` as a function which takes two arguments, `x` and `y`, which returns the g_2 component of shear at position (x,y) from an NFW halo with mass $10^{15}M_{\odot}$ whose center lies at (0,0). Now image `sampled.x` and `sampled.y` were sampled randomly:

```
def Random(minimum, maximum, size):
    return np.random.uniform(minimum, maximum, size)


rules.x = Random(args.xmin, args.xmax, args.ngal)
rules.y = Random(args.ymin, args.ymax, args.ngal)
```

`sampled.x` and `sampled.y` now represent the values for x and y after the random sampling has occurred. Along these lines, Balrog can build-up simulations with fairly little recoding between completely different types of simulations. Balrog makes sure that all the simulation parameters are sampled in the proper order and will throw an error if something ambiguous was defined by the user.

5.3 Output

Each Balrog run generates a number of output files. These are organized into a fixed directory structure. Users indicate the `--outdir` command line option, and the remainder of the naming scheme occurs automatically, placing files in subdirectories under `--outdir`. Four subdirectories are written, labelled accoring to what type of files they contain. Table 5.4 lists the contents of each of these subdirectories, giving a brief desription of each file. Depending on how Balrog was configured, not necessarily every file in Table 5.4 will be present in every run. The `*` symbol in Table 5.4 will be replaced with the base name of the input image file. For example, if the input image is named `example.fits`, `*` will be replaced with `example`. If the input file name does not end with the `.fits` extension, the file name itself is used as the base name. This does not include any diretories preceeding the file name. For example, if the input image was given as `/Users/somebody/home/image.f`, the base name would be `image`.

Table 5.4: File structure of the output written by BALROG. * is replaced with the base name of the input image

Path	Contents
`--outdir/balrog_cat/`	Output catalog files
`*.measuredcat.nosim.fits`	SEXTRACTOR catalog over original (subsampled) image, prior to simulation.
`*.measuredcat.sim.fits`	SEXTRACTOR catalog over image which includes simulated galaxies.
`*.truthcat.sim.fits`	Truth catalog of the simulated galaxies' properties.
`--outdir/balrog_image/`	Output images
`*.nosim.fits`	Copy of (subsampled) input image.
`*.sim.fits`	Image containing simulated galaxies.
`*.example.psf`	Copy of (subsampled) PSFEX PSF image.
`--outdir/balrog_log/`	Log files
`--outdir/balrog_sexconfig/`	Files used to configure SEXTRACTOR
`*.assoc.nosim.txt`	SEXTRACTOR's association mode matching list for run without simulated galaxies.
`*.assoc.sim.txt`	SEXTRACTOR's association mode matching list for run with simulated galaxies.
`*.measuredcat.nosim.sex.params`	Copy of input SEXTRACTOR parameter file used for run without simulated galaxies, which properly factors in association mode.
`*.measuredcat.sim.sex.params`	Copy of input SEXTRACTOR parameter file used for run with simulated galaxies, which properly factors in association mode.

5.4 Debugging