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 Introduction **3**

2 The Balrog Pipeline **5**
2.1 Input Data 6
2.2 Simulated Galaxies 7
2.3 SExtractor Implementation 8

3 What Is Balrog Good For? **10**
3.1 Completeness 10
3.2 Clustering 11
3.3 Optimization 12

4 Installation **13**

5 Quick Start **14**
5.1 Balrog Basics 14
5.2 SExtractor Basics 16

6 Native Command Line Arguments **18**

7 Balrog's `pyconfig` File **25**
7.1 User-Defined Command Line Environment 25
7.2 Defining How to Simulate Galaxies 27
7.3 SExtractor Configuration Overrides 32

8 Output **34**
8.1 Output Images 34
8.2 Output Catalogs 35
8.3 SExtractor Configuration Files 36
8.4 Log Files 36

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 questions about how 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] (Rowe et al., *in prep.*) and source extraction and measurement occurs using SExtractor[2] (Bertin & Arnouts, 1996). Balrog facilitates the ease of running these codes en masse over many images, filling in many of the bookkeeping steps in an automated fashion.

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.

In order to maximize convenience, Balrog has been written making our best attempts at user-friendliness. Example files are packaged with the code so that following installation, the pipeline is able to run out of the box without specifying any arguments. Users are encouraged to use and inspect the default example runs to become more familiar with the Balrog environment. To preserve an intuitive feel, Balrog's simulation framework mimics ordinary Python syntax. Where necessary, files and directories are given understandable names. Numerous errors and warnings are handled, printing useful messages about why the exception was raised. Log files are automatically written, useful for follow-up degugging in the cases where exceptions do occur. User configurations are copied into the output directory, owing to the consideration that every run should be reproducible from the output.

Nothing has been mentioned about the Cosmos catalog yet. I need to figure out where to put that in. Not necessarily in the intro

In this brief introduction, we have merely scratched the surface explaining Balrog's uses and capabilites. The remainder of the documentation elaborates further. Section 2 enumerates the components of the Balrog pipeline, illustrating its algorithm. Until now, we have been rather vague about practical applications for Balrog. Section 3 addresses just

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

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

that, offering some examples of what can be done with BALROG outputs. Section 4 discusses what is necessary for installation. Beyond, the remainder of the document concerns how to configure and run BALROG. Section 5 presents an approach to hit the ground running and quickly get started with some key features of BALROG. The following sections build upon Section 5 by more comprehensively spelling out all the usage details. Section 6 covers BALROG's built-in command line arguments. Section 7 treats the file we refer to as BALROG's `pyconfig` file, which is responsible for a number of customizations, including user-defined command line arguments and how the simulated galaxies' properties are generated. The format of BALROG's outputs are explained in Section 8. In general, the separate sections have been written to read fairly independtly, so the entire document does not need to be read sequentially for the sections to make sense. Our approach is to build up the documentation conceptually and break it into manageably sized divisions. One thought to keep in mind is that depending on the user's intended application for BALROG's, configuring BALROG runs can range being from very simple to being fairly involved. The documentation does its best to adequately explain the intricacies when necessary, but not at the cost of clarity for the simpler concepts.

2. The Balrog Pipeline

Section 1 briefly introduced the workflow through the Balrog pipeline. The purpose of this section is to characterize the algorithm in full. The focus here is methodology, not usage instructions. The text is organized as follows. To begin, the workflow of the pipeline as a whole is described. This is then subdivided into three sections to be expanded further, with links to the extended sections where relevant. In Section 2.1, Balrog's required input data is discussed. In Section 2.2, properties of the simulated galaxies are described, with a brief conceptual overview of how the properties can be generated and comments regarding the steps implemented in GalSim. Finally, the functionality of SExtractor's implementation in Balrog is considered in Section 2.3.

We preseent Figure 2.1 as a complement to the text, and as a guide for what is to come. The figure is a flowchart visually representing Balrog. Roughly speaking, the text steps through this flowchart from left to right. However, as the lines in Figure 2.1 indicate, Balrog runs trace out more than a single horizontal line through the diagram. In order to maintain visually clarity and simplicity, Figure 2.1 does not include every possible detail of the pipeline. Rather, it lays out the structure of how the various steps depend on each other. Effectively, there are two requirements: a set of simulation configurations and some imaging data. The pipeline's central components are GalSim and SExtractor, which operate on these requirements to produce output catalogs for further analysis. The following paragraphs comment further.

The Balrog pipeline begins by opening a log file and parsing the command line parameters. The code enforces a number of rules on the user's configurations. If any errors or warnings occur they are directed to the log file. Errors are raised when users make a syntax error and Balrog cannot continue. Generally, warnings occur when something is missing, but the code is able to continue using an internal default. Warnings likely, but not always indicate something is not quite right. The log file remains open, recording messages through the full pipeline.

Next Balrog interprets how the user wants to generate their simulated galaxies. Section 2.2 fully prescribes the attributes of these galaxies. To summarize, each galaxy is composed of one or more Sérsic profiles. The user's directives are executed, and a truth catalog of the galaxy parameters is written.

The code then reads in the imaging data into which the simulated galaxies will be drawn. Refer to Section 2.1 to define what is required of the imaging data in this context. By default, SExtractor is run over the input image. Please note, at this point no simulated galaxies have been inserted. However, this command is potentially useful for reasons discussed further in Section 2.3. An image of each galaxy, commonly known as a postage stamp, is generated and then added atop the input image. Galaxy postage stamp drawing is treated in Section 2.2. Once galaxies have been added to the image, SExtractor is called again. Details of SExtractor's implementation are deferred to Section 2.3. SExtractor outputs a catalog of the object measurements, which is now ready for the user to compare with

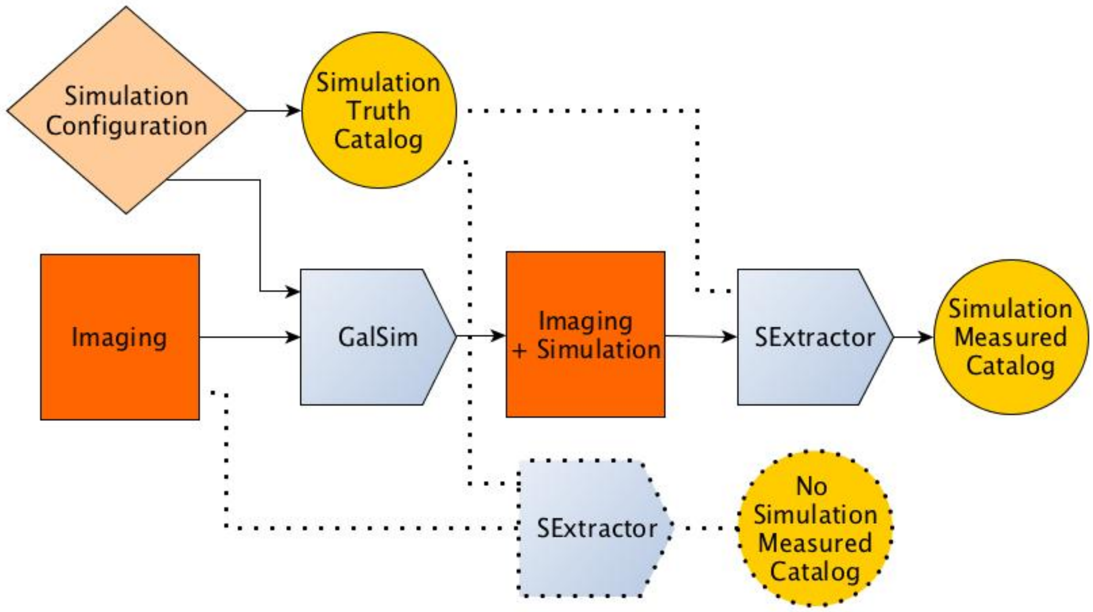

Figure 2.1: Visualization of BALROG's data flow. Optional (but also default) configurations are represented as dotted connections.

the trutch catalog.

2.1 Input Data

BALROG reads in an astronomical image of pixellated flux values. The data should include a photometric calibration defining what one ADU count means physically. Galaxy simulations are done in apparent magnitude space, and BALROG needs a zeropoint, m_z, to convert this to an image ADU level. m_z is defined in the usual way, where the objects flux, $\mathcal{F}$, and apparent magnitude, m, are related by: $m - m_z = -2.5\log_{10}(\mathcal{F})$. A default zeropoint of 30 is assumed if one is not otherwise specified. Required with each image is the image's weight map and PSF model. The weight map will be needed when extracting the sources from the image. At a minimum, the PSF model is necessary for convolving the simulated galaxies prior to embedding them in the given image. It is also obligatory if attempting to fit models to deconvolved measurements of galaxy profiles during the source measurement process. The only file type BALROG supports for input PSF models is that generated by PSFEX[1] (Bertin, 2011). 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 incorporate GalSim's WCS features, and hence the software requires each image contain a WCS solution. BALROG reads this solution from the image's header. If no WCS exists in the header, the pipeline enforces a fiducial WCS with a constant pixel scale of 0.263 arcsec/pixel.[2] BALROG supports subsampling the input images if desired, meaning

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

[2] 0.263 arcsec/pixel is the fiducial pixel scale for DECam.

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 fluxes to the original fluxes. 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.

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

2.2 Simulated Galaxies

To simulate galaxies, Balrog makes use of GalSim. It calls GalSim's class for implementing Sérsic objects to model the galaxies' light distriubtions as Sérsic profiles. By effectively adding together these Sérsic objects, 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 $\hat{x}$-direction. Flux values are generated as magnitudes, then converted to ADU levels using the image's zeropoint. 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 $\mu = 1 + \kappa$. To be explictly clear, the shear and magnification are lensing effects applied to a galaxy, and the axis ratio and orientation angle are intrinsic to the galaxy, as they would be in the abscence of lensing.

Balrog presents users with a number of different options for how to generate the truth properties of their galaxies. Simple types include a constant to be applied commonly to every galaxy or an array containing an element for each galaxy. Alternatively, values can be sampled from the columns in a catalog file. Multiple columns from the same table are automatically jointly sampled. Last but certainly not least, users can define their own functions which determine the truth paramters of the simulated galaxies. This is perhaps Balrog's most powerful feature. It is this functionality which adds the flexibility for Balrog to support virtually anything users can think of and code up in Python themselves. Conveniently, the functions may operate over the galaxies' truth parameters themselves, allowing one parmeter to be defined in terms of another. Maybe I should say something along the lines that Catalog is quite useful if it reproduces the same distributions.

The simulation process runs in a loop with a number of iterations equal to the number of galaxies to be simulated. A subloop then iterates over the number of Sérsic profiles the galaxies are composed of. For each profile, a Sérsic object is obstantiated with the Sérsic index, half light radius, and flux which were generated. Initially these objects are circularly symmetric so the shearing method is called to create the galaxy's axis ratio, oriented in the appropriate direction. All these elliptical objects inside the subloop are added together to create the composite galaxy made of superimposed profiles. Lensing is then applied to the combined object, implementing both the lensing reduced shear and magnfication. The PSF model is sampled at the galaxy's centroid position and convolved with the galaxy profile. Finally, the object is convolved with a top-hat pixel response function whose scale is equal to that of the local WCS pixel scale. Should I say something about how the PSFEx model is not automatically centroided and I have to do it in the code?

The galaxy object is then drawn into a postage stamp image. The implemented drawing routine requires FFTs, whose targeted accuracies can be adjusted by users. The pixel scale of the postage stamp is fixed to have the scale of the local WCS at the galaxy's position. The size of the postage stamp is computed automatically by GalSim by dictating that only a certain small enough fraction of the galaxy's flux is permitted to be lost outside the postage stamp's boundaries. Noise is added to the galaxy's postage stamp by GalSim's functions for creating CCD-like noise. The functionality relies on specifying the gain to set the noise level. This gain defaults to unity when nothing else if available. Balrog sets the read noise to zero. The final step in the simulation process is to add the postage stamp to input image, centering the postage stamp onto the galaxy's generated (x, y) centroid coordinates.

2.3 SExtractor Implementation

The default behavior is to run SExtractor in *association mode*. For those unfamilar with the terminology, association mode means SExtractor will only make possible measurements at a predefined list of coordinates. Here, the list is given as the simulated galaxies' locations. Hence, SExtractor is effectively aware of where in the image the simulated galaxies live. If it finds an object at one of the these coordinates, it extracts it; but it does not extract objects anywhere else. This bypasses the need to extract every single object in the image. Namely, it selects against the objects unrelated to the simulated galaxies that existed in the image prior to any simulation, which have no truth information to compare with anyway. The attractiveness of running in association mode is the reduction in execution time it offers. In order to avoid simulated galaxies overlapping each other, simulated galaxies should be inserted into the image at a significantly lower number density than the number density of the image itself. This means association mode would suppress measurments for the majority of all objects. By default Balrog configures SExtractor to make a Sérsic fit to each simulated galaxy. Because model fitting is a time expensive step, association mode offers a significant payoff when using this configuration. For a large dataset, running SExtractor many times over many images, model fitting each object in every run is likely time infeasible.

In addition to running SExtractor over the image with the simulated galaxies, by default Balrog is configured to run SExtractor over the image prior to inserting the simulated galaxies. This is to confirm if the association mode functionality of SExtractor is genuinely measuring a simulated galaxy. For example, Balrog may happen to place a galaxy into the image at a location where a big, bright object already lives. If the simulated galaxy is rather faint, SExtractor may just interpret its flux as part of the original object, having no idea the simulated galaxy is even there. The SExtractor run over the pre-simulation image allows users to check for such occurences if they would like. By default, this run does not include model fitting. For the case described here, there is no benefit to dedicate the additional time needed for model fitting.

3. What Is Balrog Good For?

No astronomical imaging survey is perfectly homogenous. Variations in image quality and depth are a necessary result from any multi-epoch measurement, even without variations in weather, sky brightness, or telescope properties that can drive inhomogeneities within a single epoch [REF: e.g. Holmes, Hogg, & Rix 2012]. Effective use of imaging data requires a good model for what happens to the data during processing, but the process of creating a catalog from an image generally is generally nonlinear[1], and the relationship between the resulting data products and the world of forms can be quite difficult to model analytically or even heuristically [e.g., SDSS sky subtraction, see REF:Aihara paper, or any deblending algorithm ever].

Our preferred method for characterizing this relationship is the likelihood function:

$$L = p(\text{Catalog}|\text{world of forms}) \tag{3.1}$$

. for given data reduction and catalog-making recipes. This would be necessary for a fully Bayesian imaging analysis (which is probably presently impractical for any large imaging survey), but even for a catalog of point estimates of object properties, L is useful for characterizing the noise and bias of the estimators.

We provide three examples of contemporary relevance below. For each of these, we provide example pseudocode that shows how to compute the quantity of interest using Balrog.

3.1 Completeness

The most common first step in any procedure (REF: SExtractor, PHOTO, maybe THELI?) for creating a catalog from an image is a matched filtering of the image followed by a thresholding. Typically, the filter is chosen to be similar (in the broadest sense of similarity) to the expected image psf [REF: Photo-lite, SEXtractor, etc.]. The resulting detection map is thresholded, and the pixels that exceed the threshold become the focus of subsequent analysis.

For a surface-brightness profile of the same shape as the filter, the matched filter corresponds to the optimal[2] measurement of the flux of an object at that position, so thresholding the detection map corresponds to an actual flux limit. For other surface-brightness profiles, however, this is not true, and any catalog-making process that begins with this step cannot be said to be "magnitude-limited" or "flux-limited". This is a problem, because the completeness of most astronomical imaging is frequently characterized with a single number, corresponding to the magnitude of the faintest object in the catalog; occasionally, conscientious authors will record a significance level (corresponding to the threshold) or that the quoted limiting magnitude is appropriate for either point-sources or galaxies.

[1] Thresholding does this, and no matter what else you do, all catalog-making requires thresholding.

[2] Really optimal, in the sense that it saturates the Cramer-Rao bound, and so a lower-variance unbiased estimator is mathematically impossible.

Our preferred method for describing the survey depth is to compute the probability of detection for the ensemble of sources that will be subjected to analysis. Balrog makes this step straightforward: by embedding a catalog of objects in the real images, and then re-running the catalog-making pipeline, it is easy to determine the detection probability for any objects of interest as a function of their underlying properties.

3.2 Clustering

Two-point clustering measures are among the most common intermediate science products of large astronomical surveys. Variations in the effective survey depth across the sky can produce large systematic errors in the clustering measures (e.g., [REF:]Ross et al 2012, [REF:]Huterer et al 2011). This is universally modeled in such studies by a window function $W(\bar{\theta})$, which is proportional to the probability that a galaxy in the sampled population would have been detected at the position $\bar{\theta}$.

The observed galaxy overdensity field $\delta_g^{\mathrm{obs}} = \frac{n_g}{\bar{n}} - 1$ is then related to the "true" density field (for the sample being selected) as:

$$\delta_g^{\mathrm{obs}}(\bar{\theta}) = \delta_g(\bar{\theta})W(\bar{\theta}) \tag{3.2}$$

and the galaxy autocorrelation function $w(\theta)$ is:

$$w(|\theta|) = \left\langle \delta_g(\bar{\theta}', \delta_g(\bar{\theta}' + \bar{\theta}) \right\rangle \tag{3.3}$$

Here, the expectation values are meant to signify averaging over independent realizations of the universe. Ergodicity allows us to replace this with a volume average. If we assume that W is independent of the choice of realization, then we can form an unbiased (though sub-optimal; see [REF:Landy-Szalay], though the choice is irrelevant for this argument) estimator for w, $\hat{w}$:

$$\hat{w}(\theta) = \frac{\left\langle \delta_g^{\mathrm{obs}}(\bar{\theta}), \delta_g^{\mathrm{obs}}(\bar{\theta}' + \bar{\theta}) \right\rangle}{|W|^2}. \tag{3.4}$$

For high-precision galaxy surveys, however, we cannot ignore the dependence of the detection probability W on δ_g. Others have presented evidence for that this is significant in Sloan Digital Sky Survey imaging [REF: Aihara et al; Ross et al; Huff & Graves], but because of the basic mechanics of detection and catalog-making, similar effects are virtually certain to be present in any large imaging survey.

To see why this may be important, consider the case where W has a linear dependence on δ_g, $W = W_0 + W_1\delta_g$. If this is ignored, and only the static survey window function is known (i.e., we assume $W = W_0$) then the estimator $\hat{w}$ is biased:

$$\hat{w}(\theta) = \frac{\left\langle \delta_g^{\mathrm{obs}}(\bar{\theta}), \delta_g^{\mathrm{obs}}(\bar{\theta}' + \bar{\theta}) \right\rangle}{|W_0|^2} \tag{3.5}$$

$$= \left\langle \delta_g(\bar{\theta}), \delta_g(\bar{\theta}' + \bar{\theta}) \right\rangle + \frac{W_1}{W_0} \left\langle \delta_g^2(\bar{\theta}), \delta_g(\bar{\theta}' + \bar{\theta}) \right\rangle + \left(\frac{W_1}{W_0}\right)^2 \left\langle \delta_g^2(\bar{\theta}), \delta_g(\bar{\theta}' + \bar{\theta})^2 \right\rangle \tag{3.6}$$

There is no static window function W_0, and hence no way to use the standard technique (for correlation functions) of cross-correlating with random catalogs (i.e., DD/RR-1, or (DD-2DR+RR)/RR, etc.) to estimate $w(\theta)$.

We can, however, attempt to re-weight δ_g^{obs} by an estimate of the δ-dependent detection probability, thus keeping the window function inside the ensemble-average, and forming a new estimator:

$$\hat{w}(\theta) = \left\langle \frac{\delta_g^{\mathrm{obs}}(\bar{\theta})}{W(\bar{\theta}|\delta_g)}, \frac{\delta_g^{\mathrm{obs}}(\bar{\theta}' + \bar{\theta})}{W(\bar{\theta}' + \bar{\theta}|\delta_g)} \right\rangle \tag{3.7}$$

We now show how to use BALROG to estimate $W(\bar{\theta}|\delta_g)$.

3.3 Optimization

For gaussian noise with covariance matrix C, an observable vector $\bar{p}$ (such as measured properties for a single entry in a galaxy catalog), and a signal $\bar{s} = \alpha\hat{s}$ (where $\hat{s}$ is a unit vector), the maximum-likelihood estimator for α is:

$$\alpha = \frac{\bar{x}^T C^{-1} \hat{s}}{\hat{s}^T C^{-1} \hat{s}} \tag{3.8}$$

TODO: Demonstrate what you can do if you know the measurement covariance matrix.

4. Installation

BALROG consists of two depencies: GalSim and SEXTRACTOR. BALROG itself is written entirely in Python and will run out of the box if *sufficiently recent* versions of GalSim and SEXTRACTOR have already been installed. As of the writing of this documentation,[1] BALROG requires the latest version of GalSim, which is version `1.1`. We have not thoroughly tested what constitues a *recent enough* version of SEXTRACTOR. What we are able to say is that the oldest version we have successfully tested against is `2.17.0`. BALROG and GalSim both exist as repositories on GitHub and can be downloaded from the command line.

```
% git clone https://github.com/emhuff/Balrog.git
% git clone https://github.com/GalSim-developers/GalSim.git
```

This should automatically check out the master branch of each. Master is the only supported branch for BALROG. The master branch of GalSim will always be a version $\geq$ `1.1`. SEXTRACTOR is packaged as a tarball, which can be downloaded from its official website. We recommend choosing the most recent version.

The GalSim documentation includes an extensive installation guide, with many helpful hints and links to `FAQ` pages. We refer BALROG users to this guide for how to install GalSim. The GalSim team deserves a big thanks for the utility of their installation notes. The BALROG authors are far from accomplished system adminstrators, but we both were independently successful installing GalSim on our systems by following these notes.

Section 3 of the SEXTRACTOR user manual contains a brief section explaining how to install the software. The code itself also includes a file called `INSTALL`, with slightly more detailed installation instructions. One item not explictly stated in the installation notes is that SEXTRACTOR requires the ATLAS[2] linear algebra package as a dependency. SEXTRACTOR will not install properly unless ATLAS is installed. The ATLAS website includes the requisite source code and an extensive installation guide. Once ATLAS builds and installs, the SEXTRACTOR installation steps should be straightforward to follow and complete.

To test the BALROG installation, change directories to the directory `git` cloned for BALROG. The directory contains a file called `./balrog.py`. Run the file from the command line:

```
% ./balrog.py --fulltraceback
```

If installation was successful, this will run without printing any error messages.

[1] 16 April 2014
[2] http://math-atlas.sourceforge.net/

5. 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 later sections. We will refer readers to the relevant comprehensive sections for more details. Included in our startup guide is a brief aside specific to SExtractor. Accoringly, we divide the section into two pieces: Section 5.1 considers Balrog in general, and Section 5.2 narrorws the focus SExtractor alone.

5.1 Balrog Basics

Throughout the current section, we will supplement the discourse with concrete example calls users can run. In order to enable copying and pasting the commands into the user's terminal and running them verbatim, users should create an alias to the Balrog excutable Python file, `balrog.py`. This file is found in the directory which was pulled from GitHub during installation, which we will refer to as `$INSTALLDIR` for convenience. Adopting this convention, the alias statment looks as follows:

```
% alias runbalrog='$INSTALLDIR/balrog.py'
```

For convenient reference in later sections, we have effectively abbreviated what we have just defined in Table 5.1.

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 initial call is as simple as:

```
% runbalrog
```

The rest of the examples here build off this call by adding additional configurations via command line arguments. To explore the command line arguments, one can run the following command to print a useful summary:

```
% runbalrog --help
```

Table 5.1: Definitions used throughout this manual.

Name	**Meaning**
`$INSTALLDIR`	Directory `git` clones for Balrog, i.e. where Balrog is installed
`runbalrog`	`$INSTALLDIR/balrog.py`

For more extended help, the command line arguments built-in to Balrog are detailed further in Section 6. In brief, they 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. Each comes with a default Balrog will assume if the user does not supply one. The default example's image data (`--image`), weight map (`--weight`), and PSF (`--psf`) live in `$INSTALLDIR/default_example/`. The SExtractor configuration files live in `$INSTALLDIR/astro_config/`. File names are intended to be reasonably transparent. Balrog can be told to print messages more verbosely, and it is also instructive to repeat the example Balrog call in verbose mode.

```
% runbalrog --stdverbosity v
```

While the original call ran without printing anything, this one will write out some messages relevant to Balrog filling in defaults for parameters. Such messages can be helpful for users to recognize if they are not in fact using the files they think they are using. Closely related to `--stdverbosity` is `--logverbosity`. While `--stdverbosity` is for `stdout`/`stderr`, `--logverbosity` dictates the level of messaging logged to file. Run the the previous command, but now flag the logging to verbose:

```
% runbalrog --stdverbosity v --logverbosity v
```

Opening the file found in `$INSTALLDIR/default_example/output/balrog_log/run.log.txt` should contain exactly the same statements printed to the command line. Saving output to file is useful when attempting to debug runs after-the-fact. Had any Balrog runtime errors occurred during the run, `run.log.txt` is the file they would be logged to. Additionally, Balrog logs the exact value of each command line argument in `args.log.txt` and directs all SExtractor's `stdout` to `sex.log.txt`.

The command line arguments attempt to be rather intuitive as to what they mean, but one which may not be so obvious is `--pyconfig`. In order to flexibly handle how galaxies will be simulated, Balrog accepts defined blocks of Python code within the file specified by `--pyconfig`. Owing to its command line name, we will refer to this file simply as the `pyconfig` file throughout the documentation. Included within the `pyconfig` file is support for implementing custom functions and command line arguments to be called upon during the galaxy generation process. The core functions defined in `pyconfig` files have a strictly structured syntax which will be fully described in Section 7. The syntax is designed to be as natively Pythonesque as possible, so many users will likely be able to extrapolate directly from examples without reading lengthy documentation. In the current `runbalrog` examples, `--pyconfig` defaults to `$INSTALLDIR/config.py`. This file builds galaxies as single component Sérsic objects. As an instructive guide, the file includes descriptive comment lines explaining what the different functions are used to do. Rather than copying the comments into this document, we encourge user to open the `pyconfig` file themselves. Reading through the file itself is a more efficient method to quickly get started than reading through anything else we could write here. A slightly more sophisticated `pyconfig` file can be found in `$INSTALLDIR/config2.py`. This generates galaxies with both a bulge and a disk component. Together, these two examples are designed to demonstrate the range of statements available to Balrog's `pyconfig` file.

All Balrog output is saved in subdirectories under the directory chosen by command line argument `--outdir`. This directory has defaulted to `$INSTALLDIR/default_example/output` for the example run. The complete set of output generated by Balrog is detailed in Section 8. In brief there are four subdirectories: `balrog_log`, `balrog_image`, `balrog_cat`, and `balrog_sexconfig`. We have already noted three log files written to `balrog_log`. In addition, an explicit copy of the user's `pyconfig` file is also written there. `balrog_image` includes a copy of any image Balrog either used or generated. Ones with `.sim` in their file names include the simulated galaxies, and ones with `.nosim` do not. `balrog_cat` contains `example.truthcat.sim.fits`, the truth parameters assigned to simulated galaxies, as well as the SExtractor catalogs Balrog generates. Again, files including the simulated galaxies contain a `.sim` in their names. and the ones containing a `.nosim` do not. `balrog_sexconfig` copies the files Balrog used for configuring SExtractor. When running Balrog over files other than the defaults, the output file names are automatically generated based on the input file names. It may seem that a single run produces a substantial amount of output, but the idea is that any Balrog call be reproducible and debuggable from its output in case anything does go wrong.

5.2 SExtractor Basics

Here, we momentarily center the attention soley on SExtractor. However, before proceeding further, please take note that this treatment is not intended to be a comprehensive guide to running or understanding SExtractor. Rather, we simply offer a few comments which will hopefully help run SExtractor successfully within Balrog. For those who have never used SExtractor, we direct you to the official SExtractor user manual or alternatively the so-called Source Extractor for Dummies text.

SExtractor runs are configured via roughly a handful of files. In this scope, the two most relevant ones are given by command line arguments `--sexconfig` and `--sexparam`. `sexparam` files are often denoted with a `.param` extension in their file names, while `sexconfig` files are often denoted with a `.sex` extension in their file names. In the example files shipped with Balrog, the `sexconfig` file is suffixed with a `.config` extension for consistency with our terminology. `sexconfig` is allowed to specify any of the arguments which can also be given as command line arguments to SExtractor. These set conditions such as the detection thresholds, the aperture sizes for photometry, the magnitude zeropoint, and the background subtraction strategy. The `sexparam` file is a list of keywords. Each keyword is a measurment SExtractor will make for every extracted object, which will therefore appear as a column in the output catalog.

It is `sexparam` which controls whether or not SExtractor will perform model fits to the galaxies. How do to this is rather scarely documented, but has been passed down by word of mouth in the Astronomy community. We consider it worthwhile to elaborate in writing. SExtractor includes up to two possible types of models to fit. One, denoted by the key `DISK`, is a model with fixed Sérsic index of $n = 1$, i.e. an exponential. The other, denoted by the key `SPHEROID`, is a model with a free Sérsic index. SExtractor can fit either of these independently or both simultaneously. Each model written into and uncommented from the `sexparam` file is fit, meaning if just the `DISK` key is present a disk only model with $n = 1$ is fit; if just the `SPHEROID` key is present a bulge only model with free n is fit; and if both `DISK` and `SPHEROID` keys are present both the disk and bulge are fit simultaneously, which is of course different than fitting them independently. Each model can fit for the flux, magnitude, axis ratio, and orientation angle of the model. The spheroid model also fits the sersic index and half light radius. The disk model fits the scale radius, as opposed to the half light radius. Open the `bulge+disk.param` file included with Balrog in the `astro_config` directory for the SExtractor names of all the parameters. The meanings of the `DISK` and `SPHEROID` names are human understandable.

6. Native Command Line Arguments

Balrog runs can be configured via command line arguments. Two categories of arguments exist. First are the built-in ones, native to Balrog. These constitute the focus of this section. In addition, Balrog also supports a mechanism for users to define their own command line arguments. However, because the functionality is drawn from what we have deemed the `pyconfig` file, its treatment is saved for Section 7.1. To generate a summary of Balrog's command line environment, run the following command, where `runbalrog` is defined in Table 5.1.

```
% runbalrog --help
```

This prints a listing of all the command line arguments (including those defined by the user), along with their available help strings. The built-in arguments' help strings are intended to be a useful quick reference tool, but in order to preserve suitable readability from the command line, they do not necessarily spell out every possible detail users may wish to know. Accordingly, we include a more extensive supplement within this section.

We direct the reader's attention to Table 6.1. It is a comprehensive listing, containing an entry for each of Balrog's command line arguments. Table 6.1 is the core substance of this section, and is intended to be more or less readable on it own, apart from the text of the document. Ergo, we refrain from copying every detail from Table 6.1 into the text, preferring the orderly organization and conceptual clarity of the table over something which would require many more words to describe in paragraph form. However, we will make a few general comments to put Table 6.1 into better context.

Each command line argument can be specified either with its full name or an abbreviated form, e.g. `--image` vs. `--i`. The full names attempt to be as intuitive as possible while maintaining a reasonbly small number of characters. The abbreviatons trade clarity for brevity for those who prefer compactness. Any parameter that is left out of a user's Balrog call assumes a default value in the code. Each of these defaults is listed in Table 6.1. In a few cases, the default values are conditionally defined, meaning Balrog will select one out of two possible defaults depending on how the run has been configured. Table 6.1 utilizes pseudocode to explain the behavior of how these defaults are ultimately chosen. When Balrog parses the user's command line, it performs a number of checks, thereby enforcing proper data types, file existence, etc. Warnings or errors are raised as needed. Table 6.1 has been grouped into subcategories of similar parameters. The divisions do not matter per se; they are merely an organizational guide. However, because Balrog contains quite a few possible arguments to adjust, we find them to be a useful aid in adding to tractability.

`--sexconfig` and `--sexparam`/`--nosimsexparam` require a bit more explanation than is practical within the space of Table 6.1 First, not every keyword-value pair in `--sexconfig` is relevant during Balrog runs. Please note, we are not trying to say these settings have zero or very little effect on SExtractor. What we mean is that their values within `--sexconfig` will be ignored by Balrog, because the values are deduced directly from Balrog's command line, and then given as SExtractor command line arguments, which override the `sexconfig` file. These amount to what has been written into Table 6.2. Table 6.2 notes that the `--sexconfig` and `--sexparam/--nosimsexparam` given to Balrog are not necessarily exactly the ones used when running SExtractor. This is to account for the syntax differnce between running with association mode on or off. If association mode is on, a `sexparam` file must be written with the proper `VECTOR_ASSOC` line. Balrog does so, and then uses the file. Settings in `sexparam` files cannot be overridden from SExtractor's command line. If association mode is turned off Balrog must make sure that no `ASSOC_*` keywords exist in the `sexconfig` file, and writes one out where this is the case for usage downstream. To the best of the authors' knowledge, settings like `ASSOC_*` present in a `sexconfig` file cannot be removed entirely via command line overriding. Striclty speaking, there is not a problem if `VECTOR_ASSOC` is present in the `sexparam` file with association mode off, but Balrog will write and use ones assuredly without it in order to avoid unnecessary warnings. We emphasize, no loss generality has been lost for running SExtractor. We have just automated the process of creating an on/off switch for association mode, and moved some configurations to Balrog's command line because other Balrog functionality besides SExtractor depends on them as well. At any rate, Balrog's output logs will allow users to read off exactly how their SExtractor commands were called (c.f. Section 8).

Table 6.1: Command line arguments natively built-in to BALROG.

Argument	Default
- - - - - - - - - - Output File Organization - - - - - - - - - -	
`--outdir` / `-o`	`$INSTALLDIR/default_example/output/`
Toplevel directory for BALROG output files. Files will be organized into intuitively named directories under `--outdir` (c.f. Section 8).	
`--clean` / `-c`	Unflagged $\implies$ False
Delete intermediate image files (those in `--outdir/balrog_image`) after catalogs have been written.	
- - - - - - - - - - Logging - - - - - - - - - -	

`--stdverbosity`/`--logverbosity` options:

`q`	Quiet	Errors only. (`--logverbosity q` does not exist.)
`n`	Normal	Errors and warnings
`v`	Verbose	Errors, warnings, and info statements
`vv`	Very Verbose	Errors, warnings, info, and debug statments. `vv` is not much different from `v`.

Argument	Default
`--stdverbosity` / `-sv`	n
Verbosity level for printing to `stdout`/`stderr`.	
`--logverbosity` / `-lv`	n
Verbosity level for logging to `--outdir/balrog_log/run.log.txt`.	
`--fulltraceback` / `-ft`	Unflagged $\implies$ False

Print the entire traceback when an exception forces BALROGto exit. By default, BALROG only prints the portion pointing to errors in `--pyconfig`. Problems in `pyconfig` might cause issues downstream, and seeing the full traceback could lead to confusion, thinking there is a coding error in other BALROG files.

- Input Images -

(All images must be given in FITS format. No other file types are supported.)

`--image` / `-i` `$INSTALLDIR/default_example/input/example.fits`

Image into which simulated galaxies are drawn.

`--weight` / `-w` `$INSTALLDIR/default_example/input/example.fits`

File containing the weight map associated with `--image`. This can be a separate file from `--image` or the same file, where the flux image and weight map live in different extensions.

`--psf` / `-p` `$INSTALLDIR/default_example/input/example.psf`

File containing the PSFEx PSF model for `--image`. This is a FITS file, but the convention uses `.psf` as the extension.

- FITS Extensions -

(This is an inteter position, not to be confused with a file name extension. Indexing begins at 0.)

`--imageext` / `-ie` 0

Extension of `--image` where the image flux data lives.

`--weightext` / `-we` $\begin{cases} \texttt{if --image != --weight: 0} \\ \texttt{else: --imageext + 1} \end{cases}$

Extension of `--weight` where the weight map data lives.

- Subsampling -

(Maintaining all defaults uses the entire image. $\{x \in [1, N_{\text{cols}}]\,,\ y \in [1, N_{\text{rows}}]\}$)

`--xmin` / `-x1` 1

Pixel x-coordinate for the lower bound of where to place simulated galaxies into `--image`.

`--xmax` / `-x2` N_{cols}

Pixel x-coordinate for the upper bound of where to place simulated galaxies into `--image`.

`--ymin` / `-y1` 1

Pixel y-coordinate for the lower bound of where to place simulated galaxies into `--image`.

`--ymax` / `-y2` N_{rows}

Pixel y-coordinate for the upper bound of where to place simulated galaxies into `--image`.

- Simulated Galaxy Generation -

`--pyconfig` / `-pc` `$INSTALLDIR/config.py`

Balrog's Python configuration file (c.f. Section 7). It defines the user's custom command line arguments, determines how to generate simulated galaxies, and allows for overriding specifications in `--sexconfig`.

`--ngal` / `-n` 50

Number of galaxies to simulate.

`--zeropoint` / `-z`

```
try: --zeropoint = --image[--imageext].header['SEXMGZPT']
except: --zeropoint = 30.0
```

Zeropoint for converting sampled simulation magnitudes to simulated fluxes. SExtractor will also apply this zeropoint to 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 `--image[--imageext]`. If neither of these is successfully found, Balrog uses the default.

`--gain` / `-g`

```
try: --gain = --image[--imageext].header['GAIN']
except: --gain = 1.0
```

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 `--image[--imageext]`. If neither of these is successfully found, BALROG uses the default.

`--seed` / `-s` Current time

Seed to give random number generator for any sampling which requires it, except noise realizations which are always different.

- SEXTRACTOR -

(Refer to the SEXTRACTOR user manual or Source Extractor for Dummies for more help.)

`--sexpath` / `-sex` `sex`

Path to SEXTRACTOR executable.

`--sexconfig` / `-sc` `$INSTALLDIR/astro_config/sex.config`

Config file for running SEXTRACTOR (c.f. Section 2.3).

`--sexparam` / `-sp` `$INSTALLDIR/astro_config/bulge.param`. This performs a single component Sérsic model fit to each galaxy with free Sérsic index.

Param file specifying which measurements SEXTRACTOR outputs (c.f. Section 2.3).

`--sexnnw` / `-sn` `$INSTALLDIR/astro_config/sex.nnw`

SEXTRACTOR neural network file for star-galaxy separation

`--sexconv` / `-sf` `$INSTALLDIR/astro_config/sex.conv`

SEXTRACTOR filter convolution file when making detections.

`--noassoc` / `-na` Unflagged $\Longrightarrow$ use association mode.

Do not run SEXTRACTOR in association mode (c.f. Section 2.3).

`--nonosim` / `-nn` Unflagged $\Longrightarrow$ perform the SEXTRACTOR run

Skip SExtractor run over the original image, prior to any simulation (c.f. Section 2.3).

`--nosimsexparam` / `-nsp`   `$INSTALLDIR/astro_config/sex.param`. This does not do model fitting.

Param file specifying which measurements SExtractor outputs during run over the original image, prior to any simulation (c.f. Section 2.3).

`--catfitstype` / `-ct`   `ldac`

Type of FITS catalog file SExtractor writes out. `--catfitstype` $\in$ {`ldac`, `1.0`}.

Table 6.2: Translation between Balrog command line argument names and SExtractor command line argument names.

Balrog		SExtractor
- - - - - - - - - - - - - -		Explit -
`--image`	→	`IMAGE`
`--weight`	→	`WEIGHT_IMAGE`
`--sexnnw`	→	`STARNNW_NAME`
`--sexconv`	→	`FILTER_NAME`
`--zeropoint`	→	`MAG_ZEROPOINT`
`--psf`	→	`PSF_NAME`
`--catfitstype`	→	`CATALOG_TYPE`
`--outdir`	→	`CATALOG_NAME`
- - - - Possibly Modified To Account For Association Mode - - - -		
`--sexconfig`	→	`c`
`--sexparam/` `--nosimsexparam`	→	`PARAMETERS_NAME`

7. Balrog's `pyconfig` File

As introduced in previous sections, Balrog's `pyconfig` file is a configuration file for Balrog made up of Python statements. Relying on the ease and simplicity of writing Python code, this file is what contributes a large portion of Balrog's flexibility. The authors cannot natviely build-in to Balrog all the features every user would like to see. Thus, we attempt to create an enviroment which makes it easy for users to fold in additional content themselves. The `pyconfig` file plays a significant role in Balrog, and hence we will take care to provide adequate documentation.

As far as general comments go, we remark that the `pyconfig` file makes use of five core functions called: `CustomArgs`, `CustomParseArgs`, `SimulationRules`, `GalsimParams`, and `SextractorConfigs`. Balrog checks for functions of exactly these names. If one of the function does not exist, Balrog continues by skipping the portion of the code where the function would have been called. An omission should not cause the Balrog run to fail; however, some level of flexibility will have been lost. We issue warning messages to alert the user when one or more of the functions are missing.

Each of the five core functions in `pyconfig` will be described further in what follows. However, it is also important to keep in mind their overall workflow on a more conceptual level. Algorithmically, what occurs is users create command line arguments then parse them appropriately. These arguments, as well as the one Balrog relies on natively, are then made available to functions which implement Python statments to define how Balrog's galaxies are simulated. We have included some convenience features to coordinate the coding of the simulation specfications, including support for users to write their own functions to do so. Within the `pyconfig` environment, it is also possible to override some SExtractor configurations if desired, implementing a certain level of Python control of SExtractor.

In the remainder of Section 7 we concentrate on `pyconfig`'s usage, with a particular focus on elucidating its syntax. Section 7.1 treats user-defined command line arguments within `pyconfig` and Section 7.2 illustrates how `pyconfig` specifies simulated galaxy generation. Section 7.3 describes the way in which `pyconfig` can be used to override SExtractor settings in `--sexconfig`. As an aid while reading this section, we strongly encourage users to open one of the example `pyconfig` files shipped with Balrog, `$INSTALLDIR/config.py` or `$INSTALLDIR/config2.py`. As mentioned in Table 5.1, `$INSTALLDIR` is the directory where Balrog is installed.

7.1 User-Defined Command Line Environment

Within the `pyconfig` file, the definitions for custom command line arguments occur inside the `CustomArgs` function. Passed to `CustomArgs` as a function 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 )
```

Users are allowed to add arguments by any names except those already defined as native Balrog command line arguments. Adding two arguments of the same name in the `ArgumentParser` class raises an exception. The utility of adding custom command line arguments is that the values given to the arguments are made available to the other core functions in `pyconfig`.

The user-defined arguments are parsed within `CustomParseArgs`. When this function is called, Balrog has already grabbed any of the user's custom arguments from the command line, and saved them into an object called `args`. `args` is passed as the single function argument to `CustomParseArgs`. It is equivalent to an object returned by `parser.parse_args()`; each one of the user's command line arguments becomes an attribute of `args`. `CustomParseArgs` allows users to apply any Python operations they so desire to their arguments. 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')
```

As the example demonstrates, one feature of the `CustomParseArgs` is a convenient way for users to effectively define new defaults for Balrog using commonly-known ordinary python syntax. More generally, different Balrog runs are now capable of recognizing and appropriately parsing completely different sets of input parameters.

Returning to `args`, the object not only contains an attribute for each of the user's customly defined arguments, its attributes also include each of the native Balrog arguments. When `CustomParseArgs` is called, the built-in arguments have already been parsed, so arguments which were left out of the command line call have assumed their defaults. Inside `pyconfig`, users are allowed to modify the attributes of `args` derived from bult-in Balrog arguments, if they so desire. However, in order to prevent changes from potentially causing unhandled exceptions later in Balrog, any modifications will not propogate outside the `pyconfig` file. Balrog's native arguments will behave according to how they were specified at the command line. Thus, allowing changes to the native attributes is more of a local convenience than anything else.

`args` will be given as a function argument to the `SimulationRules`, `GalsimParams`, and `SextractorConfigs` functions. Changes within these functions to `args`'s attributes, both those derived from native or custom arguments, are also effectively *local only*. Modifications made in one of the functions do not propogate outside that function. The code has been written this way so users do not need to consider the order in which the three functions execute in order to know excatly what `args` they are using in each function. The behavior is consistent with including the `CustomParseArgs` in the first place. The only place to make persisting changes to `args` is `CustomParseArgs`.

Table 7.1: Attributes of the `rules` object defining the simulated galaxies.

Attribute	Meaning
`rules.x`	Galaxy centroid x-coordinate [pixels], $x \in [1, N_{cols}]$
`rules.y`	Galaxy centroid y-coordinate [pixels], $y \in [1, N_{rows}]$
`rules.g1`	Reduced shear, g_1 component
`rules.g2`	Reduded shear, g_2 component
`rules.magnification`	Magnification, $1 + \kappa$
`rules.sersicindex`	Sérsic index
`rules.halflightradus`	Sérsic half light radius [arcsec]
`rules.magnitude`	Galaxy brightness [mag]
`rules.axisratio`	Minor to major axis ratio
`rules.beta`	Orientaiton angle of major axis (measured from x-axis) [degrees]

Please note, no parsing of any kind can be done inside `CustomArgs`. `CustomArgs` is for definitions only. It informs Balrog what it should be looking for when it receives a command line. No values have yet been assigned to any of the defined parameters when `CustomArgs` exits. `CustomParseArgs` on the other hand tells Balrog what do once it has actually recieved the command line and has saved each of the parameters into an attribute of `args`.

7.2 Defining How to Simulate Galaxies

Deciding how the simulated galaxy properties should be sampled occurs inside the function `SimulationRules`. Passed into the `SimulationRules` are three arguments: `args`, `rules`, and `sampled`. As discussed in Section 7.1, `args` refers to the parsed command line arguments, both native Balrog and user-defined. `rules` is an object whose attributes 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.

Section 2.2 laid out the parameters which characterize each simulated galaxy. Briefly recapitulating, each galaxy is made up of one or Sérsic profiles. Each Sérsic profile is composed of five components: Sérsic index (n), half light radius, brightness magnitude, minor to major axis ratio, and orientation angle counter-clockwise from the $\hat{x}$-direction (β). Furthermore, a simulated galaxy also shares fives additional parameters among each Sérsic profile: two centroid position coordinates, (x, y), two reduced shear coordinates (g_1, g_2), and magnification. This totals ten "*variables*" which fully specify the galaxy. `rules` and `sampled` are each composed of ten attributes, matching these same ten variables. The attribute names for `rules` and `sampled` have been transparently chosen. Nevertheless, for completeness the names and meanings are fully specified in Table 7.1. Table 7.1 uses `rules` as the concrete example. The attribute names of `sampled` are identical. For the time being, we will continue assuming each galaxy has been simulated as a single Sérsic profile. We will return to handling two or

Table 7.2: Syntax examples for each of the simulation types Balrog understands.

Type	Example
Constant	`rules.g1 = 0`
Array	`rules.x = np.linspace(args.xmin, args.xmax, args.ngal)`
(Sampled	`rules.y = sampled.x)`
Catalog	`rules.magnitude = Catalog(file='cosmos.fits', ext=1, col='IMAG')`
Function	`rules.y = Function(function=rand, args=[args.xmin,args.xmax,args.ngal])`

more in a few paragraphs.

Users overwrite each of the `rules` object's attributes to define their simulations. Assignments can assume four (or possibly five depending how categorized) types of statements. The syntax for each is designed to be straightforward, intuitive, and Pythonesque. For quick reference, an example of each is included in Table 7.2. The first type of rule assignment is a constant, meaning each of the galaxies in the simulated galaxy sample will have the same value for the selected parameter, e.g. `rules.g1 = 0`. Second, rule types can also be given 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. The example in Table 7.2 is given as `rules.x = np.linspace(args.xmin, args.xmax, args.ngal)`, meaning galaxies will be evenly spaced along the x-axis across the image. The third sampling type for `rules` is drawing from a catalog. This makes use of a function Balrog has defined called `Catalog`. Calls to `Catalog` require three arguments: `file`, `ext`, and `col`. `file` is the file path to a FITS file; `ext` identifies which extension contains the data table to use; and `col` is the name of the column to draw from. If `Catalog` is used multiple times, multiple columns selected from the same data table are automatically jointly sampled. An example call to `Catalog` looks like: `rules.magnitude = Catalog(file='cosmos.fits', ext=1, col='IMAG')`. Last but not least, the fourth assignable sampling type to `rules` is a function. Users write their own function according to usual Python syntax, then feed this function and its necessary arguments as the arguments to Balrog's `Function` command. The one requirement of sampling functions is they 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 as its truth parameter. Here, we provide an example code snippet, illustrative of how one could place the simulated galaxies at random image positions.

```
def rand(minimum, maximum, ngal):
    return np.random.uniform( minimum, maximum, ngal )

def SimulationRules(args, rules, sampled):
    rules.x = Function(function=rand, args=[args.xmin, args.xmax, args.ngal])
    rules.y = Function(function=rand, args=[args.ymin, args.ymax, args.ngal])
```

`Function` affords a great deal of flexiblity toward simulating a wide variety of different scenarios. We will make further statements regarding slightly more sophisticated usage of `Function`, but it is useful to consider a few other points first. I need to mention args/kwargs.

Let us now return to the `sampled` object which is passed to `SimulationRules` as an argument. `sampled` is used in conjuction with `rules` to define one parameter's rule in terms of another's. Conceptually speaking, `sampled` can be thought of an object whose attributes are arrays that have been set equal to the simulated galaxy parameters post sampling. One might consider `sampled` its own sampling type, but at the same time, its also functionally equivalent to an array. This is why a Sampled type has been added as a fifth entry in Table 7.2, in parentheses directly under the Array line. To better understand `sampled`, consider the following code sample:

```
def SimulationRules(args, rules, sampled):
    rules.x = Function(function=rand, args=[args.xmin, args.xmax, args.ngal])
    rules.y = sampled.x
```

The x-coordinates randomly sample, and then each galaxy's y-coordinate is set equal to exactly the same value as its x-coordinate. Please note, this is expressly different from the original example where both x and y sampled randomly. `sampled` can also be used as an argument within the `args` passed to a sampling function, as is the case in the proceeding example.

```
def SimulationRules(args, rules, sampled):
    rules.x = Function(function=rand, args=[args.xmin, args.xmax, args.ngal])
    rules.y = Function(function=rand, args=[args.ymin, args.ymax, args.ngal])
    rules.axisratio = Function(function=SampleFunction, args=[sampled.x,
        sampled.y,args.xmax, args.ymax])

def SampleFunction(x, y, xmax, ymax):
    dist = np.sqrt(x*x + y*y)
    max = np.sqrt(xmax*xmax + ymax*ymax)
    return dist/max
```

This functionality allows users to have access to the sampled galaxy truth information in their functions. This particular example gave each galaxy a position dependent axis ratio. As a further eample, one could apply position dependent lensing effects in a similar fashion.

Thus far we have been operating with single component Sérsic models. However, two or more components can be implemented. Doing so requires a call to a Balrog function called `InitialSersic` before any of `rules`'s five Sérsic attributes (`sersicindex`, `halflightradius`, `magnitude`, `axisratio`, `beta`) are reassigned. `InitialSersic` takes `rules` and `sampled` as arguments as well as `nProfiles`, the number of superimposed Sérsic models, e.g. `InitializeSersic(rules, sampled, nProfiles=2)`. This has recast the Sérsic components of `rules` and `sampled` as length two list. Now the elements of these attributes of `rules` must be updated accordingly, using syntax akin to that of any other Python list. For example, the following builds galaxies as bulge + disk models, using an exponentional disk ($n = 1$) and a de Vaucouleurs bulge ($n = 4$).

```
def SimulationRules(args, rules, sampled):
    rules.x = Function(function=rand, args=[args.xmin, args.xmax, args.ngal])
    rules.y = Function(function=rand, args=[args.ymin, args.ymax, args.ngal])
    rules.g1 = 0
    rules.g2 = 0
    rules.magnfication = 1

    InitializeSersic(rules, sampled, nProfiles=2)
    rulex.sersicindex = [1, 4]
    rules.halflightradius[0] = Catalog('cosmos.fits',1,'HALF_LIGHT_RADIUS')
    rules.halflightradius[1] = sampled.halflightradius[0]
    rules.magnitude[0] = Catalog('cosmos.fits',1,'IMAG')
    rules.magnitude[1] = sampled.magnitude[0]
    rules.beta[0] = Function(function=rand, args=[-90, 90, args.ngal])
    rules.beta[1] = sampled.beta[0]
    rules.axisratio[1] = sampled.axisratio[0]
    rules.axisratio[0] = Function(function=rand, args=[0.05, 1, args.ngal])
```

The example uses bulges and disks with identical half light radii, magnitudes, axis ratios, and orientation angles. The values for the half light radius and magnitude are sampled from the catalog `cosmos.fits`. The axis ratios and orientations angles are random, as are the galaxy positions. No lensing is applied. We direct the reader's attention to the final two lines. Conceptually one would often think of these in the opposite order, but either is permissible and equivalent in BALROG. Also notice that the `x`, `y`, `g1`, `g2`, and `magnification` attributes are unaffected by the `InitializeSersic` statement. These attributes are never lists.

All attributes of `rules` have a default, which will be used if the attribute is not overwritten. Any time a default must be used a warning message is issued. The defaults themselves are listed in Table 7.3. Notice that in some cases, the default is to sample from the `cosmos.fits` catalog which ships with BALROG. These should create realistic populations. By default a single component Sérsic model is adopted. If the Sérsic mode is multicompoents, each element of an attribute's list assumes the default in Table 7.3. Whenever one uses `InitializeSersic`, the Sérsic attributes are reinitialized to their defaults, so be careful if `InitializeSersic` is called more than once in `SimulationRules`.

Previously we mentioned we would elaborate further on some finer details of using `Function` to sample. We now do so. Inside `Function` statements, it is possible to make an element of `args` a `Catalog` statement. Here is a concrete example.

```
def SimulationRules(args, rules, sampled):
    nc = Catalog(file='cosmos.fits',ext=1,col='SERSIC_INDEX')
    n = Function(function=g, args=[1, 0.05, args.ngal, nc])
    rules.sersicindex = n

def g(avg, std, ngal, add):
    gaus = gaussian(avg, std, ngal)
    return gaus+add
```

Table 7.3: `rules` defaults.

Attribute	Default
`rules.x`	`np.random.uniform(args.xmin, args.xmax, args.ngal)`
`rules.y`	`np.random.uniform(args.ymin, args.ymax, args.ngal)`
`rules.g1`	0
`rules.g2`	0
`rules.magnification`	1
- - - - - - - - - - - - - - - -	Sérsic -
(If Sérsic attributes are multi-component, each list element uses this default)	
`rules.sersicindex`	`Catalog('cosmos.fits',1,'SERSIC_INDEX')`
`rules.halflightradius`	`Catalog('cosmos.fits',1,'HALF_LIGHT_RADIUS')`
`rules.magnitude`	`Catalog('cosmos.fits',1,'IMAG')`
`rules.axisratio`	1
`rules.beta`	0

```
def gaussian(avg, std, ngal):
    return np.random.normal( avg, std, ngal )
```

An amount which is sampled from the catalog is added to the Gaussian. This particular implementation is merely illustrative, not necessarily something one would realistically do scientifically. If the table given in `Catalog` was also used to sample any other parameters, everything is still automatically jointly sampled. `Function` can also take other `Function` statments as elements of `args`. To demonstrate, we form another example very similar to the previous one.

```
def SimulationRules(args, rules, sampled):
    ns = Function(function=exact, args=[np.ones(args.ngal)])
    n = Function(function=g, args=[4, 0.05, args.ngal, ns])
    rules.sersicindex = n

def exact(item):
    return item

def g(avg, std, ngal, add):
    gaus = gaussian(avg, std, ngal)
    return gaus+add

def gaussian(avg, std, ngal):
    return np.random.normal( avg, std, ngal )
```

Here, the usage of the `exact` funciton is trivial, but nevertheless representive of the proper usage.

`SimulationRules` is written to be handle errors. When users try to enter a sampling specification that BALROG does not understand an exception is raised. Exceptions are also raised if users attempt to reassign attributes of `sampled` or access an attribute of `rules` or `sampled` which does not actually exist. The mentioned exceptions cause BALROG to terminate. The logic is that any BALROG `pyconfig` syntax errors are treated like ordinary Python syntax errors and thus force the execution thread to exit. The traceback will include line numbers for where the process was terminated.

The final consideration to make about generating simulated galaxies is the function `GalsimParams`. This function is passed three arguments: `args`, `gsparams`, and `galaxies`. `args` is the same parsed command line argument object as passed given to `SimulationRules`. Likewise, `galaxies` is the same object as `sampled`. It has just been renamed for clarity. `gsparams` is an object which populates each galaxy's GalSim `GSParams`. `gsparams` has an attribute for each `GSParam`, and the attributes behave essentially the same as the non-Sérsic components of `rules`. Each attribute of `gsparams` can take on the same four (five) sampling types as `rules`: a constant, array, (`galaxies` attribute–not particularly useful here on its own but could be inside `Function`), `Catalog`, or `Function`, where all the same functionality applies to `Function` as above.

Understanding all the `GSParams` is a fairly technical subject, so we refer users who need to adjust them to GalSim's GSParams reference page for the complete details. In the context of BALROG these parameters are relevant for drawing the galaxies into postage stamp images. `alias_threshold`, for example, determines what maximum fraction of the galaxy's flux may be lost outside the boundaries of the postage stamp when determining how large of a postage stamp is needed for drawing the galaxies. Convolution with the PSF and pixel response function is done in Fourier Space. `maxk_theshold` determines the maximum k used in FFTs such that k-values must fall below `maxk_threshold` before cutting off the FFT. We will mention, there are also parameters for setting target accuracies. Each `GSParam` has a built-in GalSim default. Any attribute not explictly assigned in `GalsimParams` uses this default. In most cases, the defaults work well enough. The exception is when very bright objects (~ 15 mag, depending on the calibration) are being drawn, and even a small percentage of the flux totals to a signifcant number of ADU. When one thinks there might be an issue, it is useful to visually inspect the image into which the simulated galaxies have been drawn (c.f. Section 8) for signs of aliasing.

7.3 SExtractor Configuration Overrides

The final of the core functions in `pyconfig` which we are yet to discuss in detail is `SextractorConfigs`. This function is very simple. It is passed two arguments: `args` and `config`. As was the case in `SimulationRules` and `GalsimParams`, `args` contains the parsed command line arguments. `config` is a dictionary. Users are free to add to the dictionary the keyword-value pairs that a `sexconfig` file understands. These will then override what was given in `--sexconfig` by giving them as command line parameters to SEXTRACTOR. This effectively allows users to do anything from the BALROG command line they could do from the SEXTRACTOR command line. While none of the keywords will cause an error in `SextractorConfigs`, the same issue discussed in Section 6 applies here. A dozen or so of the keywords will not actually overwrite anything. Namely, each of the SExtractor parameters in the right-hand column of Table 6.2 cannot explicitly be overridden via `config`. To ensure proper implementation of SExtractor's association mode, no `ASSOC_*` keywords added to `config` would have any effect either.

8. Output

Each Balrog run generates a number of output files. At first, this number may seem larger than was perhaps expected. However, we air on the side of writing to disk anything users might possibly want afterward. We intend for Balrog to be transparent and debuggable. All runs should be recreateable from their output files.

The output files are organized into a fixed directory structure. Users indicate the `--outdir` command line argument, 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. Images write to `--outdir/balrog_image/`, catalogs go to `--outdir/balrog_cat/`, SExtractor configurations are saved to `--outdir/balrog_sexco` and log files are directed to `--outdir/balrog_log/`. Table 8.1 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 8.1 will be present in every run. The `*` symbol in Table 8.1 will be replaced with the base name of the input image file. By base name, we mean `--image`'s file name, stripping off the `.fits` extentension, as well as any proceeding directories locating it on the filesystem. For example, if the input image is named `/Users/Balrog/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.

The following sections will further examine the contents of all Balrog's output files, building off Table 8.1. Each section discusses one of the output's four subdirectories, equivalent to one of the headings in Table 8.1. Section 8.1 briefly addresses the output images, while Section 8.2 describes the catalog files Balrog saves. Section 8.3 concerns SExtractor files, and Section 8.4 details what exactly Balrog logs to file.

8.1 Output Images

Every successful Balrog run saves three, possbily four, files to `balrog_image/`. The `*.nosim.fits` and `*.psf` files are copies of the input image (`--image`) and the input PSF (`--psf`) respectively. If no subsampling was done, `*.nosim.fits` and `*.psf` are actually symbolic links to the input files instead of hard copies. Making symbolic links avoids writing potentially large files to disk compared to making hard copies. SExtractor only operates over full files, so subsampling requires writing new files. Subsampling the PSF amounts to appropriately changing the `POLZERO1` and `POLZERO2` header keywords. The `*.sim.fits` file is the image the simulated galaxies have been written into. If the weight map lives in a separate file from the flux image, it is written to `*.weight.fits`. It will also be a symbolic link if no subsampling has occurred. A `.sim` or `.nosim` qualifier for the weight image is irrelevant because the simulation process does not change it. When users flag the `--clean` option, the three images in `balrog_image/` are deleted upon completion of Balrog.

8.2 Output Catalogs

BALROG writes up to three catalog files: `*.measuredcat.nosim.fits`, `*.measuredcat.sim.fits`, and `*.truthcat.sim.fits`. The truth catalog, `*.truthcat.sim.fits`, is always made. The catalog contains a column for each of the attributes of listed in Table 7.1 Five of the columns are named exactly as they appear in Table 7.1: `x`, `y`, `g1`, `g2`, and `magnification`. `sersicindex`, The `halflightradius`, `magnitude`, `axisratio`, and `beta` attributes are allowed to exist as array. The truth catalog includes columns for each of these attributes, whose names are indexed with respect to how many Sérsic profiles have been simulated. The indexing appears as a '`_%i`' appended to the attribute name. Indexing begins at zero. For example, when simulating a two component Sérsic object, the truth catalog would contain `sersicindex_0` and `sersicindex_1` columns. A single component model would only include `sersicindex_0`. In the truth catalog, `magnitude` exists as `flux` [ADU] instead of explicitly `magnitude` [mag] because flux is what is drawn into the image by GalSim. The final two columns of `*.truthcat.sim.fits` are `flux_noiseless` and `flux_noised`. While the `flux_%i` keywords are the simulation parameters that were generated for each Sérsic component, `flux_noised` is the total flux GalSim actually drew into the postage stamp, and `flux_noiseless` is the same without any noise. Only one `flux_noiseless` and `flux_noised` exists per galaxy.

`*.measuredcat.nosim.fits` and `*.measuredcat.sim.fits` are the two SEXTRACTOR catalogs. `*.measuredcat.sim.fits` indcludes the simulated galaxies and `*.measuredcat.nosim.fits` does not. If `--nonosim` is flagged, there will not be a `.nosim.fits` file. When using association mode, the catalogs are made up solely of simulated galaxies, modulo blending effects (c.f Section 2.3). Likely all the simulated galaxies will not be detected so there usually will not be as many rows as simulated galaxies. Furthermore, if association mode was used, `.sim.fits` includes the truth information in the `VECTOR_ASSOC` column. Each row of of `VECTOR_ASSOC` is an array of truth parameters. The catalog's header enumerates exactly what each element of the array means. There are keywords `V%i` and `VUNIT%i`, which will match the names of the truth catalog's columns. These are the ordered values of the `VECTOR_ASSOC` array along with their units. For example, we can print the relevant header information from the deafault example runs in Section 5.1:

```
% listhead "default_example/output/balrog_cat/example.measuredcat.sim.fits[2]" |
   grep V | grep " ="
TTYPE1 = 'VECTOR_ASSOC' / ASSOCiated parameter vector
V0 = 'g2 '
VUNIT0 = 'none '
V1 = 'g1 '
VUNIT1 = 'none '
V2 = 'magnification'
VUNIT2 = 'none '
V3 = 'flux_noiseless'
VUNIT3 = 'ADU '
V4 = 'y '
VUNIT4 = 'pix '
V5 = 'x '
```

```
VUNIT5 = 'pix '
V6 = 'flux_noised'
VUNIT6 = 'ADU '
V7 = 'halflightradius_0'
VUNIT7 = 'arcsec '
V8 = 'beta_0 '
VUNIT8 = 'deg '
V9 = 'sersicindex_0'
VUNIT9 = 'none '
V10 = 'axisratio_0'
VUNIT10 = 'none '
V11 = 'flux_0 '
VUNIT11 = 'ADU '
```

Think of 'V' as shorthand for 'VECTOR'. The indecies then label the components of this vector. Since a truth catalog itself is written by Balrog it is not strictly necessary to include the truth vector in the measured catalog. However, it is convenient and essentially comes for free with Balrog's association mode setup.

8.3 SExtractor Configuration Files

Table 8.1 itself already accounts for many of the details we would like to point out relevant to the files in the `balrog_sexconfig/` directory, so just a few additional comments are in order. To run SExtractor in association mode, users give SExtractor a list of coordinates to match to, which is stored in a `.txt` file. This is what `*.assoc.nosim.txt` and `*.assoc.sim.txt` are used for; they amount to `txt` file versions of the truth catalog. The files given by command line arguments `--sexconfig`, `--sexparam`, `--nosimsexparam`, `--sexnnw`, and `--sexconv` are copied to the output directory to facilitate reproducibility. Users may very well modify the SExtractor configurations files on their filesystem, and even if they do they still have copies of how they existed when Balrog ran. Because Balrog automatically accounts for turning SExtractor association mode on and off, `--sexconfig` and `--sexparam`/`--nosimsexparam` may need to be slightly modified, even when allowing for SExtractor configuration from the command line (c.f. Section 6). `*.measuredcat.nosim.sex.config`, `*.measuredcat.sim.sex.config`, `*.measuredcat.nosim.sex.para` and `*.measuredcat.sim.sex.params` are the slightly modified versions and are written to the output directory as well. Any settings unrelated to association mode remain unchanged.

8.4 Log Files

Balrog saves four files to the `balrog_sexconfig/` directory. The first is an exact copy of the user's `pyconfig` file. Next, as its name would suggest, `run.log.txt` is a Balrog *run log*. Any errors or warnings Balrog raised would be directed to this file. The verbosity of the file's messages is dictated by `--logverbosity`; see Table 6.1 for the available options. For the most conservative logging, using `--logverbosity vv` will print everything Balrog has been programmed to possibly log. `args.log.txt` logs the command line arguments passed to BALROG and `sex.log.txt` logs how SEXTRACTOR was called, along with the output SEXTRACTOR printed. `args.log.txt` and `sex.log.txt` include comment lines specifying what is printed in that section. The easiest way for users to familiarize themselves with the files' contents is to open the files and have a look themselves. Quickly summarizing adding a few notes about the run log, `args.log.txt` contains the exact call BALROG was instantiated with from the command line, the value BALROG initially assumed for each argument (filling in with defaults where necessary), the final parsed value for each argument, and some *pseudo arguments*–parameters BALROG deduces from the command line arguments. Strictly speaking users should not need to understand the pseudo arguments; nevertheless, they may be illustrative for understanding BALROG's workflow or helpful during debugging.

Table 8.1: File structure of the output written by BALROG. * is replaced with the base name of the input image

- Output Images -

`--outdirbalrog_image/`

`*.nosim.fits`
Copy of (subsampled) input image.

`*.sim.fits`
Image containing simulated galaxies.

`*.weight.sim.fits`
Copy of (subsampled) weight map.

`*.psf`
Copy of (subsampled) PSFEX PSF image.

- Output Catalogs -

`--outdir/balrog_cat/`

`*.measuredcat.nosim.fits`
SEXTRACTOR catalog measured over original image, prior to simulation. Only written if `--nonosim` is unflagged.

`*.measuredcat.sim.fits`
SEXTRACTOR catalog measured over image which includes simulated galaxies.

`*.truthcat.sim.fits`
Truth catalog of the simulated galaxies' properties.

- - - - - - - - - - - - - - - - - SExtractor Configuration Files - - - - - - - - - - - - - - - - -

`--outdir/balrog_sexconfig/`

`*.assoc.nosim.txt`
Association mode matching list for SExtractor run prior to adding simulated galaxies. Only written if `--nonosim` and `--noassoc` are unflagged.

`*.assoc.sim.txt`
Association mode matching list for SExtractor run with simulated galaxies. Only written if `--noassoc` is unflagged.

`${--sexconfig}`
Explicit copy of file given by `--sexconfig`.

`${--sexparam}`
Explicit copy of file given by `--sexparam`.

`${--nosimsexparam}`
Explicit copy of file given by `--nosimsexparam`.

`${--sexnnw}`
Explicit copy of file given by `--sexnnw`.

`${--sexconv}`
Explicit copy of file given by `--sexconv`.

`*.measuredcat.nosim.sex.config`
SExtractor `config` file actually with run prior to simulation. If this file exists, it is closely related, but not identical to `--sexconfig` (c.f. Section 6). Only written if need be.

`*.measuredcat.sim.sex.config`
SExtractor `config` file actually used with run post simulation. If this file exists, it is closely related, but not identical to `--sexconfig` (c.f. Section 6). Only written if need be.

`*.measuredcat.nosim.sex.params`
SExtractor `param` file used during run prior to inserting simulated galaxies. Closely related, but not necessarily identical to `--sexparam` (c.f. Section 6). Only written if `--nonosim` is unflagged.

`*.measuredcat.sim.sex.params`
SExtractor `param` file used during run including simulated galaxies. Closely related, but not necessarily identical to `--sexparam` (c.f. Section 6).

- Log Files -

`--outdir/balrog_log/`

`run.log.txt`
Any Balrog warnings, errors, or other messages generated a run. `--logverbosity` applies to this file

`args.log.txt`
Logs the command line parameters given for a Balrog run.

`sex.log.txt`
SExtractor command line input/output.

`${--pyconfig}`
Explicit copy of file given by `--pyconfig`