Balorog

For those who dig too deeply and greedily into their data...

Eric Suchyta and Eric Huff

Contents

1 Introduction

Balrog is ...

2 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...

3 Quick Start

First run:

```
% ./balrog.py
```

and check the output directories in `default_example/output`. Check `config.py` for how to do simulations.

4 The Balrog Algorithm

Balrog does ...

4.1 Postage Stamp Size

To get the postage stamps ...

5 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:

```
% ./balrog.py --help
```

Section 5.1 further details each of the native options and Section 5.2 explains how to create custom options.

5.1 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 1 lists each option, with a

description of what it means, including its default. Abbreviations for each option also exist, trading clarity for brevity. Use the `--help` flag if interested in the abbreviations.

Table 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). $x \in [1, N_{\rm cols}]$. **DEFAULT**: 1

`--xmax`
`-xmax`
: x-coordinate pixel for the upper bound of the subimage (if subsampling). $x \in [1, N_{\rm cols}]$.
DEFAULT: $N_{\rm cols}$

`--ymin`
`-ymin`
: y-coordinate pixel for the lower bound of the subimage (if subsampling). $y \in [1, N_{\rm rows}]$.
DEFAULT: 1

`--ymax`
`-ymax`
: y-coordinate pixel for the upper bound of the subimage (if subsampling). $y \in [1, N_{\rm rows}]$.
DEFAULT: $N_{\rm rows}$

`--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`

5.2 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 6 below.

6 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:

```
rules.magnification = 1
```

`rules` has 11 attributes in total, whose names are intended to be simple to understand. These are printed and desribed in Table 2 below.

Table 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 3.

Table 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 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 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.

7 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 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 4 will be present in every run. The `*` symbol in Table 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 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.

8 Debugging