blendz Documentation

Release 1.0

Daniel Michael Jones

Oct 03, 2018

Contents

Bayesian photometric redshifts of blended sources `blendz` is a Python module for estimating photometric redshifts of (possibly) blended sources with an arbitrary number of intrinsic components. Using nested sampling, `blendz` gives you samples from the joint posterior distribution of redshift and magnitude of each component, plus the relative model probability to identify whether a source is blended.

`blendz` is easy to install using pip

```
pip install blendz
```

and can be run using either simple configuration files

```
pz = blendz.Photoz(config_path='path/to/config.txt')
pz.sample(2)
```

or keyword arguments

```
pz = blendz.Photoz(data_path='path/to/data.txt',
                   mag_cols = [1, 2, 3, 4, 5],
                   sigma_cols = [6, 7, 8, 9, 10],
                   ref_band = 2,
                   filters=['sdss/u', 'sdss/g',
                            'sdss/r', 'sdss/i', 'sdss/z'])

pz.sample(2)
```

to set the configuration.

CHAPTER 1

Citation

If you use this code in your research, please attribute this paper:

```
@article{blendz,
         author = {{Jones}, D.~M. and {Heavens}, A.~F.},
         title = "{Bayesian photometric redshifts of blended sources}",
         journal = {ArXiv e-prints},
         archivePrefix = "arXiv",
         eprint = {1808.02846},
         keywords = {Astrophysics - Cosmology and Nongalactic Astrophysics,␣
↪Astrophysics - Astrophysics of Galaxies},
         year = 2018,
         month = aug,
         adsurl = {http://adsabs.harvard.edu/abs/2018arXiv180802846J},
         adsnote = {Provided by the SAO/NASA Astrophysics Data System}
         }
```

CHAPTER 2

Documentation Contents

2.1 Installation

2.1.1 Easy installation

`blendz` can be installed from the command line using pip:

```
pip install blendz
```

`blendz` requires `numpy` and `scipy` to run, two common packages in scientific python code. These are easily installed using the Anaconda python distribution if you're not already a python user.

This allows you to use `blendz` straight away by installing Nestle, a pure Python implementation of the Nested Sampling algorithm. While this is easier to install than Multinest, photo-z runs will be slower. If you have a large number of galaxies you want to analyse, you should install Multinest using the instructions below.

2.1.2 Installing from source

To download and install `blendz` from source instead, clone the repository and install from there:

```
git clone https://github.com/danmichaeljones/blendz.git
cd blendz
python setup.py install
```

Downloading from source allows you to run the tests, which require `pytest`.

```
pip install pytest
cd path/to/blendz
pytest
```

2.1.3 Installing Multinest

`blendz` uses the PyMultinest library to run Multinest. To enable Multinest sampling in `blendz`, you need to install both of these libraries yourself.

Detailed instructions to install these can be found here with additional details for installing on macOS available here.

It is recommended you ensure that you have MPI and `mpi4py` working before installing Multinest to enable parallel sampling which can greatly increase the speed of photo-z runs. Try installing `mpi4py`:

```
pip install mpi4py
```

and test:

```
mpiexec -n 1 python -m mpi4py.bench helloworld
```

If you need to install an MPI library, you can do install openMPI on Linux using

```
sudo apt-get install openmpi-bin libopenmpi-dev
```

and on macOS using MacPorts by

```
sudo port install openmpi
```

2.1.4 Common errors

Could not load Multinest library

If you see an error like

```
ERROR:   Could not load MultiNest library "libmultinest.so"
ERROR:   You have to build it first, and point the LD_LIBRARY_PATH environment␣
↪variable to it!
```

this is because PyMultinest cannot find the Multinest library. If you installed Multinest in the folder

```
path/to/Multinest
```

the following command

```
export LD_LIBRARY_PATH="path/to/MultiNest/lib:$LD_LIBRARY_PATH"
```

will add Multinest to the path variable so that it can be found. To avoid having to run this every time you open a new terminal window, you should add this line to your terminal startup file (*.bashrc* on Linux and *.bash_profile* on macOS). This can be done on Linux using

```
echo 'export LD_LIBRARY_PATH="path/to/MultiNest/lib:$LD_LIBRARY_PATH"' >> ~/.bashrc
```

and on macOS using

```
echo 'export LD_LIBRARY_PATH="path/to/MultiNest/lib:$LD_LIBRARY_PATH"' >> ~/.bash_
↪profile
```

Intel MKL fatal error

The following error

```
Intel MKL FATAL ERROR: Cannot load libmkl_mc.so or libmkl_def.so
```

seems to be problem related to Anaconda's packaging of the MKL library. Forcing a reinstallation of `numpy` by

```
conda install -f numpy
```

can sometimes fix it. For more information, see this discussion on GitHub.

2.2 Getting started

For most normal uses of `blendz`, the only class you should need is `blendz.Photoz`. This is designed to be the only user-facing class.

The order of things to do to use `blendz` is as follows:

- *Setting the configuration* - done either using configuration files or keyword arguments.
- *Prior calibration* - The prior parameters can be set manually in the configuration, or using the prior calibration procedure. The output of the default calibration procedure is another configuration file that can be read in, containing the prior parameters.
- *Running the inference and model comparison* - After running the nested sampling for each number of components under consideration, the posterior samples and blend probabilities are available for analysis.

2.3 Setting the configuration

```
import blendz
```

Classes in `blendz` use a `Configuration` object instance to manage all of their settings. These *can* be created directly by instantiating the class, and passed to classes that require them using the `config` keyword argument:

```
cfg = blendz.Configuration(configuration_option='setting_value')
templates = blendz.fluxes.Templates(config=cfg)
```

However, constructing the configuration like this is usually not necessary. The `photoz` class is designed as the only user-facing class and handles the configuration for all of the classes it depends on. Instead, there are two recommended ways of setting the configuration:

2.3.1 Pass keyword arguments to classes

The configuration can be set programmatically by passing settings as keyword arguments:

```
pz = blendz.Photoz(data_path='path/to/data.txt',
                   mag_cols = [1, 2, 3, 4, 5],
                   sigma_cols = [6, 7, 8, 9, 10],
                   ref_band = 2,
                   filters=['sdss/u', 'sdss/g',
                            'sdss/r', 'sdss/i', 'sdss/z'])
```

2.3.2 Read in a configuration file

Configurations can also be read in from a file (or multiple files) by using the `config_path` keyword argument.

`config_path` should either be a string of the absolute file path to the configuration file to be read, or a list of strings if you want to read multiple files.

```
path1 = join(blendz.RESOURCE_PATH, 'config/testRunConfig.txt')
path2 = join(blendz.RESOURCE_PATH, 'config/testDataConfig.txt')

data = blendz.Photoz(config_path=[path1, path2])
```

2.3.3 Configuration file format

Configuration files are INI-style files read using the configparser module of the standard python library. These consist of `key = value` pairs separated by either a = or `:` separator. Whitespace around the separator is optional.

A few notes about their format:

- Configuration options *must* be separated into two (case-sensitive) sections, `[Run]` and `[Data]`. An explanation of all possible configuration options, split by these sections can be found on the *All configuration options* page.
- Comments can be added to configuration files using #
- If you want to use default settings, leave that option out of the coniguration file. Don't just leave an option blank after the =/`:` separator.
- Multiple configuration files can be loaded at once. While this provides a simple way to separate `[Run]` and `[Data]` settings (e.g., for running the same analysis on different datasets), options can be spread over different files however you want, provided that each setting is within its correct section.

An example of a configuration file (leaving some settings as default) is given below.

```
[Data]

data_path = path/to/datafile.txt
mag_cols = 1, 3, 5, 7, 9
sigma_cols = 2, 4, 6, 8, 10
ref_band = 2
filters = sdss/u, sdss/g, sdss/r, sdss/i, sdss/z
zero_point_errors = 0.01, 0.01, 0.01, 0.01, 0.01


[Run]

z_hi = 2
ref_mag_lo = 20
ref_mag_hi = 32
template_set = BPZ6
```

2.4 Prior calibration

If the prior parameters are not set manually by *Setting the configuration*, they can be estimated using a set of calibration data - unblended sources with photometric fluxes and known spectroscopic redshifts.

The `blendz.Photoz.calibrate` function just calls the prior-specific calibration function. As a result, if you define your own priors (see *Specifying new priors*), you will need to write your own calibration function.

The calibration function can be called by creating a `Photoz` object with configuration set to the calibration data. In this configuration, the `prior_parameters` option should be set to `None`.

```
pz_calib = blendz.Photoz(config_path='calibration_config.txt')
pz_calib.calibrate()
```

For the default priors, this will result in a file called `calibrated_prior_config.txt` being created. This is a configuration file with the prior parameters set to the maximum *a posteriori* parameters found in the calibration. This can then be read in alongside a photoz-configuration for sampling as normal.

```
pz = blendz.Photoz(config_path=['photoz_config.txt',
                                'calibrated_prior_config.txt'])

pz.sample([2, 1])
```

The photoz-configuration file should also have the `prior_parameters` option set to `None`.

2.5 Running the inference and model comparison

2.5.1 Sampling

Once the `Photoz` object is instantiated, e.g., with a configuration file,

```
pz = blendz.Photoz(data_path='path/to/data.txt',
                   mag_cols = [1, 2, 3, 4, 5],
                   sigma_cols = [6, 7, 8, 9, 10],
                   ref_band = 2,
                   filters=['sdss/u', 'sdss/g',
                            'sdss/r', 'sdss/i', 'sdss/z'])
```

the inference can be run for each number of components you'd like to analyse (e.g., compare between the single components and two-component blend cases), you need to sample. This can be done by calling the `sample` function, which accepts either an `int` or a `list` of `int` for the components to sample, e.g.,

```
pz.sample([1, 2], save_path='photoz_out.pkl', save_interval=1)
```

This excerpt also shows the saving feature, which saves `Photoz` object to file every `save_interval` sources, and once all sources are analysed. These save files can be loaded when creating the `Photoz` instance by

```
pz = blendz.Photoz(load_state_path='photoz_out.pkl')
```

2.5.2 Running in parallel

The inference can be run in parallel by saving a script to file (e.g., the code above into a file `photoz_run.py`) and running with MPI:

```
mpiexec python photoz_run.py
```

This requires both MPI and MultiNest be manually installed - see *Installation*.

2.5.3 Analyse the inference results

After the sampling run is complete, the posterior samples can be accessed using the `chain` method, e.g.

```
pz.chain(2, galaxy=0)
```

returns the samples from the two-component posterior for galaxy 0. If the optional keyword argument is omitted, a list of chains is returned with one array for each source.

A variety of summery statistic functions are also provided, such as the mean of each parameter

```
pz.mean(2, galaxy=0)
```

and the maximum *a posteriori* value for each parameter,

```
pz.max(2, galaxy=0)
```

Again, the `galaxy` argument is optional, and omitting it will return an array of shape `(num_galaxies, num_components * 2)`.

2.5.4 Model comparison

The model comparison results can be accessed using the `logbayes` function, e.g.,

```
pz.logbayes(2, 1, galaxy=0)
```

will return the Bayes factor for comparison between the two-component blend and single source cases. A model comparison prior can be included by multiplying this value. If the `galaxy` argument is omitted, an array of `float`, one for each source, is returned.

2.6 Specifying filters

2.6.1 Included filters

The `blendz` installation includes filter response curves from several instruments. The `filter_path` configuration option points to the folder where these are stored by default, and so they can be specified in the `filters` configuration option using the following names:

SDSS

Described in Stoughton et al. (2002)

Configuration option	Description
`sdss/u`	u
`sdss/g`	g
`sdss/r`	r
`sdss/i`	i
`sdss/z`	z

Viking

Described in Edge et al. (2013)

Configuration option	Description
`viking/h`	H
`viking/j`	J
`viking/k`	K
`viking/y`	Y

LSST

Described in Abell et al.(2009)

Configuration option	Description
`lsst/u`	u
`lsst/g`	g
`lsst/r`	r
`lsst/i`	i
`lsst/z`	z
`lsst/y`	Y

HST

As packaged in BPZ (Benitez (2000))

Configuration option	Description
`hst/F110W`	F110W
`hst/F160W`	F160W
`hst/F435W`	F435W
`hst/F606W`	F606W
`hst/F775W`	F775W
`hst/F850LP`	F850LP

2.6.2 Loading custom filters

Each custom filter can be specified by a single plaintext file of two columns, wavelength (Angstroms) in the first, and the filter response in the second, separated by whitespace, e.g.

```
912.0  0.0329579004203
920.0  0.0332336431181
930.0  0.0335731230922
940.0  0.033939398051
950.0  0.0342922864396
960.0  0.0346317644112
970.0  0.0349582358084
980.0  0.0352716948728
990.0  0.0355717998991
1000.0  0.0358573156287
1010.0  0.0361306346606
```

The `filter_path` configuration option should point to the folder where these files are stored, and each entry in the `filters` configuration option should be the path (including the name and file extension) relative to `filter_path`, i.e., for the following folder layout:

```
containing_folder/
├── filters/
│   ├── instrument_one/
│   │   ├── filter_one.txt
│   │   ├── filter_two.txt
│   ├── instrument_two/
│   │   ├── filter_three.txt
│   │   ├── filter_four.txt
```

you should set the configuration options to:

```
filter_path = "containing_folder/filters"
filters = ["instrument_one/filter_one.txt", "instrument_one/filter_two.txt", \
           "instrument_two/filter_three.txt", "instrument_two/filter_four.txt"]
```

The default filters only don't need file extensions as they are saved in plaintext files without file extensions.

2.7 Specifying templates

Templates are specified in `blendz` using two different types of file, the template file itself, and a template set. Template sets are configuration files containing the name, type and filepath of a collection of templates so that the whole set can be specified as a single configuration option.

2.7.1 Included templates

The `blendz` installation includes the 8 templates from BPZ. The `template_set_path` configuration option points to the folder where these are stored by default. As a result, these can be easily used by setting the `template_set` configuration option to either `BPZ8` or `BPZ6`, where the latter excludes the two starburst templates added to BPZ by Coe et al. 2006.

A set of only a single template can also be specified using one of the following options:

`single/El_B2004a`	`single/Sbc_B2004a`
`single/Scd_B2004a`	`single/Im_B2004a`
`single/SB2_B2004a`	`single/SB3_B2004a`
`single/ssp_5Myr_z008`	`single/ssp_25Myr_z008`

2.7.2 Loading custom templates

If you want to supply your own templates, you need to create a template set file. This is a configuration file containing the following information:

- The unique name of every template
- The path to the file specifying the template *relative to the location of the template set file*.
- The galaxy type a string to group templates together in the prior.

An example layout of a template set is given below:

```
[name_of_template_1]
path = path/to/template1.txt
type = early

[name_of_template_2]
path = path/to/template1.txt
type = late
```

When using custom templates, the configuration option `template_set_path` should point to the folder containing your ntemplate set file, and `template_set` should be the full filename, including the file extension.

Each template refered to in the template set file is then specified by a plaintext file of two columns, wavelength (Angstroms) in the first, and spectral flux density in the second, separated by whitespace, e.g.

```
912.0  0.0329579004203
920.0  0.0332336431181
930.0  0.0335731230922
940.0  0.033939398051
950.0  0.0342922864396
960.0  0.0346317644112
970.0  0.0349582358084
980.0  0.0352716948728
990.0  0.0355717998991
1000.0  0.0358573156287
1010.0  0.0361306346606
```

2.8 All configuration options

Below are all of the possible configuration settings. When being set by a *configuration file*, they should be given as described in *Setting the configuration*, split by `[Data]` and `[Run]`. When being set as *keyword arguments*, this split is not necessary, but each option should be passed as the correct type.

If you do not set an option, the default value is taken instead. Options with a *N/A* default value are not optional and must be set by you.

2.8.1 Data options

Configuration option	Explanation	Default	Python type
data_path	Absolute path to the file containing your photometry.	*N/A*	`str`
skip_data_rows	Number of columns to ignore at the top of the data file.	0	`int`
data_is_csv	Flag of whether data is comma-separated. If `False`, the data file is assumed to be whitespace separated.	`False`	`bool`
filter_path	Absolute path to the folder where the files describing your filters are stored.	Filter folder in the included resources.	`str`
mag_cols	List of the columns in the data-file where the AB-magnitude of the source in each band is. Indices start at zero.	*N/A*	`list` of `int`
sigma_cols	List of the columns in the data-file where the error on the AB-magnitude of the source in each band is. Indices start at zero.	*N/A*	`list` of `int`
spec_z_col	The column in the data-file where the spectroscopic redshift of the source is. If it's not present in the data-file, set to `None`. Indices start at zero.	`None`	`int` *or* `None`
ref_band	Index of the filter band (of the list of filters, *not* the data file column) that is considered the reference band, the band the priors are conditioned on, and where the soure magnitude is sampled. Indices start at zero.	*N/A*	`int`
filters	List of paths to the filter files, relative to `filter_path`, *with* file extensions. The included filters are saved in files without a file extension.	*N/A*	`list` of `str`
zero_point_errors	List of errors on the zero point calibration of each filter band.	*N/A*	`list` of `float`
magnitude_limit	Value of the survey magnitude limit, fixed for all galaxies. One of `magnitude_limit` or `magnitude_limit_col` must be set. If both are set. `magnitude_limit` is ignored.	*N/A*	`float`
magnitude_limit_col	Value of the survey magnitude limit, set individually for each galaxy. One of `magnitude_limit` or `magnitude_limit_col` must be set. If both are set. `magnitude_limit_col` is preferred.	*N/A*	`int`
no_detect_value	Value of the data when an observation was made but the source was not detected.	99.0	`float`
no_observe_value	Value of the data when an observation was not made.	-99.0	`float`
angular_resolution	Angular resolution of the data. Sources with a smaller angular separation than this are assumed to be blended (for the correlation function). In units of arcseconds.	*N/A*	`float`

2.8.2 Run options

Configuration option	Explanation	Default	Python type
z_lo	Minimum redshift to sample.	0	`float`
z_hi	Maximum redshift to sample.	10	`float`
z_len	Length of redshift grid to calculate functions of redshift on before interpolating.	1000	`int`
ref_mag_lo	Minimum magnitude to sample (numerically, i.e. the *brightest* magnitude).	*N/A*	`float`
ref_mag_hi	Fixed maximum magnitude to sample (numerically, i.e. the *dimmest* magnitude). One of `ref_mag_hi` or `ref_mag_hi_sigma` must be set. If both are set, `ref_mag_hi` is ignored.	*N/A*	`float`
ref_mag_hi_sigma	Maximum magnitude to sample (numerically, i.e. the *dimmest* magnitude) in terms of reference band flux error. One of `ref_mag_hi` or `ref_mag_hi_sigma` must be set. If both are set, `ref_mag_hi_sigma` is preferred.	*N/A*	`float`
template_set_path	Absolute path to the folder containing the template set file, as described in *Specifying templates*. Templates in the template set file are specified with a path relative to this.	Template folder in the included resources.	`str`
template_set	File name of the template set file.	`BPZ8` - The set of 8 templates from BPZ	`str`
sort_redshifts	Whether to use redshift sorting to break the exchangability of blended posteriors. If `False`, magnitude sorting is used.	`True`	`bool`
omega_mat	Omega-matter cosmological parameter.	0.3065	`float`
omega_lam	Omega-lambda cosmological parameter.	0.6935	`float`
omega_k	Omega-k cosmological parameter.	0.	`float`
hubble	Hubble constant in km/s/Mpc.	67.9	`float`
r0	Constant of proportionality in correlation function	5.	`float`

2.9 Specifying new priors

You can specify new priors by subclassing `blendz.model.ModelBase`, redefining the four functions that return the priors and instantiating your new model for `blendz.photoz`.

2.9.1 Creating a new model class

Your new class should have the following basic layout:

```
from blendz.model import ModelBase

class MyNewModel(ModelBase):

    #Optional:
    def __init__(self, new_prior_params, **kwargs):
        #Run the setup defined in ModelBase
        super(MyNewModel, self).__init__(**kwargs)
        #Do some other setup with your new_prior_parameters

    #Mandatory:
    def lnPrior(self, redshift, magnitude):
        #Definition of P(z_a, t_a, m_0a) for all t_a
        return 0.

    #Optional:
    def correlationFunction(self, redshifts):
        #Definition of xi({z})
        return 0.

    #Optional:
    def calibrate(self, photometry, cached_likelihood, **kwargs):
        return 0.
```

A few things to note:

- The `__init__` function is optional but allows you to define additional setup tasks that are done when your model is instantiated. It is important you call the superclass `__init__` if you define this.
- The `correlationFunction` function is also optional. The function `self.comovingSeparation(z_lo, z_hi)` defined in `ModelBase` may be helpful.
- While `__init__` is optional, you **must** redefine `lnPrior`. This function takes a `float` for both the redshift and magnitude, and returns a `numpy.array` of the natural log of the prior for each template *type* (not each template). The `self.possible_types` attribute is a list of the possible types, where each element is a string with the name of that type. These are automatically read from the template set file supplied at runtime.
- The `**kwargs` get passed by `ModelBase` to `Configuration`, allowing you to edit the configuration like other `blendz` classes using keyword arguments.
- The `redshift`, `magnitude` and `component_ref_mag` arguments passed to natural-log prior functions are floats, while the `redshifts` argument in `correlationFunction` is a 1D `numpy` array.
- The `calibrate` function is also optional. This takes as arguments a `blendz.photometry.Photometry` object and a `numpy.array` of shape `(num_galaxies, num_templates)` filled with the likelihood. This function is called by the `blendz.Photoz.calibrate(**kwargs)` function, with any keyword arguments passed to the function here. There is no return value for this function; the default model writes the resulting parameters to a configuration file that can be read by `blendz.Photoz`.

2.9.2 Using the new model

The new model can simply be instantiated and passed to `blendz.Photoz` as a keyword argument.

```
new_model = MyNewModel(new_prior_params=42, template_set='BPZ6')

pz = blendz.Photoz(model=new_model)
```

2.10 blendz API documentation

The `blendz.Photoz` class is designed to be the only user-facing class and has methods for each step of the photo-z analysis.

class blendz.**Photoz**(*model=None*, *photometry=None*, *config=None*, *load_state_path=None*, ***kwargs*)

applyToMarginals(*func*, *num_components*, *galaxy=None*, ***kwargs*)

Apply a function to the 1D marginal distribution samples of each parameter.

Args:

func (function): The function to apply to the marginal distribution samples. It should accept an array of the samples as its first argument, with optional keyword arguments.

num_components (int): Number of components.

galaxy (int or None): Index of the galaxy to apply the function to. If None, return array with a row for each galaxy. Defaults to None.

****kwargs:** Any optional keyword arguments to pass to the function.

chain(*num_components*, *galaxy=None*)

Return the (unweighted) posterior samples.

Args:

num_components (int): Number of components.

galaxy (int or None): Index of the galaxy to calculate log-evidence for. If None, return array of log-evidence for every galaxy. Defaults to None.

logbayes(*m*, *n*, *base=None*, *galaxy=None*)

Return the log of the Bayes factor between m and n components, log[B_mn].

A positive value suggests that that evidence prefers the m-component model over the n-component model.

Args:

m (int): First number of components.

n (int): Second number of components.

base (float or None): Base of the log to return. If None, uses natural log. Defaults to None.

galaxy (int or None): Index of the galaxy to calculate B_mn for. If None, return array of B_mn for every galaxy. Defaults to None.

logevd(*num_components*, *galaxy=None*, *return_error=False*)

Return the natural log of the evidence.

Args:

num_components (int): Number of components.

galaxy (int or None): Index of the galaxy to calculate log-evidence for. If None, return array of log-evidence for every galaxy. Defaults to None.

return_error (bool): If True, also return the error on the log-evidence. If galaxy is None, this is also an array. Defaults to False.

max(*num_components*, *galaxy=None*, *bins=20*)

Return the maximum-a-posteriori point for the 1D marginal distribution of each parameter.

This is calculated by forming a 1D histogram of each marginal distribution and assigning the MAP of that parameter as the centre of the tallest bin.

Args:

num_components (int): Number of components.

galaxy (int or None): Index of the galaxy to calculate the MAP for. If None, return array with rows of MAPs for each galaxy. Defaults to None.

bins (int): Number of bins to use for each 1D histogram.

mean(*num_components*, *galaxy=None*)

Return the mean point for the 1D marginal distribution of each parameter.

Args:

num_components (int): Number of components.

galaxy (int or None): Index of the galaxy to calculate the MAP for. If None, return array with rows of means for each galaxy. Defaults to None.

sample(*num_components*, *galaxy=None*, *nresample=1000*, *seed=False*, *measurement_component_mapping=None*, *npoints=150*, *print_interval=10*, *use_pymultinest=None*, *save_path=None*, *save_interval=None*)

Sample the posterior for a particular number of components.

Args:

num_components (int): Sample the posterior defined for this number of components in the source.

galaxy (int or None): Index of the galaxy to sample. If None, sample every galaxy in the photometry. Defaults to None.

nresample (int): Number of non-weighted samples to draw from the weighted samples distribution from Nested Sampling. Defaults to 1000.

seed (bool or int): Random seed for sampling to ensure deterministic results when ampling again. If False, do not seed. If True, seed with value derived from galaxy index. If int, seed with specific value.

measurement_component_mapping (None or list of tuples): If None, sample from the fully blended posterior. For a partially blended posterior, this should be a list of tuples (length = number of measurements), where each tuples contains the (zero-based) indices of the components that measurement contains. Defaults to None.

npoints (int): Number of live points for the Nested Sampling algorithm. Defaults to 150.

print_interval (int): Update the progress bar with number of posterior evaluations every print_interval calls. Defaults to 10.

save_path (None or str): Filepath for saving the Photoz object for reloading with *Photoz.loadState*. If None, do not automatically save. If given, the Photoz object will be saved to this path after all

galaxies are sampled. If save_interval is also not None, the Photoz object will be saved to this path every save_interval galaxies. Defaults to None.

save_interval (None or int) If given and save_path is not None, the Photoz object will be saved to save_path every save_interval galaxies. Defaults to None.

use_pymultinest (bool or None) If True, sample using the pyMultinest sampler. This requires PyMultiNest to be installed separately. If False, sample using the Nestle sampler, which is always installed when blendz is. If None, check whether pyMultinest is installed and use it if it is, otherwise use Nestle. Defaults to None.

saveState(*filepath*)
Save this entire Photoz instance to file.

This saves the exact state of the current object, including all data and any reults from sampling.

Args: filepath (str): Path to file to save to.

std(*num_components*, *galaxy=None*)
Return the standard deviation for the 1D marginal distribution of each parameter.

Args:

num_components (int): Number of components.

galaxy (int or None): Index of the galaxy to calculate the MAP for. If None, return array with rows of means for each galaxy. Defaults to None.

Index