pydftools Documentation

Release 0.1.0

Steven Murray

Oct 25, 2017

Contents

A pure-python port of the `dftools` R package.

This package attempts to imitate the `dftools` package (repo: https://github.com/obreschkow/dftools) quite closely, while being as Pythonic as possible. Do note that 2D+ models are not yet implemented in this Python port, and neither are non-parametric models. Hopefully they will be along soon.

From `dftool`'s description:

> This package can find the most likely P parameters of a D-dimensional distribution function (DF) generating N objects, where each object is specified by D observables with measurement uncertainties. For instance, if the objects are galaxies, it can fit a MF (P=1), a mass-size distribution (P=2) or the mass-spin-morphology distribution (P=3). Unlike most common fitting approaches, this method accurately accounts for measurement is uncertainties and complex selection functions. A full description of the algorithm can be found in Obreschkow et al. (2017).

In short, clean out Eddington bias from your fits:

- Free software: MIT license
- Documentation: https://pydftools.readthedocs.io.

CHAPTER 1

Features

- Simple and fast parameter fitting for generative distribution functions
- Several examples (with astronomical applications in mind)
- Several plotting routines so that you can go from nothing to a plot in minutes
- A `mockdata()` function which can produce data to fit.
- Support for arbitrary 1D models, several kinds of selection functions, jackknife and bootstrap resampling, Gaussian error estimation and more.

CHAPTER 2

Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 3

Contents

Installation

Stable release

To install pydftools, run this command in your terminal:

```
$ pip install pydftools
```

This is the preferred method to install pydftools, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for pydftools can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/steven-murray/pydftools
```

Or download the tarball:

```
$ curl  -OL https://github.com/steven-murray/pydftools/tarball/master
```

Once you have a copy of the source, you can install it from the top-level directory with:

```
$ pip install .
```

Examples

To help get you started using `pydftools`, we've compiled a few simple examples. Other examples can be found in the API documentation for each object or by looking at some of the tests.

Basic Example

This example is a basic introduction to using `pydftools`. It mimics example 1 of `dftools`.

```
In [1]: # Import relevant libraries
        %matplotlib inline

        import pydftools as df
        import time

         # Make figures a little bigger in the notebook
        import matplotlib as mpl
        mpl.rcParams['figure.dpi'] = 120

        # For displaying equations
        from IPython.display import display, Markdown
```

Choose some parameters to use throughout

```
In [2]: n = 1000
        seed = 1234
        sigma = 0.5
        model =df.model.Schechter()
        p_true = model.p0
```

Generate mock data with observing errors:

```
In [3]: data, selection, model, other = df.mockdata(n = n, seed = seed, sigma = sigma, model=model, v
```

```
Number of sources in the mock survey (expected): 1000.000
Number of sources in the mock survey (selected): 1000
```

Create a fitting object (the fit is not performed until the `fit` object is accessed):

```
In [4]: survey = df.DFFit(data=data,  selection=selection, model=model)
```

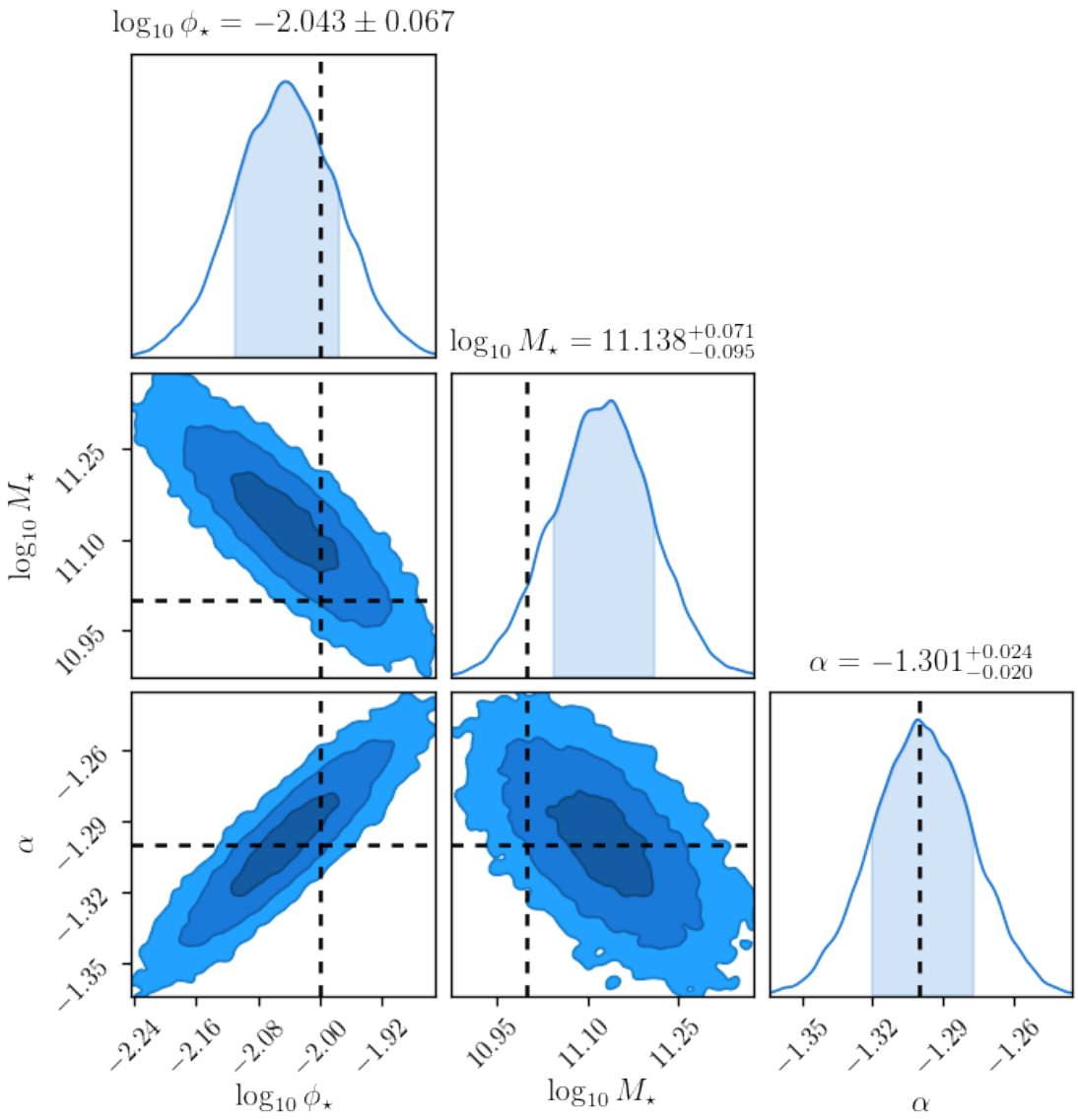
Perform the fit and get the best set of parameters:

```
In [5]: start = time.time()
        print(survey.fit.p_best)
        print("Time for fitting: ", time.time() - start, " seconds")
```

```
[ -2.04370588  11.12540248  -1.29867552]
Time for fitting:  0.47102880477905273  seconds
```

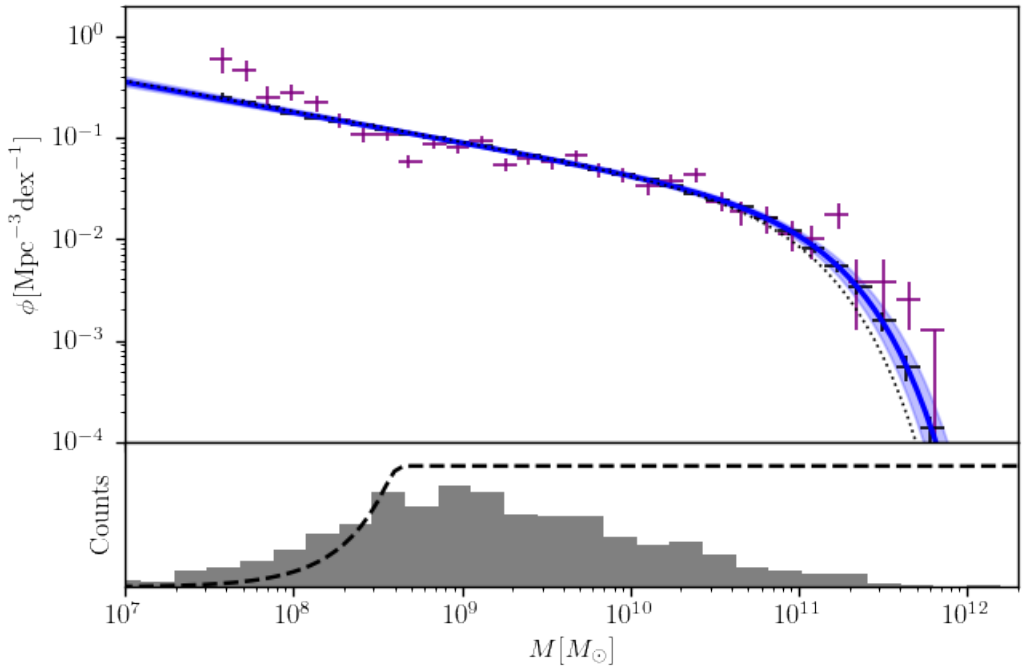
Plot the covariances:

```
In [6]: fig = df.plotting.plotcov([survey], p_true=p_true, figsize=1.3)
```



Plot the mass function itself:

```
In [7]: fig, ax = df.mfplot(survey, xlim=(1e7,2e12), ylim=(1e-4,2), p_true = p_true, bin_xmin=7.5, bi
```



Write out fitted parameters with (Gaussian) uncertainties:

```
In [8]: display(Markdown(survey.fit_summary(format_for_notebook=True)))
```

$\frac{dN}{dV\,dx} = \log(10)\phi_\star \mu^{\alpha+1} \exp(-\mu)$, where $\mu = 10^{x-\log_{10} M_\star}$ $\log_{10}\phi_\star$ = -2.044 (+-0.066) $\log_{10} M_\star$ = 11.125 (+-0.082) α = -1.299 (+-0.021)

Tutorial 1

This is a direct port of the R ‘‘dftools‘‘ ‘tutorial <http://rpubs.com/obreschkow/312101>‘__ to Python.

Objective of tutorial: Illuystrate the basic functionality of `pydftools` by reproducing the HI mass function in Fig. 7 of Westmeier et al. 2017 (https://arxiv.org/pdf/1709.00780.pdf).

Load the relevant libraries:

```
In [1]: %matplotlib inline

        import pydftools as df
        from pydftools.plotting import mfplot
        import numpy as np
        from urllib.request import Request, urlopen # For getting the data online

        from IPython.display import display, Math, Latex, Markdown, TextDisplayObject
```

Download the HI-mass data of Westmeier et al. 2017:

```
In [2]: req = Request('http://quantumholism.com/dftools/westmeier2017.txt', headers={'User-Agent': 'M
        data  = urlopen(req)

        data = np.genfromtxt(data, skip_header=1)
```

There are 31 galaxies in this sample, hence the array has 31 rows. This data can be recast into the log-masses x, normally used by `pydftools`. We assume the mass uncertainties to be normal in x and determine their amplitude using linear error propagation. We also define the vector of effective volumes:

```
In [3]: x = np.log10(data[:,0])
        x_err = data[:,1]/data[:,0]/np.log(10)
        veff_values = data[:,2]
```

Now fit these data. We first must create a Data and Selection object:

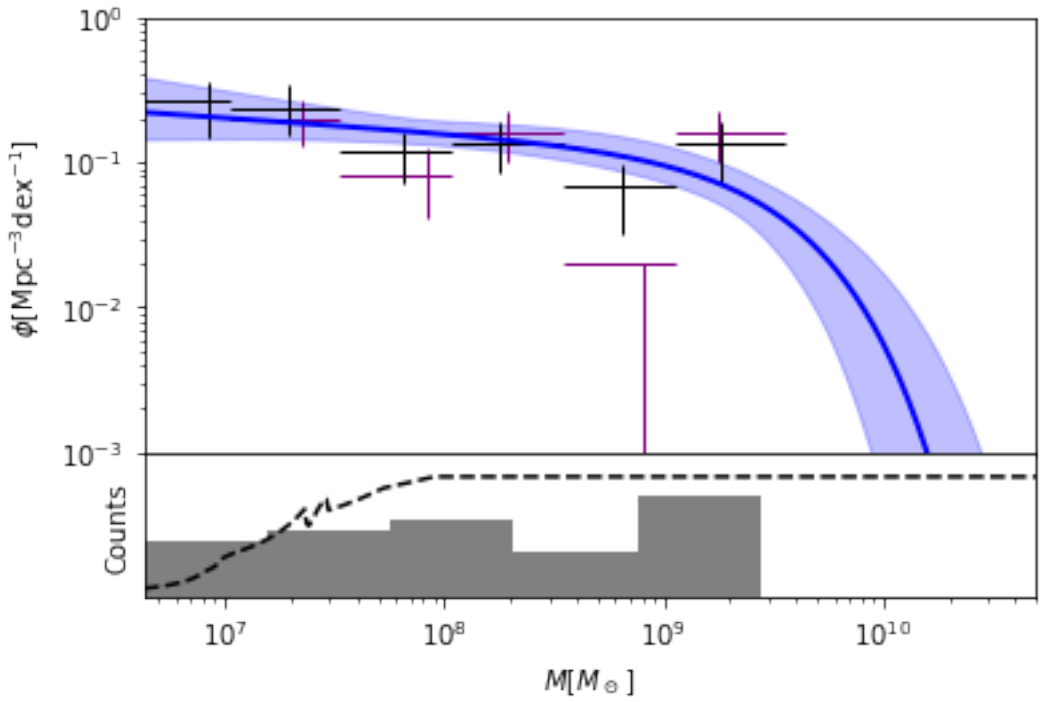
```
In [4]: data = df.Data(x = x, x_err=x_err)
        selection = df.selection.SelectionVeffPoints(veff=veff_values, xval = x, xmin = 5, xmax = 13)
```

```
Warning: xmin returns Veff(xmin)=0, setting xmin, xmax to 6.63363363363, 13.0
```

```
In [5]: survey = df.DFFit(data = data, selection=selection, grid_dx = 0.01)
```

```
In [6]: mfplot(survey, xlim=(10**6.63, 5e10), ylim=(1e-3, 1), show_bias_correction=False);
```

```
/home/steven/Documents/Projects/DFTOOLS/pydftools/pydftools/plotting.py:401: RuntimeWarning: divide b
  bin['gdf_input'] = np.bincount(x_bins, weights=1 / bin['dx'] / v)
/home/steven/Documents/Projects/DFTOOLS/pydftools/pydftools/dffit.py:871: RuntimeWarning: invalid val
  self.grid.effective_counts = rho_unbiased ** 2 / rho_unbiased_sqr  # this equation gives the effect
/home/steven/anaconda2/envs/dftools/lib/python3.6/site-packages/matplotlib/axes/_axes.py:2951: Runtim
  low = [thisx - thiserr for (thisx, thiserr)
```



and show the fitted parameters:

```
In [7]: display(Markdown(survey.fit_summary(format_for_notebook=True)))
```

$\frac{dN}{dV\,dx} = \log(10)\phi_\star \mu^{\alpha+1} \exp(-\mu)$, where $\mu = 10^{x-\log_{10} M_\star}$$\log_{10} \phi_\star$ = -1.315 (+-0.275)$\log_{10} M_\star$ = 9.540 (+-0.307)α = -1.102 (+-0.147)

The dashed line in the bottom panel shows the effective volume as a function of mass, recovered from the 31 values of `veff`. By default an effective volume of 0 for masses smaller than the smallest observed mass, and identical to the maximum volume for masses larger than the largest observed mass. If a better model is available from survey-specific considerations, then this information can be exploited to improve the fit. In this example, we replace the assumption of `veff=0` for `x<xmin`, by `veff=max(0,(x6.53)75)`:

```
In [8]: def veff_extrap(x):
            veff_max = np.max(veff_values)
```

```
            return np.clip((x-6.53)*75, 0,veff_max)

        selection = df.selection.SelectionVeffPoints(veff=veff_values, xval = x, veff_extrap=veff_ext
```

```
Warning: xmin returns Veff(xmin)=0, setting xmin, xmax to 6.53753753754, 13.0
```

Now fit again:

```
In [9]: survey = df.DFFit(data = data, selection=selection, grid_dx = 0.01)
```

and see the best fit solution:

```
In [10]: display(Markdown(survey.fit_summary(format_for_notebook=True)))
```

$\frac{dN}{dV\,dx} = \log(10)\phi_\star \mu^{\alpha+1} \exp(-\mu)$, where $\mu = 10^{x-\log_{10} M_\star}$ $\log_{10} \phi_\star$ = -1.308 (+-0.272) $\log_{10} M_\star$ = 9.535 (+-0.305) α = -1.097 (+-0.146)

As can be seen the parameters have change very slightly due to the modified effecive volume at the lowest masses. The printed parameters have symmetric Gaussian uncertainties, determined in the Lapace approximation (i.e. by inverting the Hessian matrix of the modified likelihood function). To allow for non-Gaussian parameter posteriors, we now refit the data while bootstrapping it 10^3 times:

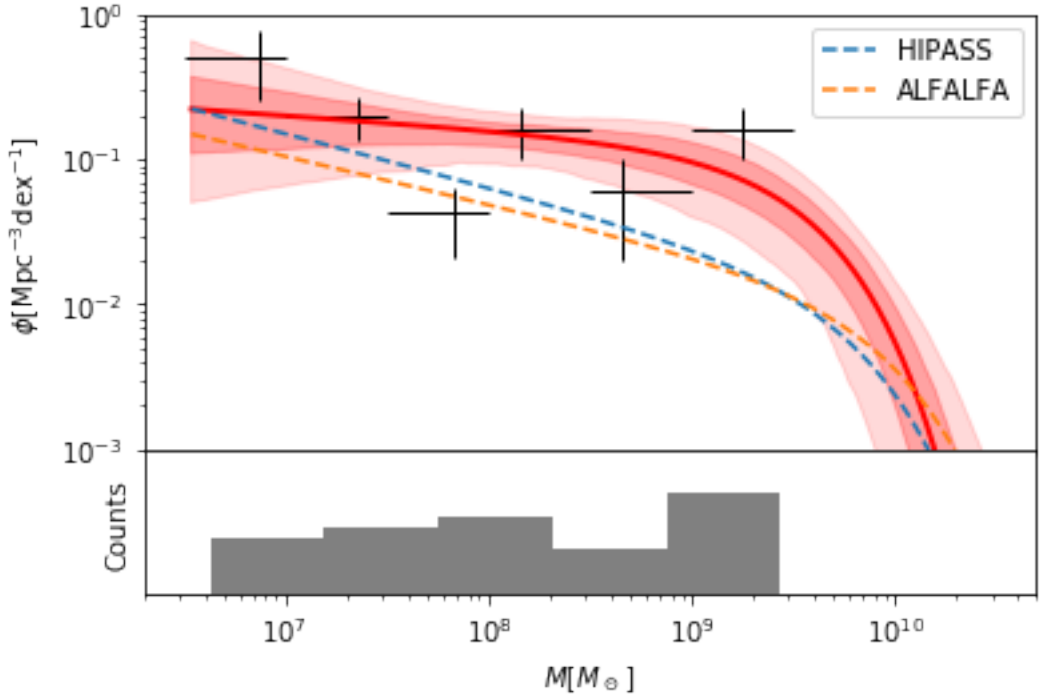
```
In [11]: survey.resample(n_bootstrap = 1000)
```

Finally, let's produce the plot with 68% and 95% confidence regions around the best fit. Also change fit color to red, change data color to black, remove posterior data, remove effective volume line, and adjust binning of input data. Then, add HIPASS and ALFALFA lines.

```
In [17]: fig, ax = mfplot(survey,xlim=(2e6,5e10),ylim=(1e-3,1),uncertainty_type=3,
                           col_fit='red',col_data='black',show_posterior_data=False,
                           ls_veff='none', nbins=6,bin_xmin=6.5,bin_xmax=9.5,
                           show_bias_correction=False,
                           xpower10=True)
         x = survey.grid.x[0]

         ax[0].plot(10**x, survey.model.gdf(x,[np.log10(6.0e-3),9.80,-1.37]), ls='--',lw=1.5, color=
         ax[0].plot(10**x, survey.model.gdf(x,[np.log10(4.8e-3),9.96,-1.33]), ls='--',lw=1.5, color=
         ax[0].legend()
```

```
Out[17]: <matplotlib.legend.Legend at 0x7ff160e0bf60>
```



and write the bes-fitting parameters:

```
In [18]: display(Markdown(survey.fit_summary(format_for_notebook=True)))
```

$\frac{dN}{dVdx} = \log(10)\phi_\star\mu^{\alpha+1}\exp(-\mu)$, where $\mu = 10^{x-\log_{10}M_\star}$$\log_{10}\phi_\star$ = -1.308 (+0.251 -0.266)$\log_{10}M_\star$ = 9.535 (+0.152 -0.175)α = -1.097 (+0.181 -0.140)

Note that there are marginal differences to the uncertainty ranges quoted in the publication, due to a difference in the bootstrapping technique used in the publication (parametric bootstrapping) and the current version of dftools (non-parametric bootstrapping).

Example With Large-Scale Structure (LSS)

This example shows how `pydftools` deals with LSS. It mimics example (2) of `dftools::dfexample`.

```
In [10]: # Import relevant libraries
         %matplotlib inline

         import pydftools as df
         from scipy.special import erf
         import numpy as np

         import time

         import matplotlib as mpl
         import matplotlib.pyplot as plt
         mpl.rcParams['figure.dpi'] = 120  # Make figures a little bigger in the notebook

         from IPython.display import display, Markdown
```

```
In [2]: # Survey parameters
        sigma = 0.3
        seed = 1
```

Let us create a selection function which involves LSS. We define a dew functions, namely, `f`, which is the selection function which gives the fraction of objects observed at any given combination of (x, r), `dVdr`, which is the radial derivative of the volume (typically proportional to r^2), and `g`, which is the relative over-/under- abundance of objects at distance r due to cosmic fluctuations:

```
In [3]: dmax = lambda x : 1e-3/np.sqrt(10) * np.sqrt(10**x)
        f = lambda x, r : erf((dmax(x) - r)/dmax(x)*20)*0.5 + 0.5
        dVdr = lambda r : 0.2 * r**2
        g = lambda r : 1 + 0.9*np.sin((r/100)**0.6*2*np.pi)
```

Now create our selection function object:

```
In [11]: selection = df.selection.SelectionRdep(xmin = 5, xmax = 13, rmin = 0, rmax = 100, f=f, dvdr=
```

```
Warning: xmin returns Veff(xmin)=0, setting xmin, xmax to 5.45645645646, 13.0
```

Use a Schechter distribution function:

```
In [13]: model = df.model.Schechter()
         p_true = model.p0
```

```
In [14]: data, selection, model, other = df.mockdata(seed = seed, sigma = sigma, model=model, selecti
```

```
Number of sources in the mock survey (expected): 561.057
Number of sources in the mock survey (selected): 561
```

Fit mock daa without any bias correction:

```
In [15]: selection_without_lss = df.selection.SelectionRdep(xmin = 5, xmax = 13, rmin = 0, rmax = 100
         survey1 = df.DFFit(data = data, selection = selection_without_lss, ignore_uncertainties=True
```

```
Warning: xmin returns Veff(xmin)=0, setting xmin, xmax to 5.45645645646, 13.0
```

Fit mock data while correcting for observational errors (Eddington bias):

```
In [16]: survey2 = df.DFFit(data = data, selection = selection_without_lss)
```

Fit mock data while correcting observational errors and LSS. We create a new (identical) Selection object, both to keep them conceptually different, and also because the object is mutable, and is changed when estimating the LSS function. If we use the same Selection object, the previous objects will be modified when fitting the following object.

```
In [17]: selection_est_lss = df.selection.SelectionRdep(xmin = 5, xmax = 13, rmin = 0, rmax = 100, f=
         survey3 = df.DFFit(data = data, selection = selection_est_lss, correct_lss_bias = True, lss_
```

```
Warning: xmin returns Veff(xmin)=0, setting xmin, xmax to 5.45645645646, 13.0
```

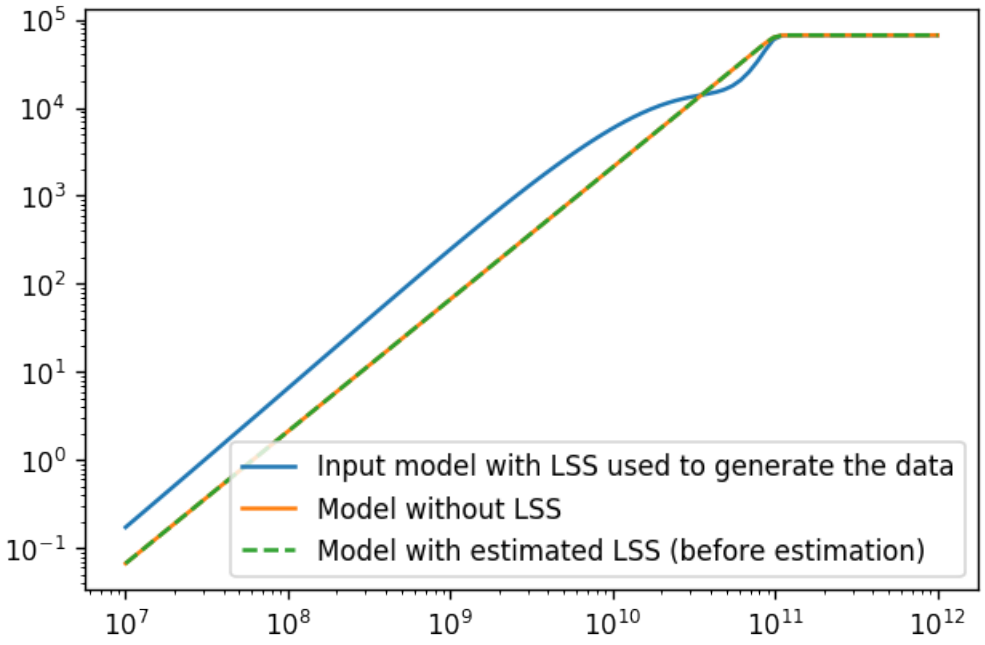
Recall that the fitting is not actually performed until the `fit` attribute is accessed. Let's plot the effective volume functions for each of our models *before fitting*:

```
In [18]: x = np.linspace(7, 12, 100)

         plt.plot(10**x, selection.Veff(x), label="Input model with LSS used to generate the data")
         plt.plot(10**x, survey1.selection.Veff(x), label="Model without LSS")
         plt.plot(10**x, survey3.selection.Veff(x), ls= '--', label="Model with estimated LSS (before

         plt.xscale('log')
         plt.yscale('log')
         plt.legend()
```

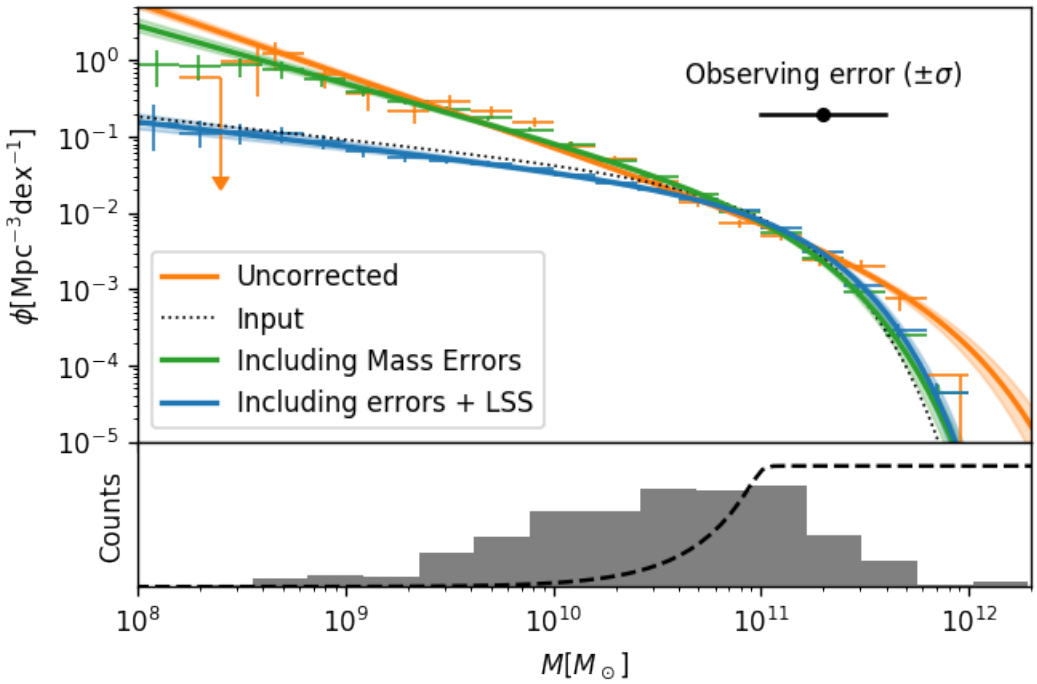
```
Out[18]: <matplotlib.legend.Legend at 0x7f7940477c88>
```



Now let's perform the fit and plot the fitted mass functions:

```
In [23]: fig, ax = df.mfplot(survey1, fit_label="Uncorrected", p_true=p_true, xlim=(1e8,2e12),ylim=(1
         fig, ax = df.mfplot(survey2, fit_label="Including Mass Errors", nbins=20, bin_xmin=8,bin_xma
         fig, ax = df.mfplot(survey3, fit_label="Including errors + LSS", nbins=20, bin_xmin=8,bin_xm

         ax[0].text(10**11.3, 5e-1, r"Observing error ($\pm \sigma$)",horizontalalignment='center')
         ax[0].plot([10**(11.3 -sigma), 10**(11.3+sigma)], [2e-1, 2e-1], color='k')
         ax[0].scatter([10**11.3], [2e-1], color='k', s=20)
```

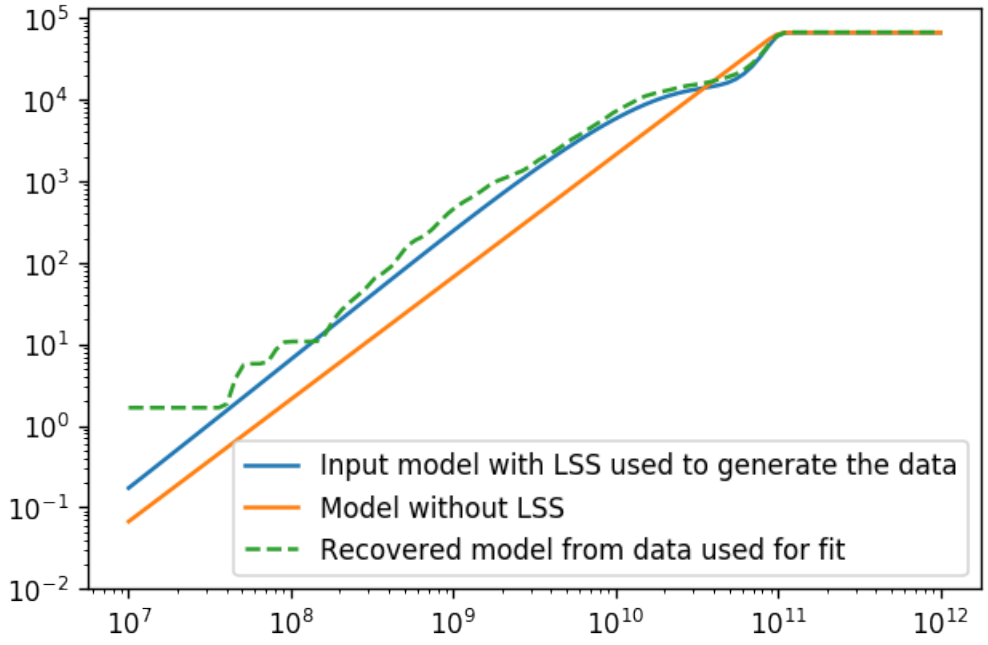


And again, have a look at the effective volumes (post-fitting):

```
In [12]: plt.plot(10**x, selection.Veff(x), label="Input model with LSS used to generate the data")
         plt.plot(10**x, survey1.selection.Veff(x), label="Model without LSS")
         plt.plot(10**x, survey3.selection.Veff(x), ls= '--', label="Recovered model from data used f

         plt.xscale('log')
         plt.yscale('log')
         plt.ylim(1e-2,)
         plt.legend()
```

```
Out[12]: <matplotlib.legend.Legend at 0x7fbbd888a4e0>
```



Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/steven-murray/pydftools/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

pydftools could always use more documentation, whether as part of the official pydftools docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/steven-murray/pydftools/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *pydftools* for local development.

1. Fork the *pydftools* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pydftools.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pydftools
$ cd pydftools/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pydftools tests
$ python setup.py test or py.test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/steven-murray/pydftools/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests.test_pydftools
```

Credits

Development Lead

- Steven Murray <steven.murray@curtin.edu.au>

Contributors

- Danail Obreschkow <danail.obreschkow@gmail.com>

History

0.1.0 (2017-10-25)

- First release on PyPI.
- All basic examples working as expected
- TravisCI, Readthedocs set up.
- Does not have multi-dimension support, or non-parametric support.

Indices and tables

- genindex
- modindex
- search