fitter Documentation

Release 1.3.0

Thomas Cokelaer

Jul 25, 2021

Contents

Compatible with Python 3.6, 3.7, and 3.8

CHAPTER 1

What is it ?

fitter package provides a simple class to identify the distribution from which a data samples is generated from. It uses 80 distributions from Scipy and allows you to plot the results to check what is the most probable distribution and the best parameters.

CHAPTER 2

Installation

```
pip install fitter
```

fitter is also available on **conda** (bioconda channel):

```
conda install fitter
```

And as a singularity file in *damona <https://damona.readthedocs.io>*:

```
pip install damona
damona install fitter
```

CHAPTER 3

Usage

3.1 standalone

A standalone application (very simple) is also provided and works with input CSV files:

```
fitter fitdist data.csv --column-number 1 --distributions gamma,normal
```

It creates a file called fitter.png and a log fitter.log

3.2 From Python shell

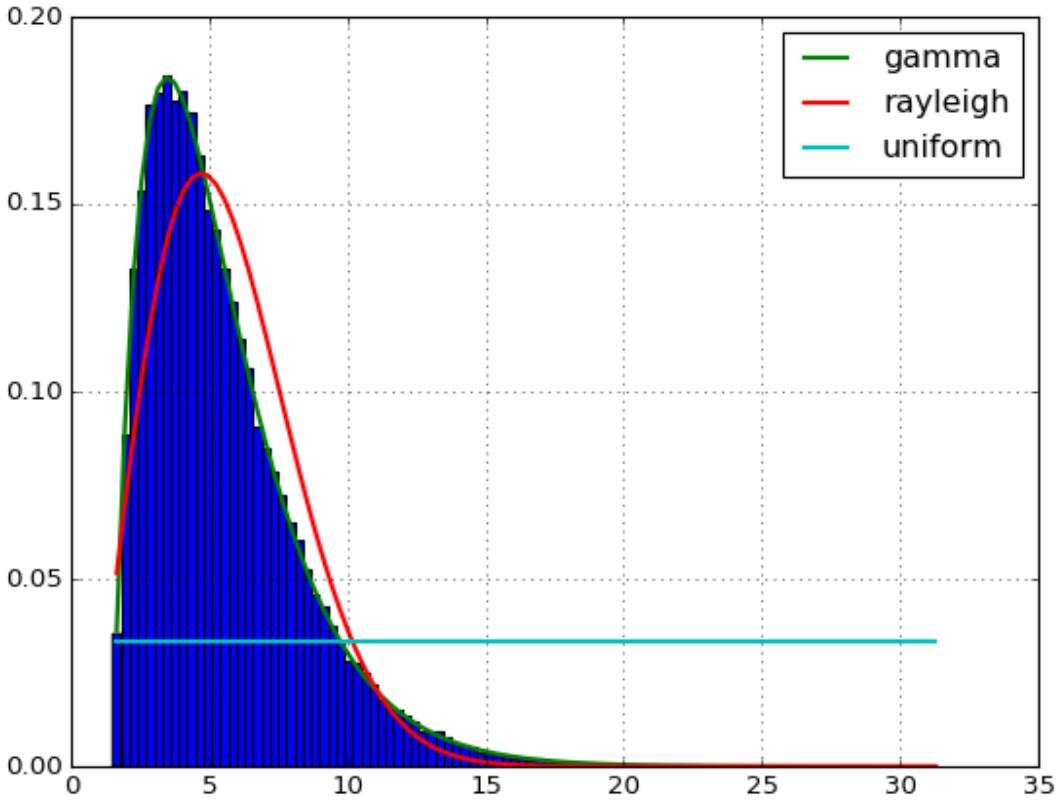
First, let us create a data samples with N = 10,000 points from a gamma distribution:

```
from scipy import stats
data = stats.gamma.rvs(2, loc=1.5, scale=2, size=10000)
```

Note: the fitting is slow so keep the size value to reasonable value.

Now, without any knowledge about the distribution or its parameter, what is the distribution that fits the data best ? Scipy has 80 distributions and the **Fitter** class will scan all of them, call the fit function for you, ignoring those that fail or run forever and finally give you a summary of the best distributions in the sense of sum of the square errors. The best is to give an example:

```
from fitter import Fitter
f = Fitter(data)
f.fit()
# may take some time since by default, all distributions are tried
# but you call manually provide a smaller set of distributions
f.summary()
```



See the online documentation for details.

3.2.1 Changelog

Version	Description
1.3.0	• parallel process implemented https://github.com/cokelaer/fitter/pull/25 thanks to @arsenyinfo
1.2.3	• remove vervose arguments in Fitter class. Using the logging module instead • the Fitter.fit has now a progress bar • add a standalone application called ... fitter (see the doc)
1.2.2	was not released
1.2.1	adding new class called histfit (see documentation)
1.2	• Fixed the version. Previous version switched from 1.0.9 to 1.1.11. To start a fresh version, we increase to 1.2.0 • Merged pull request required by bioconda • Merged pull request related to implementation of AIC/BIC/KL criteria (https://github.com/cokelaer/fitter/pull/19). This also fixes https://github.com/cokelaer/fitter/issues/9 • Implement two functions to get all distributions, or a list of common distributions to help users decreading computational time (https://github.com/cokelaer/fitter/issues/20). Also added a FAQS section. • travis tested Python 3.6 and 3.7 (not 3.5 anymore)
1.1	• Fixed deprecated warning • fitter is now in readthedocs at fitter.readthedocs.io
1.0.9	• https://github.com/cokelaer/fitter/pull/8 and 11 PR https://github.com/cokelaer/fitter/pull/8
1.0.6	• summary() now returns the dataframe (instead of printing it)
1.0.5	https://github.com/cokelaer/fitter/issues
1.0.2	add manifest to fix missing source in the pypi repository.

CHAPTER 4

Documentation

4.1 Examples

Let us start with an example. We generate a vector of values from a gamma distribution.

```
from scipy import stats
data = stats.gamma.rvs(2, loc=1.5, scale=2, size=100000)

from fitter import Fitter
f = Fitter(data, distributions=['gamma', 'rayleigh', 'uniform'])
f.fit()
f.summary()
```

Here, we restrict the analysis to only 3 distributions by providing the list of distributions to consider. If you do not provide that parameter, 80 distributions will be considered (the analysis will be longer) and computation make take a while to finish.

The *fitter.fitter.Fitter.summary()* method shows the first best distributions (in terms of fitting).

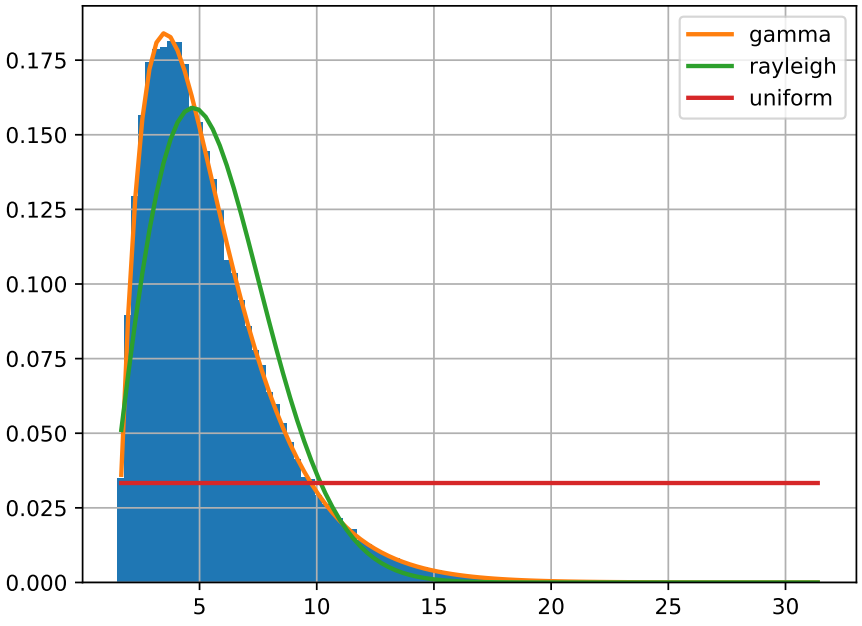
Once the fitting is performed, one may want to get the parameters corresponding to the best distribution. The parameters are stored in `fitted_param`. For instance in the example above, the summary told us that the Gamma distribution has the best fit. You would retrieve the parameters of the Gamma distribution as follows:

```
>>> f.fitted_param['gamma']
(1.9870244799532322, 1.5026555566189543, 2.0174462493492964)
```

Here, you will need to look at scipy documentation to figure out what are those parameters (mean, sigma, shape, ...). For convenience, we do provide the corresponding PDF:

```
f.fitted_pdf['gamma']
```

but you may want to plot the gamma distribution yourself. In that case, you will need to use Scipy package itself. Here is an example




```
from pylab import linspace, plot
import scipy.stats

dist = scipy.stats.gamma
param = (1.9870, 1.5026, 2.0174)
X = linspace(0,10, 10)
pdf_fitted = dist.pdf(X, *param)
plot(X, pdf_fitted, 'o-')
```

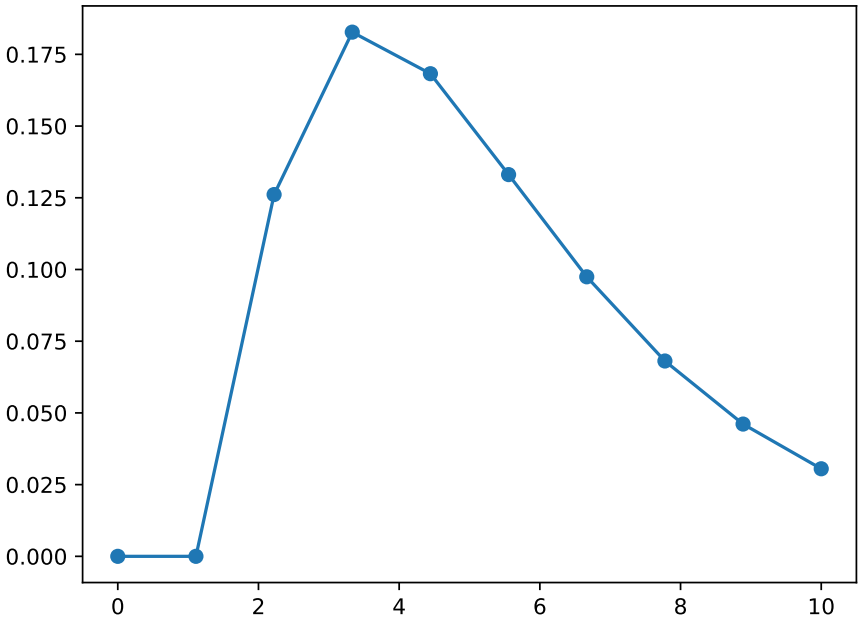
4.1.1 HistFit class: fit the density function itself

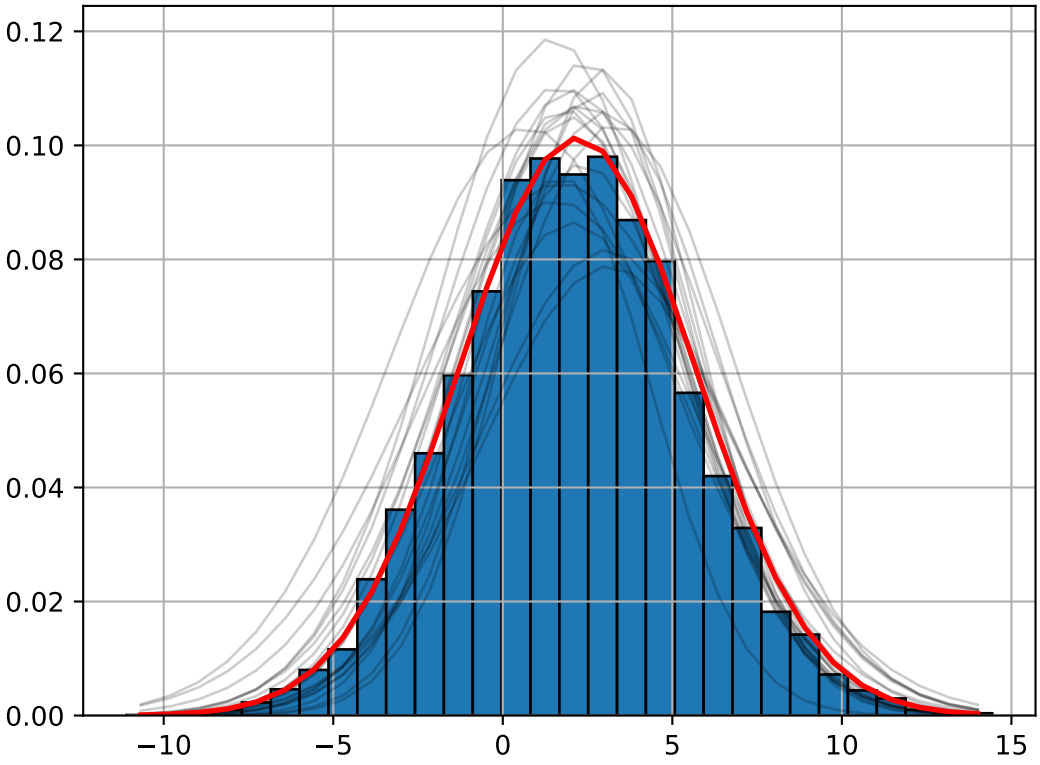
Sometimes, you only have the distribution itself. For instance:

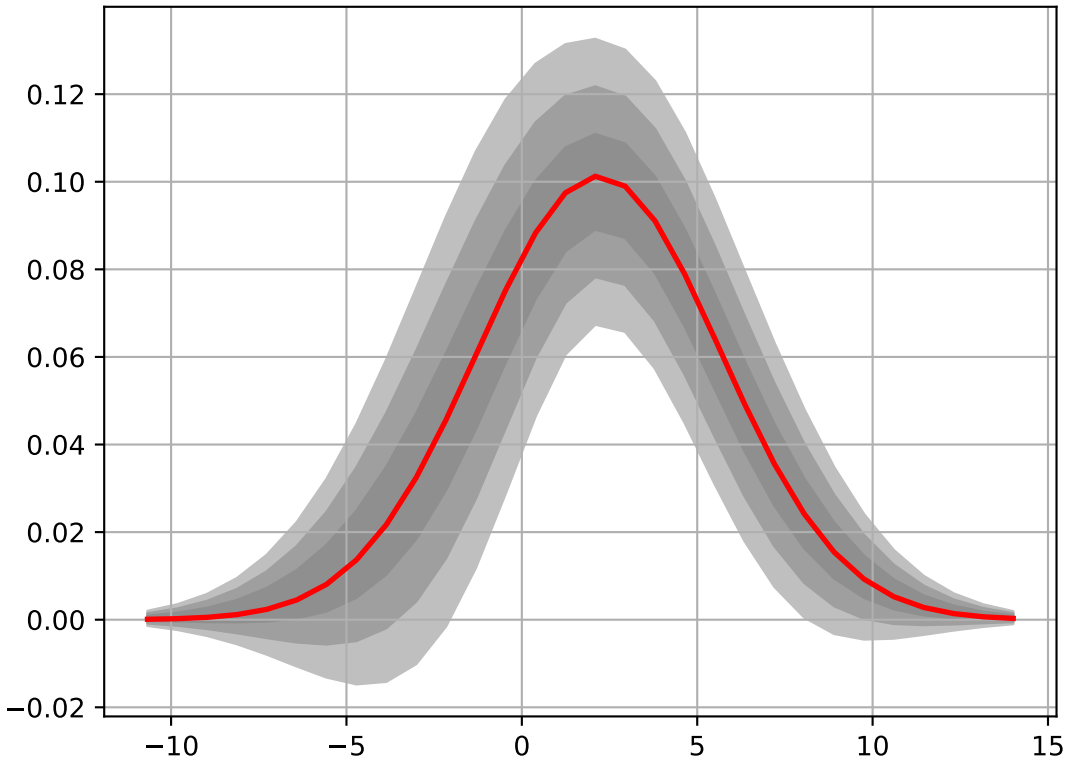
```
import scipy.stats
data = [scipy.stats.norm.rvs(2,3.4) for x in  range(10000)]
Y, X, _ = hist(data, bins=30)
```

here we have only access to Y (and X).

The histfit module provides the HistFit class to generate plots of your data with a fitting curve based on several attempt at fitting your X/Y data with some errors on the data set. For instance here below, we introduce 3% of errors and fit the data 20 times to see if the fit makes sense.


0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
2
4
6
8
10


0.12
0.10
0.08
0.06
0.04
0.02
0.00
−10
−5
0
5
10
15


0.12
0.10
0.08
0.06
0.04
0.02
0.00
−0.02
−10
−5
0
5
10
15

4.2 FAQs

4.2.1 Why does it take so long ?

A typical fitter usage is as follows:

```
from fitter import Fitter
f = Fitter(data)
f.fit()
f.summary()
```

This will run the fitting process on your data with about 80 different distributions. If you want to reduce the time significantly, provide a subset of distributions as follows:

```
from fitter import Fitter
f = Fitter(data, distributions=["gamma", "rayleigh", "uniform"])
f.fit()
f.summary()
```

Another easy way to reduce the computational time is to provide a subset of your data. If you date set has a length of 1 million data points, just sub-sample it to 10,000 points for instance. This way you can identify sensible distributions, and try again with those distributions on the entire data (divide and conquer, as always !)

4.2.2 What are the distributions available ?

Since version 1.2, you can use:

```
from fitter import get_distributions
get_distributions()
```

You may get a sub set of common distributions as follows:

```
from fitter import get_common_distributions
get_common_distributions()
```

4.3 fitter module reference

main module of the fitter package

class fitter.fitter.**Fitter**(*data*, *xmin=None*, *xmax=None*, *bins=100*, *distributions=None*, *timeout=30*, *density=True*)

Fit a data sample to known distributions

A naive approach often performed to figure out the undelying distribution that could have generated a data set, is to compare the histogram of the data with a PDF (probability distribution function) of a known distribution (e.g., normal).

Yet, the parameters of the distribution are not known and there are lots of distributions. Therefore, an automatic way to fit many distributions to the data would be useful, which is what is implemented here.

Given a data sample, we use the *fit* method of SciPy to extract the parameters of that distribution that best fit the data. We repeat this for all available distributions. Finally, we provide a summary so that one can see the quality of the fit for those distributions

Here is an example where we generate a sample from a gamma distribution.

```
>>> # First, we create a data sample following a Gamma distribution
>>> from scipy import stats
>>> data = stats.gamma.rvs(2, loc=1.5, scale=2, size=20000)

>>> # We then create the Fitter object
>>> import fitter
>>> f = fitter.Fitter(data)

>>> # just a trick to use only 10 distributions instead of 80 to speed up the␣
↪fitting
>>> f.distributions = f.distributions[0:10] + ['gamma']

>>> # fit and plot
>>> f.fit()
>>> f.summary()
        sumsquare_error
gamma          0.000095
beta           0.000179
chi            0.012247
cauchy         0.044443
anglit         0.051672
[5 rows x 1 columns]
```

Once the data has been fitted, the *summary()* metod returns a sorted dataframe where the

Looping over the 80 distributions in SciPy could takes some times so you can overwrite the *distributions* with a subset if you want. In order to reload all distributions, call `load_all_distributions()`.

Some distributions do not converge when fitting. There is a timeout of 10 seconds after which the fitting procedure is cancelled. You can change this `timeout` attribute if needed.

If the histogram of the data has outlier of very long tails, you may want to increase the `bins` binning or to ignore data below or above a certain range. This can be achieved by setting the *xmin* and *xmax* attributes. If you set xmin, you can come back to the original data by setting xmin to None (same for xmax) or just recreate an instance.

distributions = None
: list of distributions to test

fit(*amp=1*, *progress=False*, *n_jobs=-1*)
: Loop over distributions and find best parameter to fit the data for each

  When a distribution is fitted onto the data, we populate a set of dataframes:

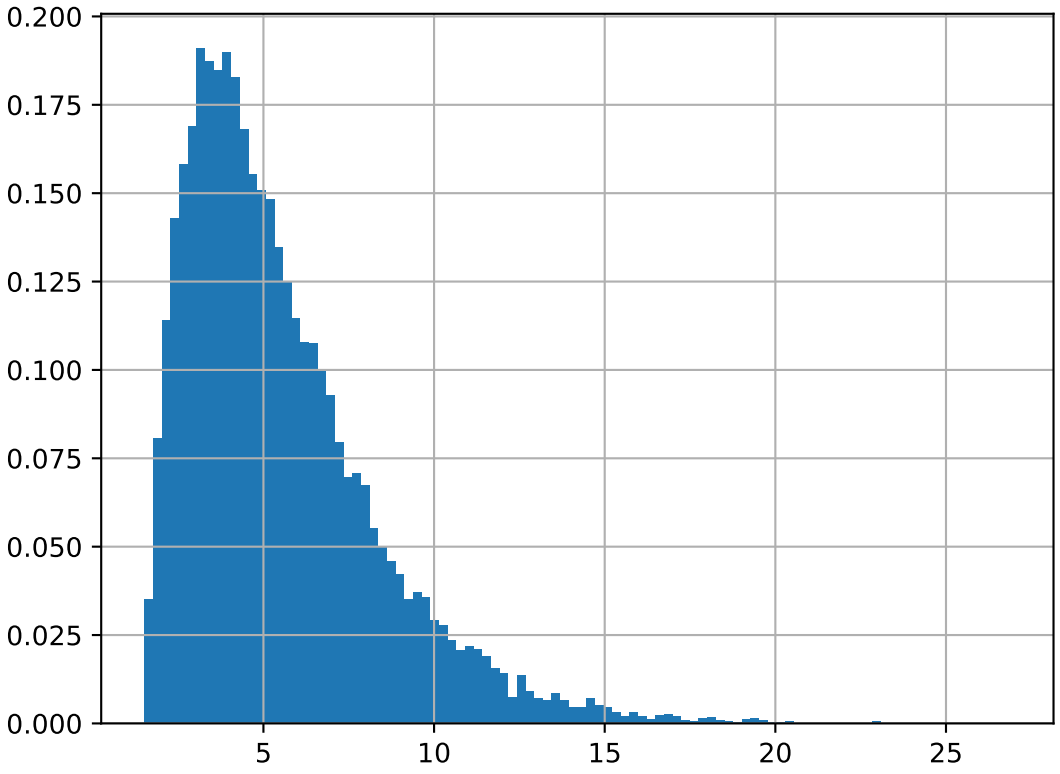
  - `df_errors` :sum of the square errors between the data and the fitted distribution i.e., $\sum_i (Y_i - pdf(X_i))^2$
  - `fitted_param` : the parameters that best fit the data
  - `fitted_pdf` : the PDF generated with the parameters that best fit the data

  Indices of the dataframes contains the name of the distribution.

get_best(*method='sumsquare_error'*)
: Return best fitted distribution and its parameters

  a dictionary with one key (the distribution name) and its parameters

hist()
: Draw normed histogram of the data using `bins`


0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
5
10
15
20
25

plot_pdf(*names=None*, *Nbest=5*, *lw=2*, *method='sumsquare_error'*)
Plots Probability density functions of the distributions

Parameters **names** (*str,list*) – names can be a single distribution name, or a list of distribution names, or kept as None, in which case, the first Nbest distribution will be taken (default to best 5)

summary(*Nbest=5*, *lw=2*, *plot=True*, *method='sumsquare_error'*)
Plots the distribution of the data and Nbest distribution

xmax
consider only data below xmax. reset if None

xmin
consider only data above xmin. reset if None

4.4 histfit module reference

class fitter.histfit.**HistFit**(*data=None*, *X=None*, *Y=None*, *bins=None*)
Plot the histogram of the data (barplot) and the fitted histogram.

The input data can be a series. In this case, we compute the histogram. Then, we fit a curve on top on the histogram that best fit the histogram.

If you already have the histogram, you can provide the arguments. In this case, X should be evenly spaced

If you have some data, histogram is computed, then we add some noise during the fitting process and repeat the process Nfit=20 times. This gives us a better estimate of the underlying mu and sigma parameters of the distribution.

You may already have your probability density function with the X and Y series. If so, just provide them; Note that the output of the hist function returns an X with N+1 values while Y has only N values. We take care of that.

Warning: This is a draft class. It currently handles only gaussian distribution. The API is probably going to change in the close future.

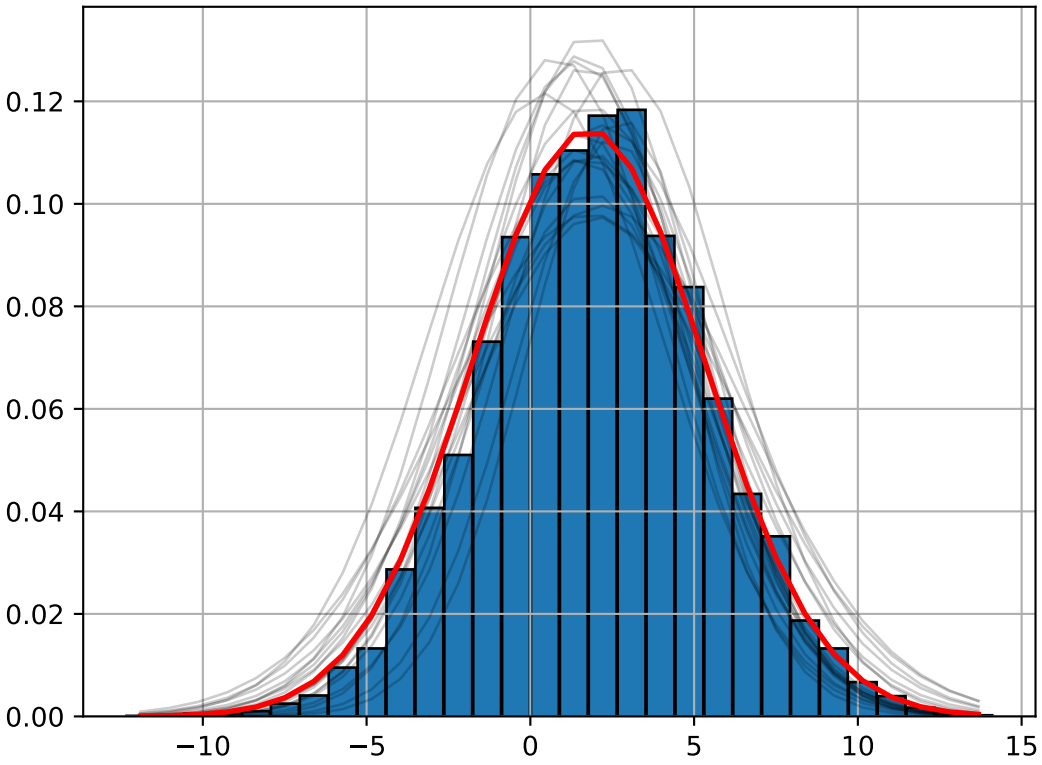
4.5 Contributions

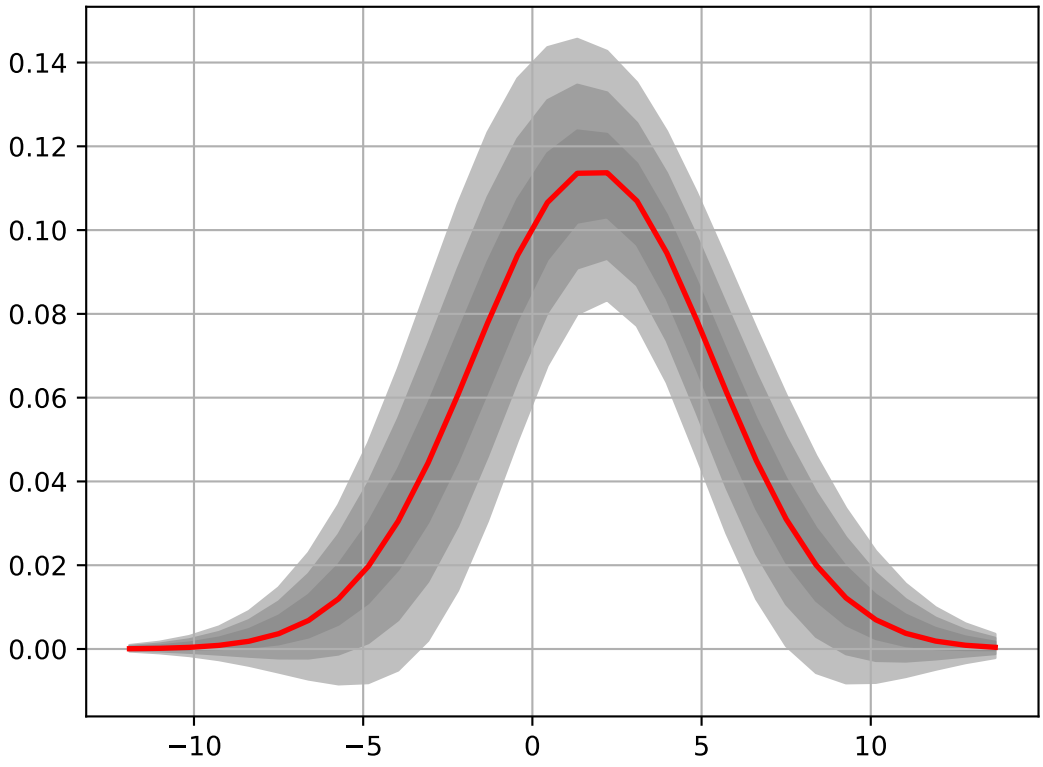
4.5.1 PRs

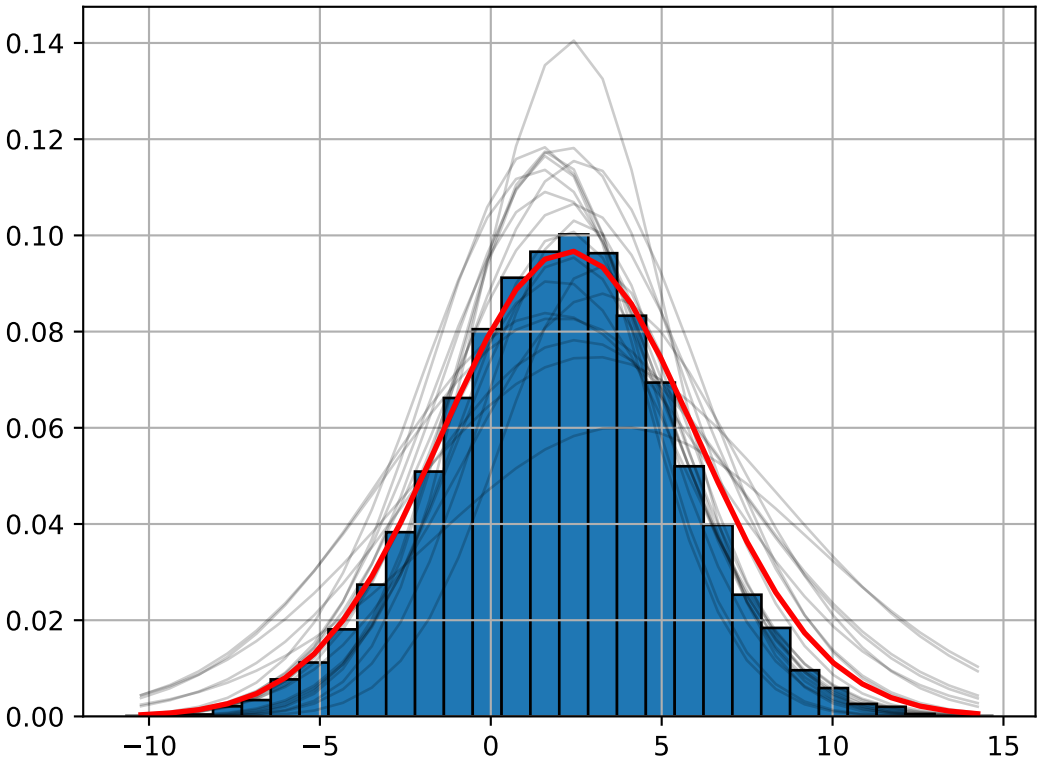
- https://github.com/cokelaer/fitter/pull/19 from https://github.com/caiostringari to add AIC/BIC/KL support
- https://github.com/cokelaer/fitter/pull/8 from https://github.com/ebroda to sort pandas dataframe properly.

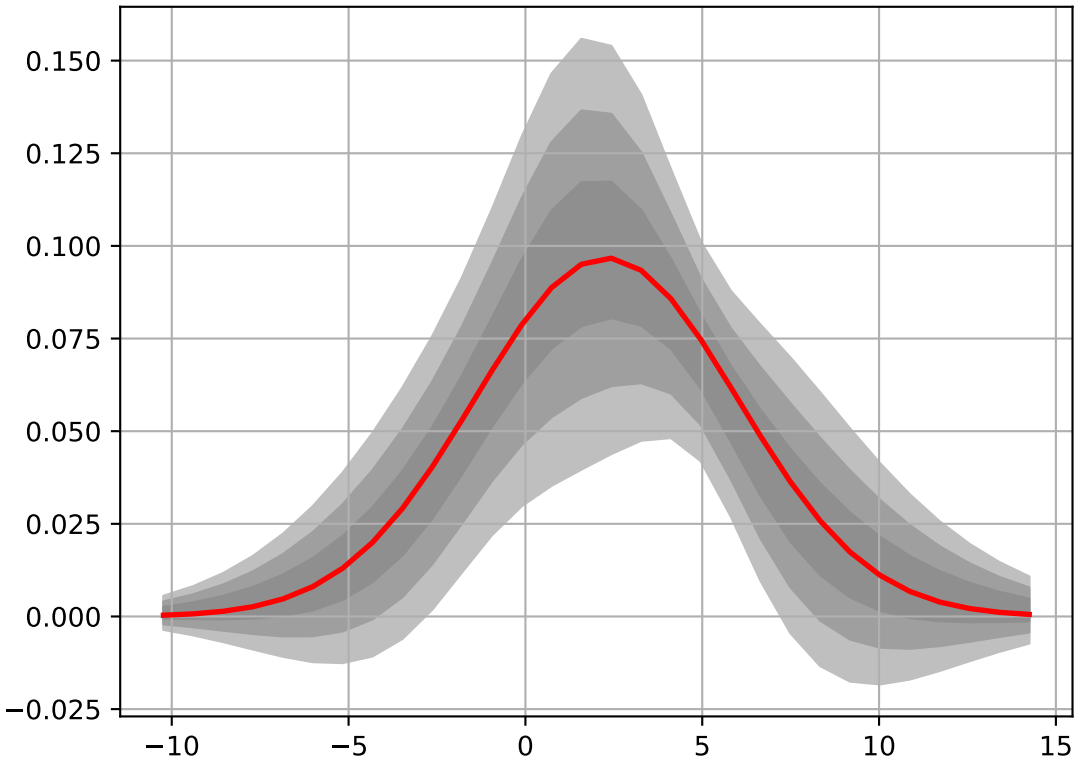
Bug report, issues, contributions ? Please visit gitbub.

Note: If I forgot to add your contribution, please do not hesitate to add a PR.


0.12
0.10
0.08
0.06
0.04
0.02
0.00
−10
−5
0
5
10
15


0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−10
−5
0
5
10


0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−10
−5
0
5
10
15


0.150
0.125
0.100
0.075
0.050
0.025
0.000
−0.025
−10
−5
0
5
10
15

Python Module Index

f

Index