probfit Documentation

Release 1.2.0

Piti Ongmongkolkul

Mar 11, 2021

CONTENTS

probfit is a set of functions that helps you construct a complex fit. It is intended to be used with iminuit. The tool includes Binned/Unbinned Likelihood estimator, χ^2 regression, binned χ^2 estimator and Simultaneous fit estimator. Normalization and convolution with cache are also included. Various builtin functions used in B physics are also provided.

In a nutshell:

```
import numpy as np
from iminuit import Minuit
from probfit import UnbinnedLH, gaussian
data = np.random.randn(10000)
unbinned_likelihood = UnbinnedLH(gaussian, data)
minuit = Minuit(unbinned_likelihood, mean=0.1, sigma=1.1)
minuit.migrad()
unbinned_likelihood.draw(minuit)
```

Probfit was created by Piti Ongmongkolkul (piti118@gmail.com). It is currently maintained by the Scikit-HEP community.

CHAPTER

ONE

FULL API DOCUMENTATION

1.1 Cost Function

Various estimators.

1.1.1 Unbinned Likelihood

class probfit.costfunc.**UnbinnedLH**(*f*, *data*, *weights=None*, *extended=False*, *extended_bound=None*, *extended_nint=100*, *badvalue=-100000*)

Construct -log(unbinned likelihood) from callable f and data points *data*. Currently can only do 1D fit.

$$\text{UnbinnedLH} = \sum_{x \in \text{data}} -\log f(x, arg, \ldots)$$

Arguments

- **f** callable object. PDF that describe the data. The parameters are parsed using iminuit's `describe`. The first positional arguement is assumed to be independent parameter. For example:

```
def gauss(x, mu, sigma):#good
    pass
def bad_gauss(mu, sigma, x):#bad
    pass
```

- **data** 1D array of data.
- **weights** Optional 1D array of weights. Default None(all 1).
- **badvalue** Optional number. The value that will be used to represent log(lh) (notice no minus sign). When the likelihood is <= 0. This usually indicate your PDF is faraway from minimum or your PDF parameter has gone into unphysical region and return negative probability density. This should be a large negative number so that iminuit will avoid those points. Default -100000.
- **extended** Set to True for extended fit. Default False. If Set to True the following term is added to resulting negative log likelihood

$$\text{ext term} = \int_{x \in \text{extended bound}} f(x, \text{args}, \ldots) \mathrm{d}x$$

- **extended_bound** Bound for calculating extended term. Default None(minimum and maximum of data will be used).
- **extended_nint** number pieces to sum up as integral for extended term (using simpson3/8). Default 100.

Note: There is a notable lack of **sum_w2** for unbinned likelihood. I feel like the solutions are quite sketchy. There are multiple ways to implement it but they don't really scale correctly. If you feel like there is a correct way to implement it feel free to do so and write document telling people about the caveat.

__call__()
Compute sum of -log(lh) given positional arguments. Position argument will be passed to pdf with independent vairable from *data* is given as the frist argument.

draw(*self*, *minuit=None*, *bins=100*, *ax=None*, *bound=None*, *parmloc=(0.05, 0.95)*, *nfbins=200*, *print_par=True*, *args=None*, *errors=None*, *parts=False*, *show_errbars='normal'*, *no_plot=False*)
Draw comparison between histogram of data and pdf.

Arguments

- **minuit** Optional but recommended `iminuit.Minuit` object. If minuit is not `None`, the pdf will be drawn using minimum value from minuit and parameters and error will be shown. If minuit is `None`, then pdf will be drawn using argument from the last call to `__call__`. Default `None`
- **bins** number of bins for histogram. Default 100.
- **ax** matplotlib axes. If not given it will be drawn on current axes `gca()`.
- **bound** bound for histogram. If `None` is given the bound will be automatically determined from the data. If you given PDF that's normalied to a region but some data is not within the bound the picture may look funny.
- **parmloc** location of parameter print out. This is passed directy to legend loc named parameter. Default (0.05,0.95).
- **nfbins** how many point pdf should be evaluated. Default 200.
- **print_par** print parameters and error on the plot. Default True.
- **args** Optional. If minuit is not given, parameter value is determined from args. This can be dictionary of the form *{'a':1.0, 'b':1.0}* or list of values. Default None.
- **errors** Optional dictionary of errors. If minuit is not given, parameter errors are determined from **errors**. Default None.
- **show_errbars** Show error bars. Default 'normal'
  - 'normal' : error = sqrt(sum of weight)
  - 'sumw2' : error = sqrt(sum of weight**2)
  - None : no errorbars (shown as a step histogram)
- **no_plot** Set this to True if you only want the return value

Returns

((data_edges, datay), (errorp,errorm), (total_pdf_x, total_pdf_y), parts)

draw_residual(*self*, *minuit=None*, *bins=100*, *ax=None*, *bound=None*, *parmloc=(0.05, 0.95)*, *print_par=False*, *args=None*, *errors=None*, *errbar_algo='normal'*, *norm=False*, ***kwargs*)
Draw difference between data and PDF

Arguments

- **minuit** Optional but recommended `iminuit.Minuit` object. If minuit is not `None`, the pdf will be drawn using minimum value from minuit and parameters and error will be shown. If minuit is `None`, then pdf will be drawn using argument from the last call to `__call__`. Default `None`

- **bins** number of bins for histogram. Default 100.
- **ax** matplotlib axes. If not given it will be drawn on current axes `gca()`.
- **bound** bound for histogram. If `None` is given the bound will be automatically determined from the data. If you given PDF that's normalied to a region but some data is not within the bound the picture may look funny.
- **parmloc** location of parameter print out. This is passed directy to legend loc named parameter. Default (0.05,0.95).
- **print_par** print parameters and error on the plot. Default True.
- **args** Optional. If minuit is not given, parameter value is determined from args. This can be dictionary of the form *{'a':1.0, 'b':1.0}* or list of values. Default None.
- **errors** Optional dictionary of errors. If minuit is not given, parameter errors are determined from **errors**. Default None.
- **show_errbars** Show error bars. Default True
- **errbar_algo** How the error bars are calculated 'normal' : error = sqrt(sum of weight) [Default] 'sumw2' : error = sqrt(sum of weight**2)
- **norm** Normalzed by the error bar or not. Default False.
- **kwargs** Passed to *probfit.plotting.draw_residual()*

show(*self*, **arg*, ***kwd*)

Same thing as *draw()*. But show the figure immediately.

See also:

draw() for arguments.

Example

1.1.2 Binned Likelihood

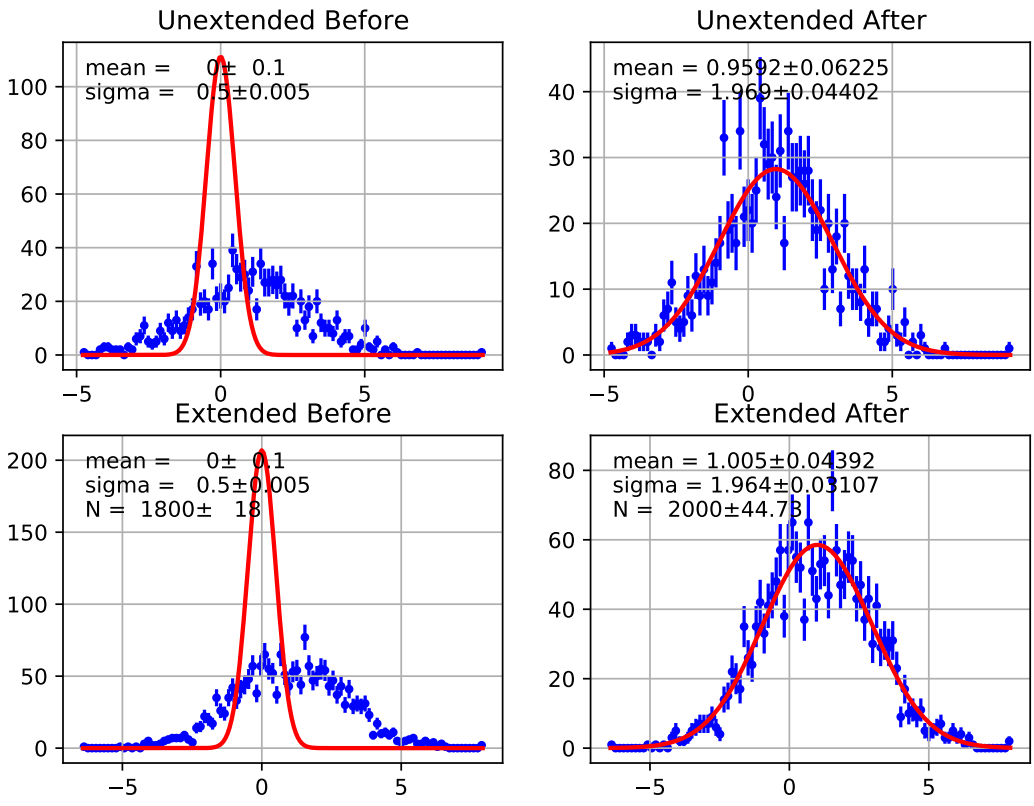
class probfit.costfunc.**BinnedLH**(*f*, *data*, *bins=40*, *weights=None*, *weighterrors=None*, *bound=None*, *badvalue=1000000*, *extended=False*, *use_w2=False*, *nint_subdiv=1*)

Create a Poisson Binned Likelihood object from given PDF **f** and **data** (raw points not histogram). Constant term and expected minimum are subtracted off (aka. log likelihood ratio). The exact calculation will depend on **extended** and **use_w2** keyword parameters.

$$\text{BinnedLH} = -\sum_{i \in bins} s_i \times \left(h_i \times \log(\frac{E_i}{h_i}) + (h_i - E_i) \right)$$

where

- h_i is sum of weight of data in ith bin.
- b_i is the width of ith bin.
- N is total number of data. $N = \sum_i h_i$.
- E_i is expected number of occupancy in ith bin from PDF calculated using average of pdf value at both sides of the bin l_i, r_i. The definition for E_i depends whether extended likelihood is requested.


Unextended Before
mean = 0± 0.1
sigma = 0.5±0.005
Unextended After
mean = 0.9592±0.06225
sigma = 1.969±0.04402
Extended Before
mean = 0± 0.1
sigma = 0.5±0.005
N = 1800± 18
Extended After
mean = 1.005±0.04392
sigma = 1.964±0.03107
N = 2000±44.73

If extended likelihood is requested (`extended=True`):

$$E_i = \frac{f(l_i, arg\ldots) + f(r_i, arg\ldots)}{2} \times b_i$$

If extended likelihood is NOT requested (`extended=False`):

$$E_i = \frac{f(l_i, arg\ldots) + f(r_i, arg\ldots)}{2} \times b_i \times N$$

Note: You are welcome to patch this with a the using real area. So that, it's less sensitive to bin size. Last time I check ROOFIT used **f** evaluated at midpoint.

- s_i is a scaled factor. It's 1 if `sum_w2=False`. It's $s_i = \frac{h_i}{\sum_{j \in \text{bin } i} w_j^2}$ if `sum_w2=True`. The factor will scale the statistics to the unweighted data.

Note: You may wonder why there is $h_i - E_i$ added at the end for each term of the sum. They sum up to zero anyway. The reason is the precision near the minimum. If we taylor expand the logarithmic term near $h_i \approx E_i$ then the first order term will be $h_i - E_i$. Subtracting this term at the end gets us the nice pure parabolic behavior for each term at the minimum.

Arguments

- **f** callable object. PDF that describe the data. The parameters are parsed using iminuit's `describe`. The first positional arguement is assumed to be independent parameter. For example:

```
def gauss(x, mu, sigma):#good
    pass
def bad_gauss(mu, sigma, x):#bad
    pass
```

- **data** 1D array of data if raw data; otherwise, this can be a tuple of contents, edges (like NumPy).
- **bins** number of bins data should be histogrammed. Default 40.
- **weights** Optional 1D array of weights. Default `None` (all 1's).
- **weighterrors** Optional 1D array of weight errors. Default `None` This is usually used for binned datasets where you want to manipulate each bin's error. It doesn't make sense if the data is unbinned data.
- **bound** tuple(min,max). Histogram bound. If `None` is given, bound is automatically determined from data. Default None.
- **badvalue** Optional number. The value that will be used to represent log(lh) (notice no minus sign). When the likelihood is <= 0. This usually indicate your PDF is faraway from minimum or your PDF parameter has gone into unphysical region and return negative probability density. This should be a large POSITIVE number so that iminuit will avoid those points. Default 100000.
- **extended** Boolean whether this likelihood should be extended likelihood or not. Default False.
- **use_w2** Scale -log likelihood so that to the original unweighted statistics. Default False.
- **nint_subdiv** controls how BinnedLH do the integral to find expect number of event in each bin. The number represent the number of subdivisions in each bin to do simpson3/8 rule. Default 1.

`__call__``()`
Calculate sum -log(poisson binned likelihood) given positional arguments

draw(*self*, *minuit=None*, *ax=None*, *parmloc=(0.05, 0.95)*, *nfbins=200*, *print_par=True*, *args=None*, *errors=None*, *parts=False*, *no_plot=False*)
Draw comparison between histogram of data and pdf.

Arguments

- **minuit** Optional but recommended `iminuit.Minuit` object. If minuit is not `None`, the pdf will be drawn using minimum value from minuit and parameters and error will be shown. If minuit is `None`, then pdf will be drawn using argument from the last call to `__call__`. Default `None`
- **ax** matplotlib axes. If not given it will be drawn on current axes `gca()`.
- **parmloc** location of parameter print out. This is passed directy to legend loc named parameter. Default (0.05,0.95).
- **nfbins** how many point pdf should be evaluated. Default 200.
- **print_par** print parameters and error on the plot. Default True.
- **no_plot** Set this to True if you only want the return value

Returns

((data_edges, data_y), (errorp,errorm), (total_pdf_x, total_pdf_y), parts)

draw_residual(*self*, *minuit=None*, *ax=None*, *parmloc=(0.05, 0.95)*, *print_par=False*, *args=None*, *errors=None*, *norm=False*, ***kwargs*)
Draw difference between data and pdf.

Arguments

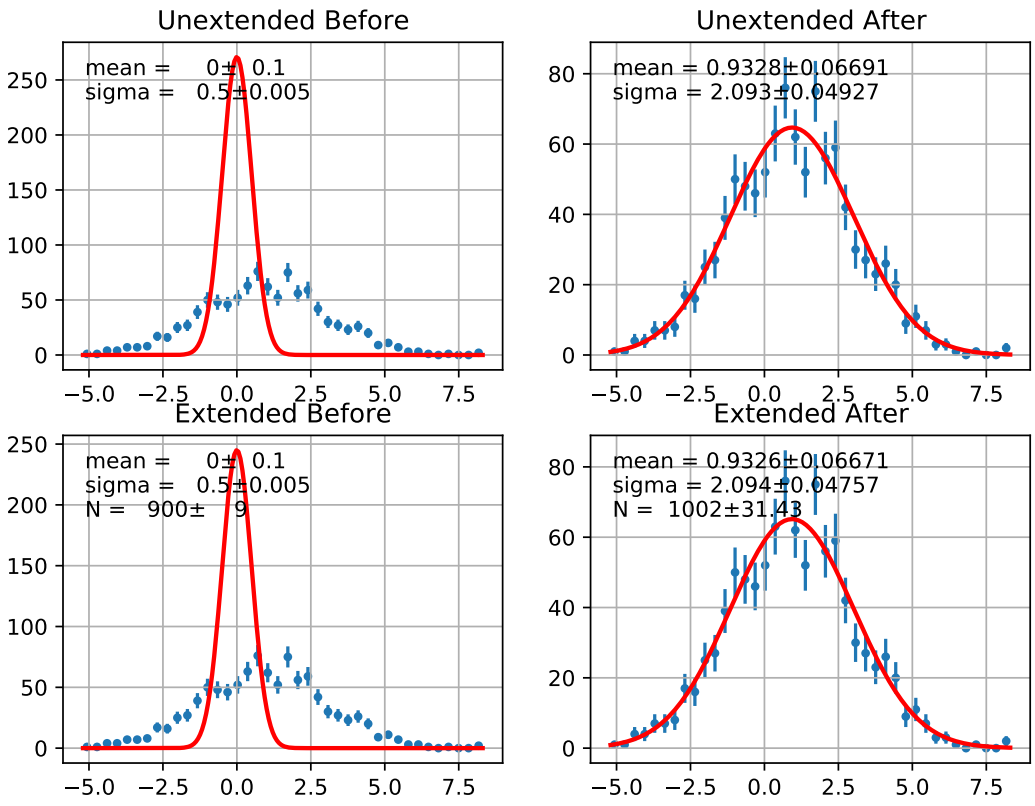
- **minuit** Optional but recommended `iminuit.Minuit` object. If minuit is not `None`, the pdf will be drawn using minimum value from minuit and parameters and error will be shown. If minuit is `None`, then pdf will be drawn using argument from the last call to `__call__`. Default `None`
- **ax** matplotlib axes. If not given it will be drawn on current axes `gca()`.
- **parmloc** location of parameter print out. This is passed directy to legend loc named parameter. Default (0.05,0.95).
- **print_par** print parameters and error on the plot. Default True.
- **norm** If True, draw difference normalized by error Default False.
- **kwargs** Passed to *probfit.plotting.draw_residual()*

show(*self*, **arg*, ***kwd*)
Same thing as *draw()*. But show the figure immediately.

See also:

draw() for arguments.

Example


Unextended Before
mean = 0± 0.1
sigma = 0.5±0.005
Unextended After
mean = 0.9328±0.06691
sigma = 2.093±0.04927
Extended Before
mean = 0± 0.1
sigma = 0.5±0.005
N = 900± 9
Extended After
mean = 0.9326±0.06671
sigma = 2.094±0.04757
N = 1002±31.43

1.1.3 χ^2 Regression

class probfit.costfunc.**Chi2Regression**(*f*, *x*, *y*, *error=None*, *weights=None*)

Create χ^2 regression object. This is for fitting funciton to data points(x,y) rather than fitting PDF to a distribution.

$$\text{Chi2Regression} = \sum_i w_i \times \left(\frac{f(x_i, arg\ldots) - y_i}{error_i} \right)^2$$

Arguments

- **f** callable object to describe line given by (**x** , **y**). The first positional arugment of **f** is assumed to be independent variable. Ex::

```
def gauss(x, mu, sigma):#good
    pass
def bad_gauss(mu, sigma, x):#bad
    pass
```

- **x** 1D array of independent variable
- **y** 1D array of expected **y**
- **error** optional 1D error array. If `None` is given, it's assumed to be all 1's.
- **weight** 1D array weight for each data point.

__call__()

Compute χ^2

draw(*self*, *minuit=None*, *ax=None*, *parmloc=(0.05, 0.95)*, *print_par=True*, *args=None*, *errors=None*, *parts=False*, *nbins=None*, *no_plot=False*)

Draw comparison between points (**x**,**y**) and the function **f**.

Arguments

- **minuit** Optional but recommended `iminuit.Minuit` object. If minuit is not `None`, the pdf will be drawn using minimum value from minuit and parameters and error will be shown. If minuit is `None`, then pdf will be drawn using argument from the last call to `__call__`. Default `None`
- **ax** matplotlib axes. If not given it will be drawn on current axes `gca()`.
- **parmloc** location of parameter print out. This is passed directy to legend loc named parameter. Default (0.05,0.95).
- **print_par** print parameters and error on the plot. Default True.
- **parts** draw components of PDF. Default False.
- **no_plot** Set this to true if you only want the return value
- **nbins** draw the PDF curve using this number of bins, instead of (if None) using the data binning.

Returns

((data_x, data_y), (errorp,errorm), (total_pdf_x, total_pdf_y), parts)

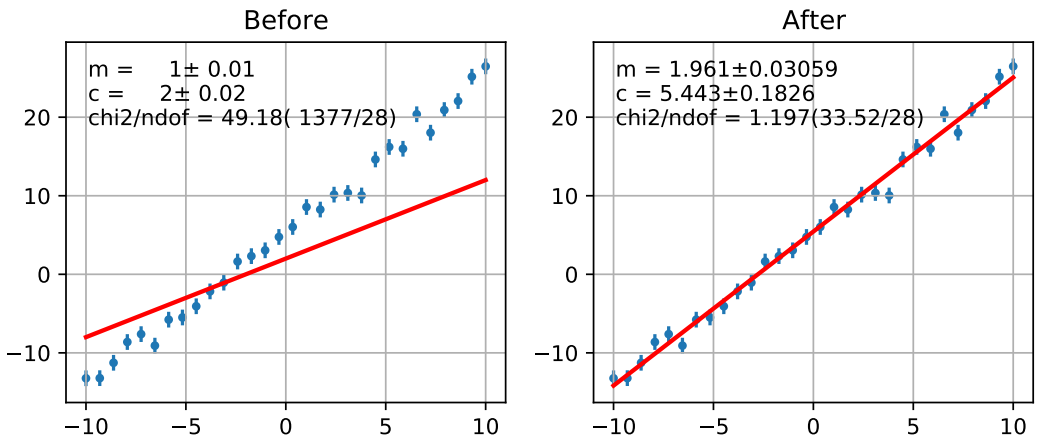
draw_residual(*self*, *minuit=None*, *ax=None*, *args=None*, *errors=None*, *grid=True*, *norm=False*)

show(*self*, **arg*, ***kwd*)

Same thing as *draw()*. But show the figure immediately.

See also:

draw() for arguments.



Example

1.1.4 Binned χ^2

class probfit.costfunc.**BinnedChi2**(*f*, *data*, *bins=40*, *weights=None*, *bound=None*, *sumw2=False*, *nint_subdiv=1*, *data_binned=False*, *bin_contents=None*, *bin_edges=None*)

Create Binned Chi2 Object. It calculates chi^2 assuming poisson statistics.

$$\text{BinnedChi2} = \sum_{i \in \text{bins}} \left(\frac{h_i - f(x_i, arg\ldots) \times b_i \times N}{err_i} \right)^2$$

Where err_i is

- $\sqrt{\sum_{j \in \text{bin}_i} w_j}$ if `sum_w2=False`.
- $\sqrt{\sum_{j \in \text{bin}_i} w_j^2}$ if `sum_w2=True`.

Arguments

- **f** callable object. PDF describing **data**. The first positional arugment of **f** is assumed to be independent variable. Ex::

```
def gauss(x, mu, sigma):#good
    pass
def bad_gauss(mu, sigma, x):#bad
    pass
```

- **data** 1D array of data if raw data; otherwise, this can be a tuple of contents, edges (like NumPy).
- **bins** Optional number of bins to histogram data. Default 40.
- **weights** 1D array weights.
- **bound** tuple(min,max) bound of histogram. If `None` is given it's automatically determined from the data.
- **sumw2** scale the error using $\sqrt{\sum_{j \in \text{bin}_i} w_j^2}$.

- **nint_subdiv** controls how BinnedChi2 do the integral to find expect number of event in each bin. The number represent the number of subdivisions in each bin to do simpson3/8. Default 1.

__call__()
Calculate χ^2 given positional arguments

draw(*self*, *minuit=None*, *ax=None*, *parmloc=(0.05, 0.95)*, *nfbins=200*, *print_par=True*, *args=None*, *errors=None*, *parts=False*, *no_plot=False*)
Draw comparison histogram of data and the function **f**.

Arguments

- **minuit** Optional but recommended `iminuit.Minuit` object. If minuit is not `None`, the pdf will be drawn using minimum value from minuit and parameters and error will be shown. If minuit is `None`, then pdf will be drawn using argument from the last call to `__call__`. Default `None`
- **ax** matplotlib axes. If not given it will be drawn on current axes `gca()`.
- **parmloc** location of parameter print out. This is passed directy to legend loc named parameter. Default (0.05,0.95).
- **nfbins** number of points to calculate f
- **print_par** print parameters and error on the plot. Default True.
- **no_plot** Set this to true if you only want the return value

Returns

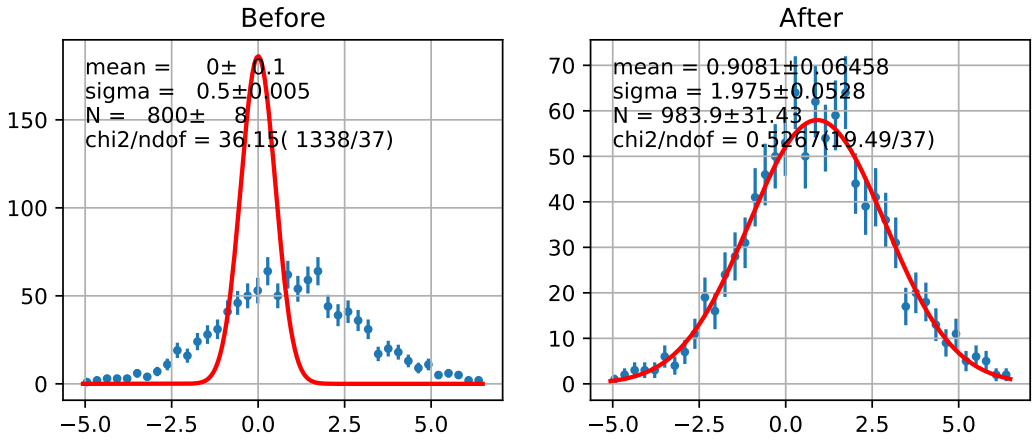
((data_edges, data_y), (errorp,errorm), (total_pdf_x, total_pdf_y), parts)

show(*self*, **arg*, ***kwd*)
Same thing as *draw()*. But show the figure immediately.

See also:

draw() for arguments.

Example



1.1.5 Simultaneous Fit

class probfit.costfunc.**SimultaneousFit**(*factors=None*, **arg*, *prefix=None*, *skip_prefix=None*)

Construct simultaneous fit from given cost functions.

Arguments

- **factors** Optional factor array. If not None, each cost function is scaled by *factors[i]* before being summed up. Default None.
- **prefix** Optional list of prefix. Add prefix to variable name of each cost function so that you don't accidentally merge them. Default None.
- **skip_prefix** list of variable names that prefix should not be applied to. Useful if you want to prefix them and shared some of them.

__call__(**args*, ***kwargs*)
Call self as a function.

args_and_error_for(*self*, *findex*, *minuit=None*, *args=None*, *errors=None*)
convert argument from minuit/dictionary/errors to argument and error for cost function with index **findex**

draw(*self*, *minuit=None*, *args=None*, *errors=None*, ***kwds*)
Draw each pdf along with data on plotting grid

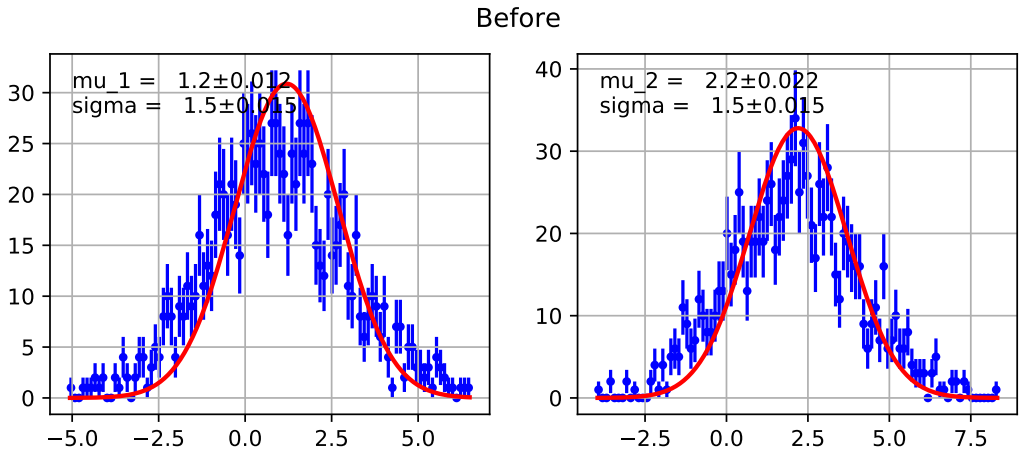
return list of return values from each cost function

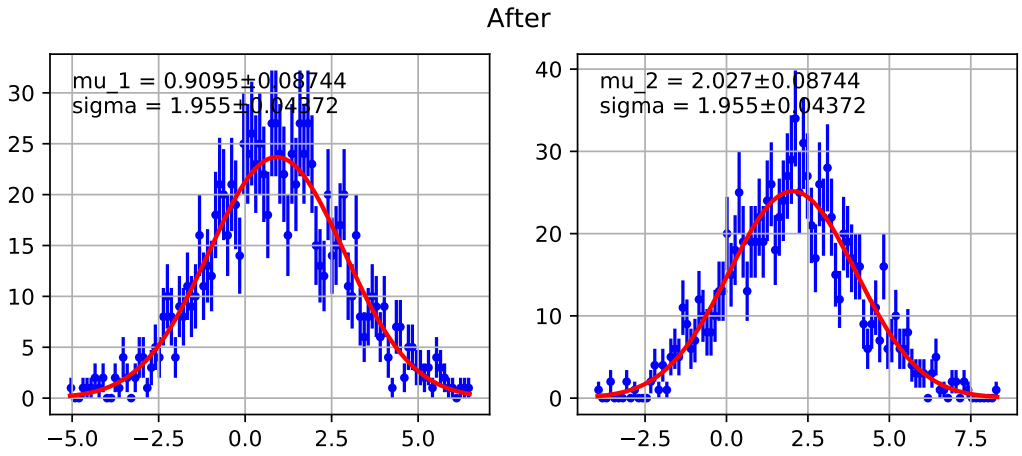
show(*self*, *m=None*)
Same thing as `draw()`. But show the figure immediately.

See also:

`draw()` for arguments.

Example





1.2 Functor

Manipulate and combined your pdf in various ways.

1.2.1 Extended

class probfit.functor.**Extended**(*f*, *extname='N'*)

Transformed given **f** into extended from.

```
def f(x,mu,sigma):
    return gaussian(x,mu,sigma)
g = Extended(f) #g is equivalent to N*f(x,mu,sigma)
describe(g) #('x','mu','sigma','N')

#g is equivalent to
def g_equiv(x, mu, sigma, N):
    return N*f(x, mu, sigma)
```

Arguments

- **f** call object. PDF.
- **extname** optional string. Name of the extended parameter. Default *'N'*

1.2.2 Normalized

class probfit.functor.**Normalized**(*f*, *bound*, *nint=300*, *warnfloat=1*)

Transformed PDF in to a normalized version. The normalization factor is cached according the shape parameters(all arguments except the first one).

```
def f(x, a, b, c):
    return do_something(x, a, b, c)
g = Normalized(f, (0., 1.))
#g is eqivalent to (shown here without cache)
def g_equiv(x, a, b, c):
    return f(x, a, b, c)/Integrate(f(x, a, b, c), range=(0., 1.))
```

Arguments

- **f** function to normalized.
- **bound** bound of the normalization.
- **nint** optional number of pieces to integrate. Default 300.
- **warnfloat** optinal number of times it should warn if integral of the given function is really small. This usually indicate you bound doesn't make sense with given parameters.

Note: Integration implemented here is just a simple trapezoid rule. You are welcome to implement something better and submit a pull request.

Warning: Never reused Normalized object with different parameters in the same pdf.

```
#DO NOT DO THIS
def f(x,y,z):
    do_something(x,y,z)
pdf1 = Normalized(f,(0,1)) #h has it's own cache

pdf2 = rename(pdf1,['x','a','b'])#don't do this

totalpdf = Add2PdfNorm(pdf1,pdf2)
```

The reason is that Normalized has excatly one cache value. Everytime it's called with different parameters the cache is invalidate and it will recompute the integration which takes a long time. For the example given above, when calling totalpdf, calling to *pdf2* will always invalidate *pdf1* cache causing it to recompute integration for every datapoint *x*. The fix is easy:

```
#DO THIS INSTEAD
def f(x,y,z):
    do_something(x,y,z)
pdf1 = Normalized(f,(0,1)) #h has it's own cache

pdf2_temp = Normalized(f,(0,1)) #own separate cache
pdf2 = rename(pdf2_temp,['x','a','b'])

totalpdf = Add2PdfNorm(pdf1,pdf2)
```

1.2.3 Convolve

class probfit.functor.**Convolve**(*f*, *g*, *gbound*, *nbins=1000*)

Make convolution from supplied **f** and **g**. If your functions are analytically convolutable you will be better off implementing analytically. This functor is implemented using numerical integration with bound there are numerical issue that will, in most cases, lightly affect normalization.s

Argument from **f** and **g** is automatically merge by name. For example,

```
f = lambda x, a, b, c: a*x**2+b*x+c
g = lambda x, a, sigma: gaussian(x,a,sigma)
h = Convolve(f,g,gbound)
describe(h)#['x','a','b','c','sigma']

#h is equivalent to
def h_equiv(x, a, b, c, sigma):
    return Integrate(f(x-t, a, b, c)*g(x, a, b, c), t_range=gbound)
```

$$\text{Convolve(f,g)}(t, arg\ldots) = \int_{\tau \in \text{gbound}} f(t-\tau, arg\ldots) \times g(t, arg\ldots)\,\mathrm{d}\tau$$

Argument

- **f** Callable object, PDF.
- **g** Resolution function.
- **gbound** Bound of the resolution. Supplied something such that g(x,*arg) is near 0 at the edges. Current implementation is multiply reverse slide add straight up from the definition so this bound is important. Overbounding is recommended.
- **nbins** Number of bins in multiply reverse slide add. Default(1000)

Note: You may be worried about normalization. By the property of convolution:

$$\int_{\mathbf{R}^d} (f * g)(x)\,dx = \left(\int_{\mathbf{R}^d} f(x)\,dx\right)\left(\int_{\mathbf{R}^d} g(x)\,dx\right).$$

This means if you convolute two normalized distributions you get back a normalized distribution. However, since we are doing numerical integration here we will be off by a little bit.

1.2.4 BlindFunc

class probfit.functor.**BlindFunc**(*f*, *toblind*, *seedstring*, *width=1*, *signflip=True*)

Transform given parameter(s) in the given **f** by a random shift so that the analyst won't see the true fitted value.

$$BlindFunc(f, ['y', 'z'], '123')(x, y, z) = f(x, y \pm \delta, z)$$

```
def f(x,mu,sigma):
    return gaussian(x,mu,sigma)
g= BlindFunc(f, toblind=['mu','sigma'], seedstring= 'abcxyz', width=1,␣
↪signflip=True)
describe(g) # ['x', 'mu', 'sigma']
```

Arguments

- **f** call object. A function or PDF.
- **toblind** a list of names of parameters to be blinded. Can be a scalar if only one.
- **seedstring** a string random number seed to control the random shift
- **width** a Gaussian width that controls the random shift
- **signflip** if True, sign of the parameter may be flipped before being shifted.

1.2.5 AddPdf

class probfit.functor.**AddPdf**(*prefix=None*, **arg*, *factors=None*, *skip_prefix=None*)

Directly add PDF without normalization nor factor. Parameters are merged by names.

```
def f(x, a, b, c):
    return do_something(x,a,b,c)
def g(x, d, a, e):
    return do_something_else(x, d, a, e)

h = AddPdf(f, g)# you can do AddPdf(f, g, h)
#h is equivalent to
def h_equiv(x, a, b, c, d, e):
    return f(x, a, b, c) + g(x, d, a, e)
```

Arguments

- **prefix** array of prefix string with length equal to number of callable object passed. This allows you to add two PDF without having parameters from the two merge.

```
h2 = AddPdf(f, g, prefix=['f','g'])
#h is equivalent to
def h2_equiv(x, f_a, f_b, f_c, g_d, g_a, g_e):
    return f(x, f_a, f_b, f_c) + g(x, g_d, g_a, g_e)
```

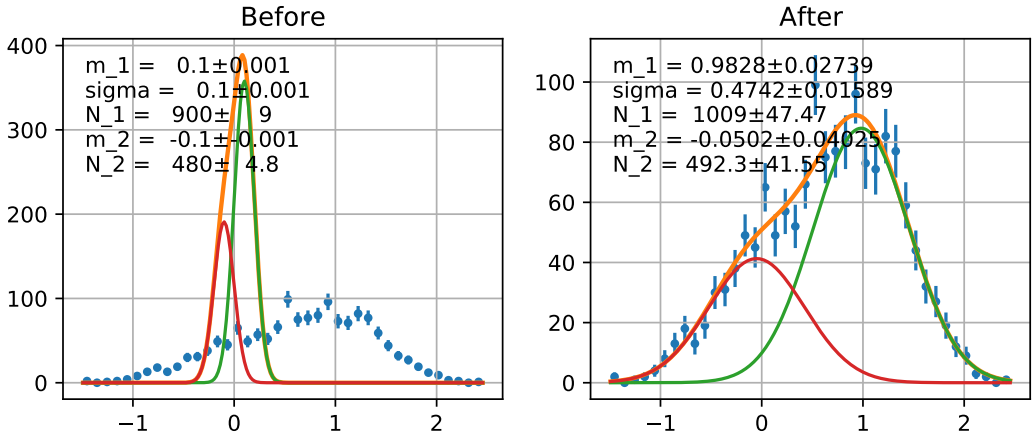
- **factors** list of callable factor function if given Add pdf will simulate the pdf of the form:

```
factor[0]*f + factor[1]*g
```

  Note that all argument for callable factors will be prefixed (if given) as opposed to skipping the first one for pdf list. If None is given, all factors are assume to be constant 1. Default None.

- **skip_prefix** list of variable that should not be prefixed. Default None. This is useful when you want to mix prefixing and sharing some of variable.

Example



1.2.6 AddPdfNorm

class probfit.functor.**AddPdfNorm**(*facname=None*, **arg*, *prefix=None*, *skip_prefix=None*)

Add PDF with normalization factor. Parameters are merged by name.

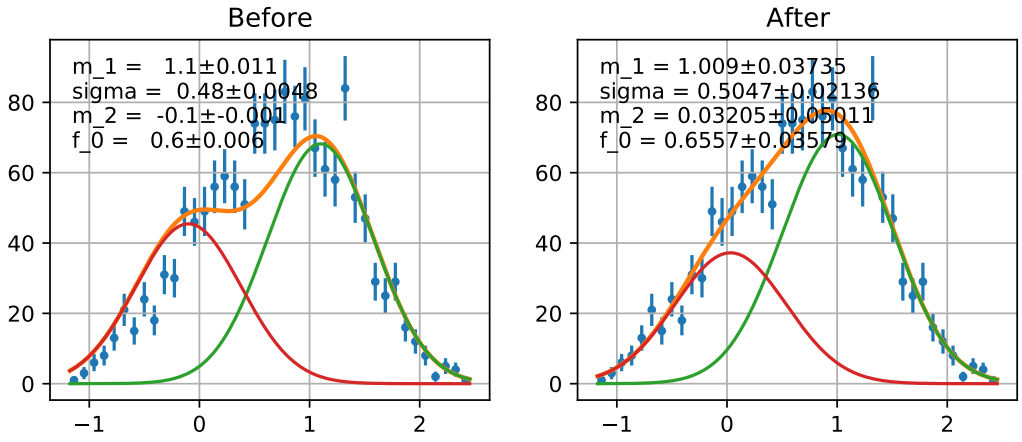
```
def f(x, a, b, c):
    return do_something(x, a, b, c)
def g(x, d, a, e):
    return do_something_else(x, d, a, e)
def p(x, b, a, c):
    return do_something_other_thing(x,b,a,c)
h = Add2PdfNorm(f, g, p)

#h is equivalent to
def h_equiv(x, a, b, c, d, e, f_0, f_1):
    return f_0*f(x, a, b, c)+ \
           f_1*g(x, d, a, e)+
           (1-f_0-f_1)*p(x, b, a, c)
```

Arguments

- **facname** optional list of factor name of length=. If None is given factor name is automatically chosen to be *f_0*, *f_1* etc. Default None.
- **prefix** optional prefix list to prefix arguments of each function. Default None.
- **skip_prefix** optional list of variable that prefix will not be applied to. Default None(empty).

Example



1.2.7 rename

probfit.funcutil.**rename**(*f*, *newarg*)
Rename function parameters.

```
def f(x,y,z):
    return do_something(x,y,z)
g = rename(f, ['x','a','b'])
print describe(g) # ['x','a','b']

#g is equivalent to
def g_equiv(x,a,b):
    return f(x,a,b)
```

Arguments

- **f** callable object
- **newarg** list of new argument names

Returns

function with new argument.

1.2.8 Decorator

class probfit.decorator.**normalized**(*bound*, *nint=1000*)
Normalized decorator

Arguments

- **bound** normalized bound
- **nint** option number of integral pieces. Default 1000.

See also:

probfit.functor.Normalized

class probfit.decorator.**extended**(*extname='N'*)
Extended decorator

Arguments

- **extname** extended parameter name. Default 'N'

See also:

probfit.functor.Extended

1.3 Builtin PDF

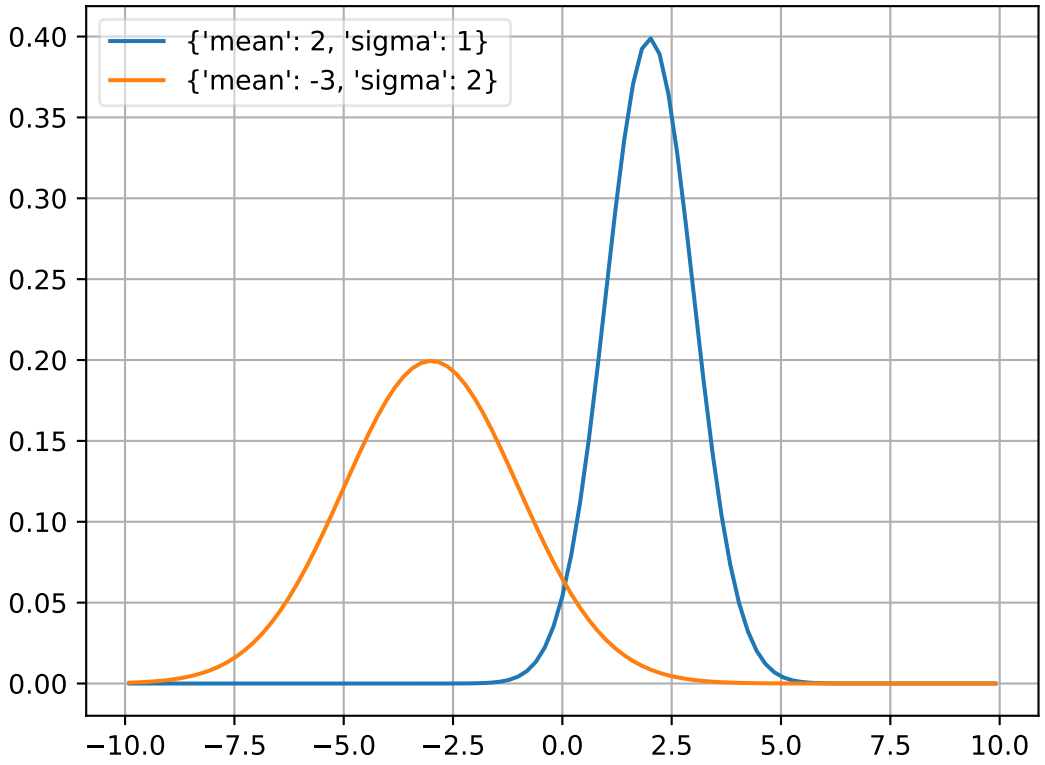
Builtin PDF written in cython.

1.3.1 gaussian

`probfit.pdf.`**`gaussian`**(*double x*, *double mean*, *double sigma*) → double

Normalized gaussian.

$$f(x; mean, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x - mean}{\sigma}\right)^2\right]$$



1.3.2 cauchy

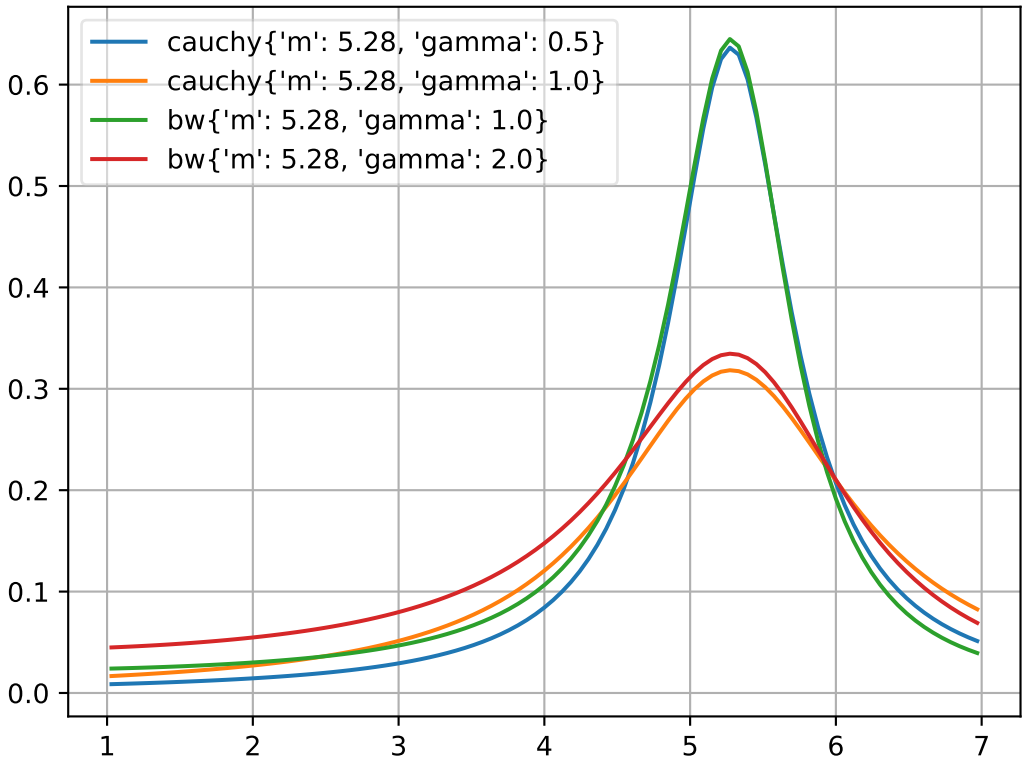
`probfit.pdf.`**`cauchy`**(*double x*, *double m*, *double gamma*) → double

Cauchy distribution aka non-relativistic Breit-Wigner

$$f(x, m, \gamma) = \frac{1}{\pi\gamma\left[1 + \left(\frac{x-m}{\gamma}\right)^2\right]}$$

See also:

`breitwigner()`


cauchy{'m': 5.28, 'gamma': 0.5}
cauchy{'m': 5.28, 'gamma': 1.0}
bw{'m': 5.28, 'gamma': 1.0}
bw{'m': 5.28, 'gamma': 2.0}
0.0
0.1
0.2
0.3
0.4
0.5
0.6
1
2
3
4
5
6
7

1.3.3 Breit-Wigner

probfit.pdf.**rtv_breitwigner**(*double x*, *double m*, *double gamma*) → double

Normalized Relativistic Breit-Wigner

$$f(x; m, \Gamma) = N \times \frac{1}{(x^2 - m^2)^2 + m^2\Gamma^2}$$

where

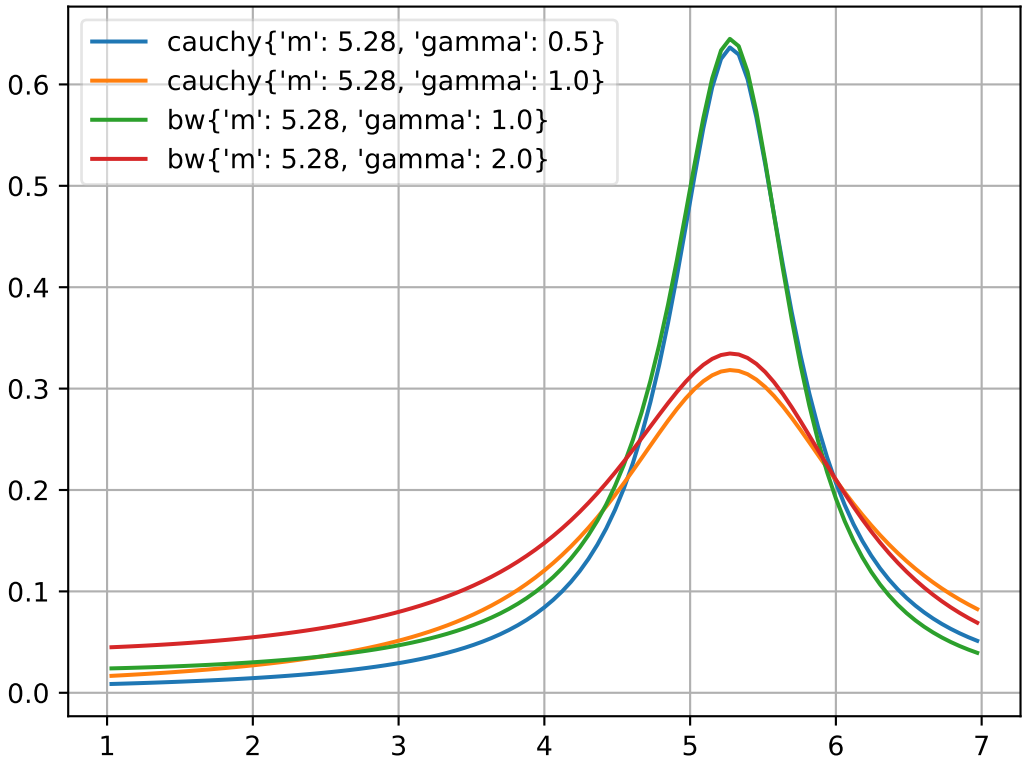
$$N = \frac{2\sqrt{2}m\Gamma\gamma}{\pi\sqrt{m^2 + \gamma}}$$

and

$$\gamma = \sqrt{m^2\left(m^2 + \Gamma^2\right)}$$

See also:

cauchy()



1.3.4 crystalball

probfit.pdf.**crystalball**(*double x*, *double alpha*, *double n*, *double mean*, *double sigma*) → double

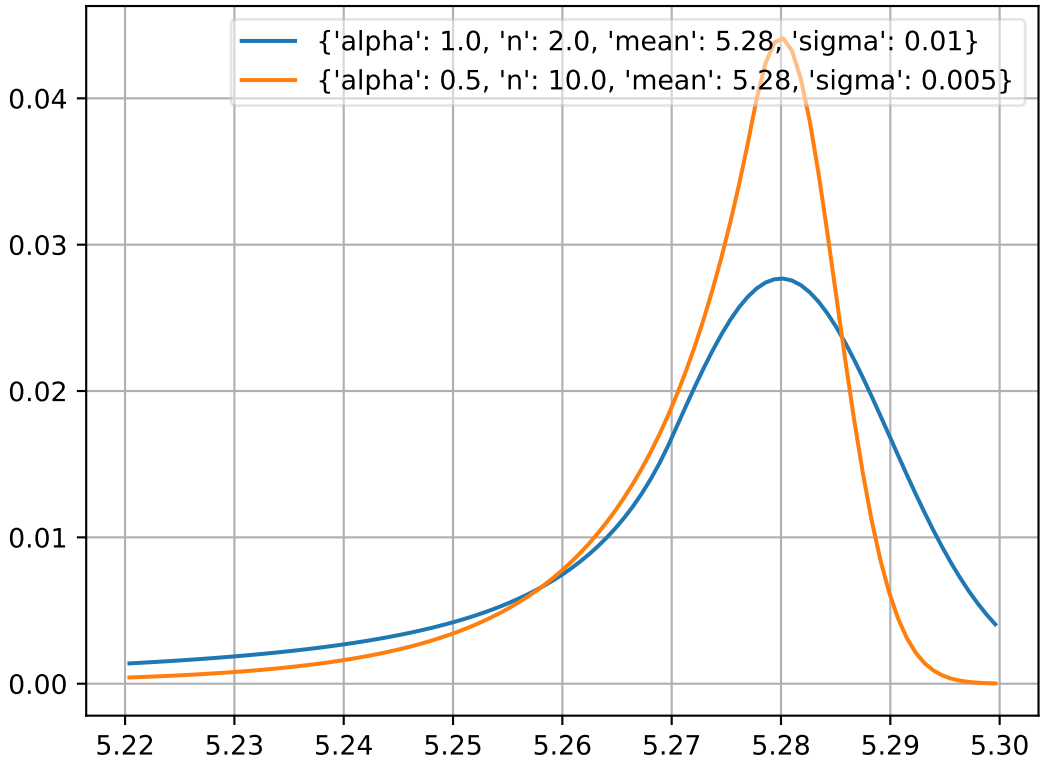
Unnormalized crystal ball function. If alpha > 0, the non-Gaussian tail is on the left. Otherwise, the tail is on the right.

$$f(x;\alpha,n,mean,\sigma) = \begin{cases} \exp\left(-\frac{1}{2}\delta^2\right) & \text{if } \delta > -\alpha \\ \left(\frac{n}{|\alpha|}\right)^n \left(\frac{n}{|\alpha|} - |\alpha| - \delta\right)^{-n} \exp\left(-\frac{1}{2}\alpha^2\right) & \text{if } \delta \leq \alpha \end{cases}$$

where

- $\delta = \frac{x-mean}{\sigma}$

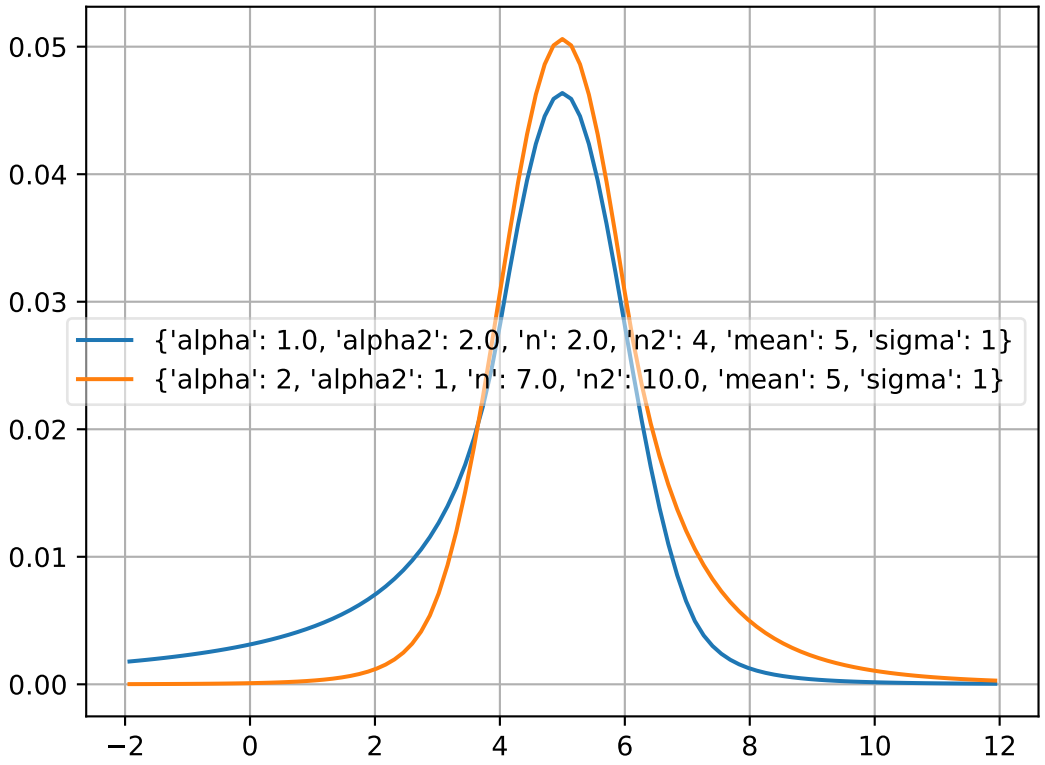
Note: http://en.wikipedia.org/wiki/Crystal_Ball_function



1.3.5 doublecrystalball

probfit.pdf.**doublecrystalball**(*double x*, *double alpha*, *double alpha2*, *double n*, *double n2*, *double mean*, *double sigma*) → double

Unnormalized double crystal ball function A gaussian core with two power tails

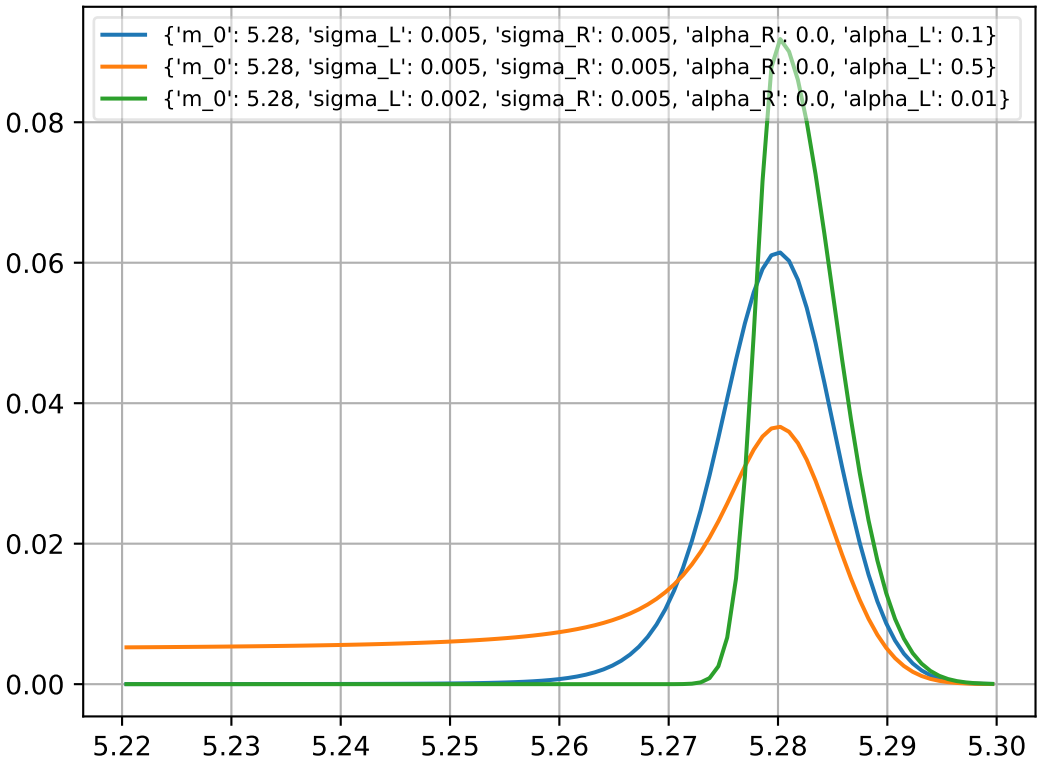


1.3.6 cruijff

probfit.pdf.**cruijff**(*double x*, *double m_0*, *double sigma_L*, *double sigma_R*, *double alpha_L*, *double alpha_R*) → double

Unnormalized cruijff function

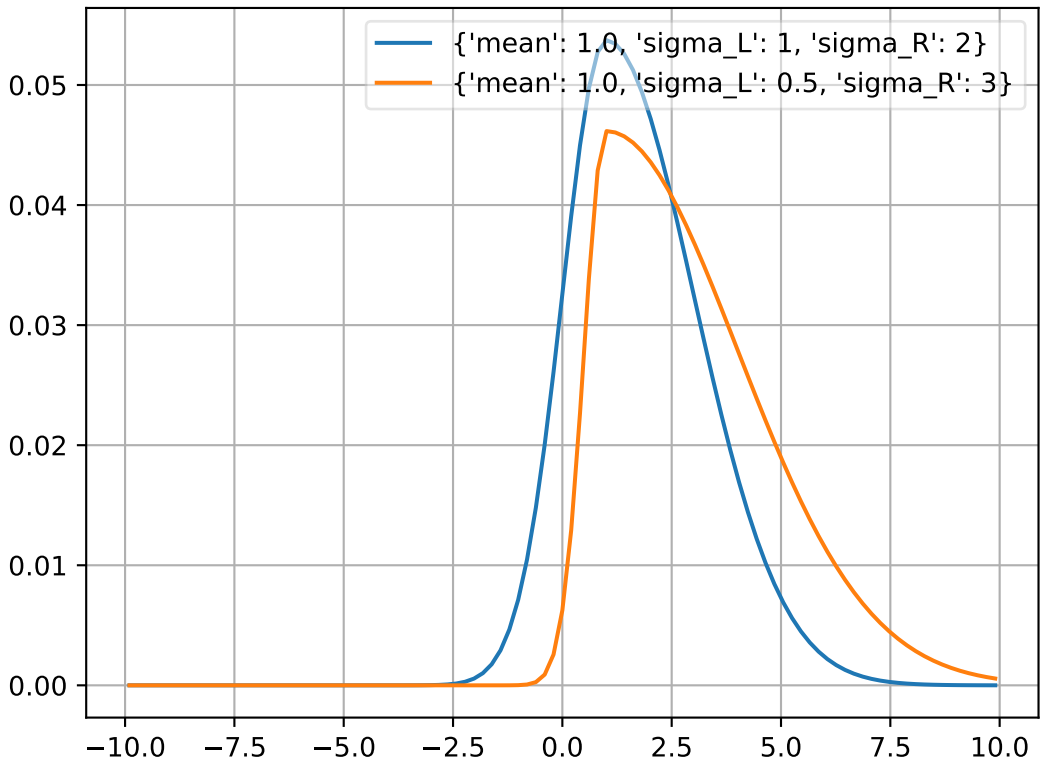
$$f(x; m_0, \sigma_L, \sigma_R, \alpha_L, \alpha_R) = \begin{cases} \exp\left(-\frac{(x-m_0)^2}{2(\sigma_L^2+\alpha_L(x-m_0)^2)}\right) & \text{if } x < m_0 \\ \exp\left(-\frac{(x-m_0)^2}{2(\sigma_R^2+\alpha_R(x-m_0)^2)}\right) & \text{if } x < m_0 \end{cases}$$


{'m_0': 5.28, 'sigma_L': 0.005, 'sigma_R': 0.005, 'alpha_R': 0.0, 'alpha_L': 0.1}
{'m_0': 5.28, 'sigma_L': 0.005, 'sigma_R': 0.005, 'alpha_R': 0.0, 'alpha_L': 0.5}
{'m_0': 5.28, 'sigma_L': 0.002, 'sigma_R': 0.005, 'alpha_R': 0.0, 'alpha_L': 0.01}
0.08
0.06
0.04
0.02
0.00
5.22
5.23
5.24
5.25
5.26
5.27
5.28
5.29
5.30

1.3.7 doublegaussian

probfit.pdf.**doublegaussian**(*double x*, *double mean*, *double sigma_L*, *double sigma_R*) → double
Unnormalized double gaussian

$$f(x; mean, \sigma_L, \sigma_R) = \begin{cases} \exp\left[-\frac{1}{2}\left(\frac{x-mean}{\sigma_L}\right)^2\right], & \text{if } x < mean \\ \exp\left[-\frac{1}{2}\left(\frac{x-mean}{\sigma_R}\right)^2\right], & \text{if } x >= mean \end{cases}$$



1.3.8 novosibirsk

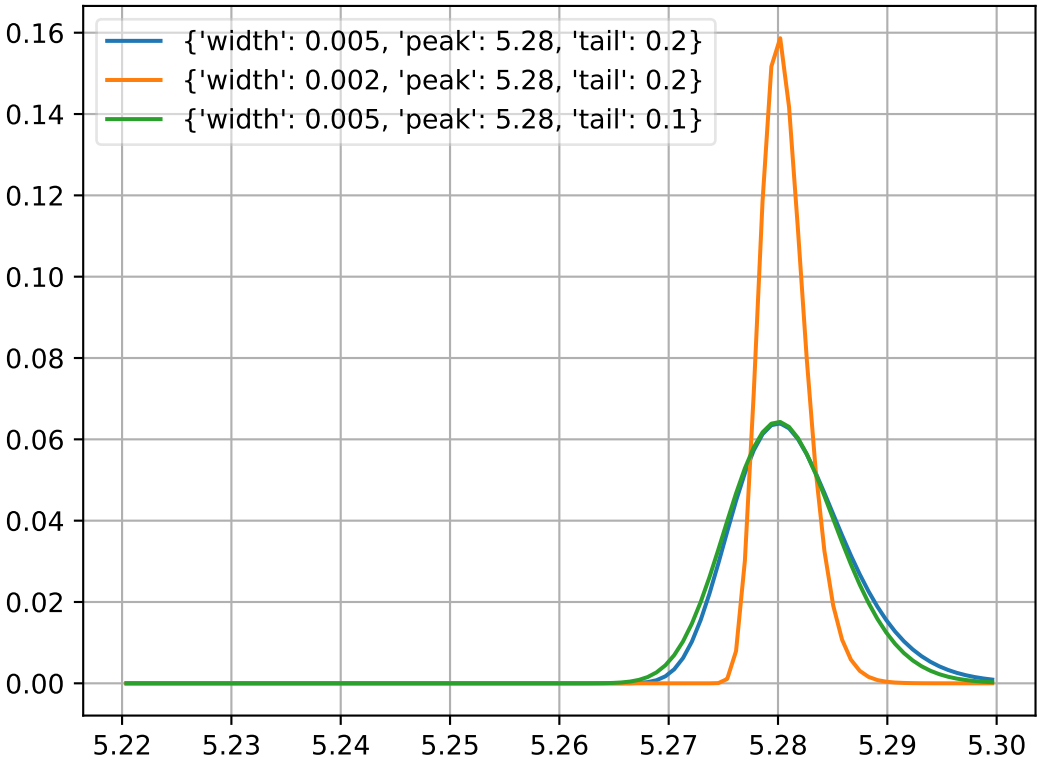
probfit.pdf.**novosibirsk**(*double x*, *double width*, *double peak*, *double tail*) → double
Unnormalized Novosibirsk

$$f(x; \sigma, x_0, \Lambda) = \exp\left[-\frac{1}{2}\frac{(\ln q_y)^2}{\Lambda^2} + \Lambda^2\right]$$

$$q_y(x; \sigma, x_0, \Lambda) = 1 + \frac{\Lambda(x - x_0)}{\sigma} \times \frac{\sinh\left(\Lambda\sqrt{\ln 4}\right)}{\Lambda\sqrt{\ln 4}}$$

where

- width = σ
- peak = m_0
- tail = Λ



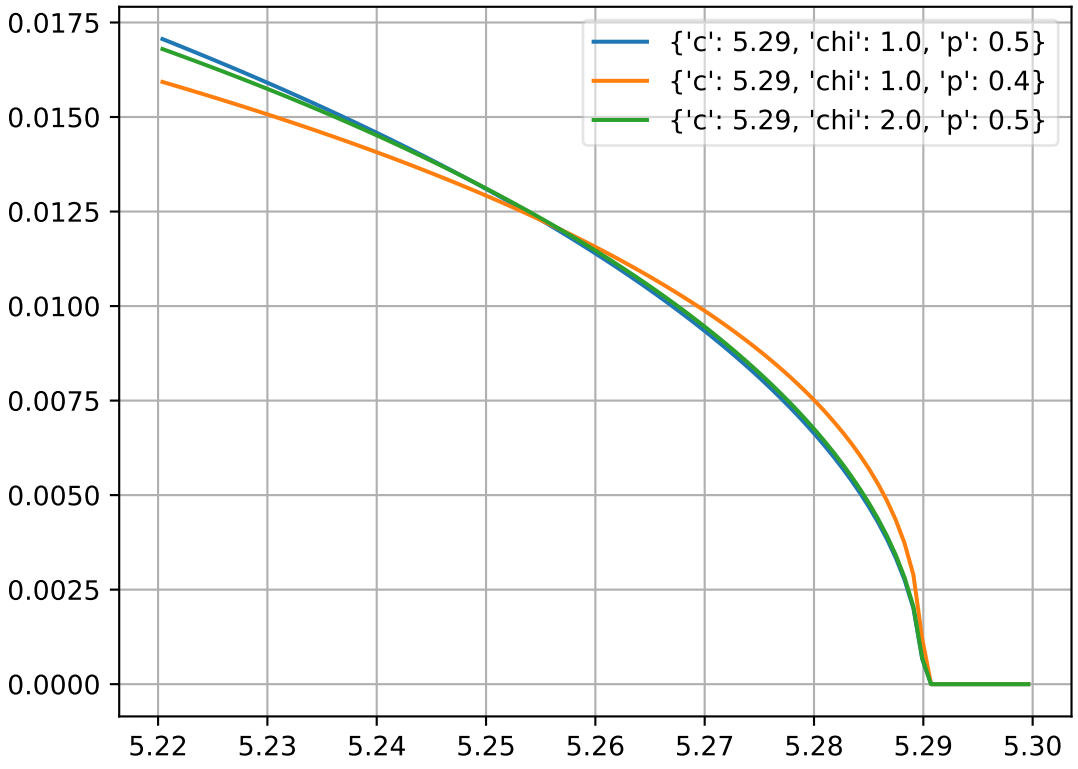
1.3.9 argus

probfit.pdf.**argus**(*double x*, *double c*, *double chi*, *double p*) → double

Unnormalized argus distribution

$$f(x; c, \chi, p) = \frac{x}{c^2}\left(1 - \frac{x^2}{c^2}\right) \exp\left(-\frac{1}{2}\chi^2\left(1 - \frac{x^2}{c^2}\right)\right)$$

Note: http://en.wikipedia.org/wiki/ARGUS_distribution


{'c': 5.29, 'chi': 1.0, 'p': 0.5}
{'c': 5.29, 'chi': 1.0, 'p': 0.4}
{'c': 5.29, 'chi': 2.0, 'p': 0.5}
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
5.22
5.23
5.24
5.25
5.26
5.27
5.28
5.29
5.30

1.3.10 linear

probfit.pdf.**linear**()
_Linear()

Linear function.

$$f(x; m, c) = mx + c$$

1.3.11 poly2

probfit.pdf.**poly2**(*double x*, *double a*, *double b*, *double c*) → double
Parabola

$$f(x; a, b, c) = ax^2 + bx + c$$

1.3.12 poly3

probfit.pdf.**poly3**(*double x*, *double a*, *double b*, *double c*, *double d*) → double
Polynomial of third order

$$f(x; a, b, c, d) = ax^3 + bx^2 + cx + d$$

1.3.13 JohnsonSU

probfit.pdf.**johnsonSU**()
_JohnsonSU(xname='x')

Normalized JohnsonSU [1]_.

$$f(x; \mu, \sigma, \nu, \tau) = \frac{1}{\lambda\sqrt{2\pi}} \frac{1}{\sqrt{1 + \left(\frac{x-\xi}{\lambda}\right)}} e^{-\frac{1}{2}\left(-\nu + \frac{1}{\tau}\sinh^{-1}\left(\frac{x-\xi}{\lambda}\right)\right)^2}$$

where

$$\lambda = \sigma \times \left(\frac{1}{2}(\exp(\tau^2) - 1)\left(\exp(\tau^2)\cosh(-\nu\tau) + 1\right)\right)^{-\frac{1}{2}}$$

and

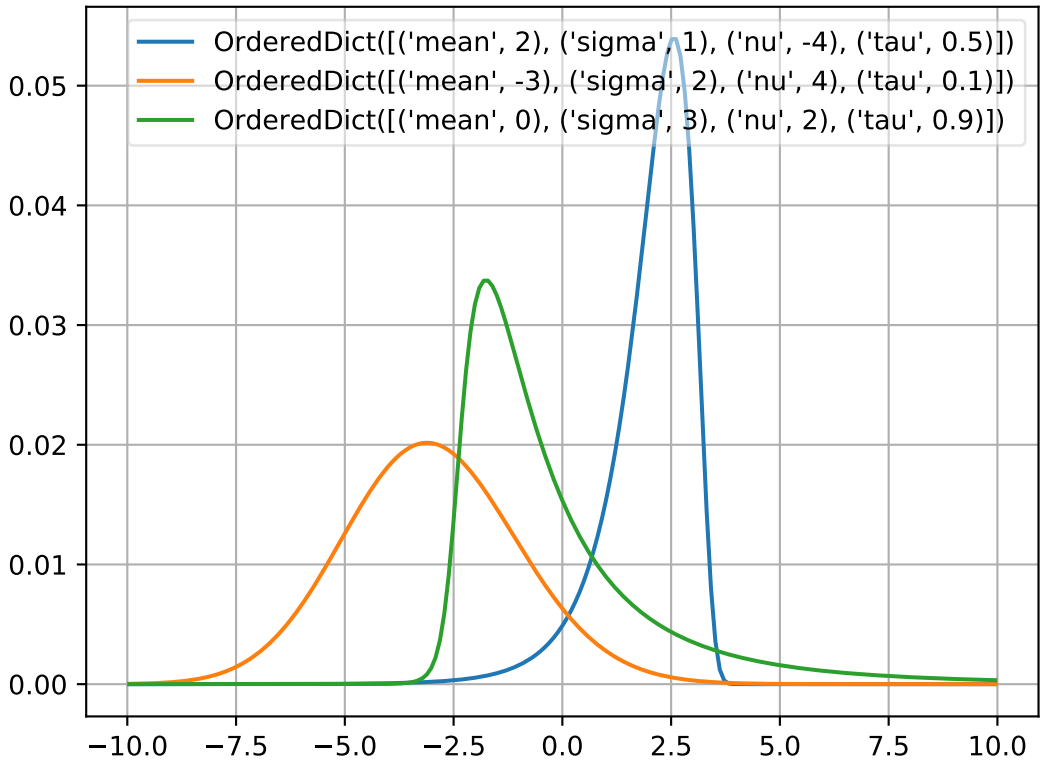
$$\xi = \mu + \lambda \exp\left(\frac{\tau^2}{2}\right)\sinh(\nu\tau)$$

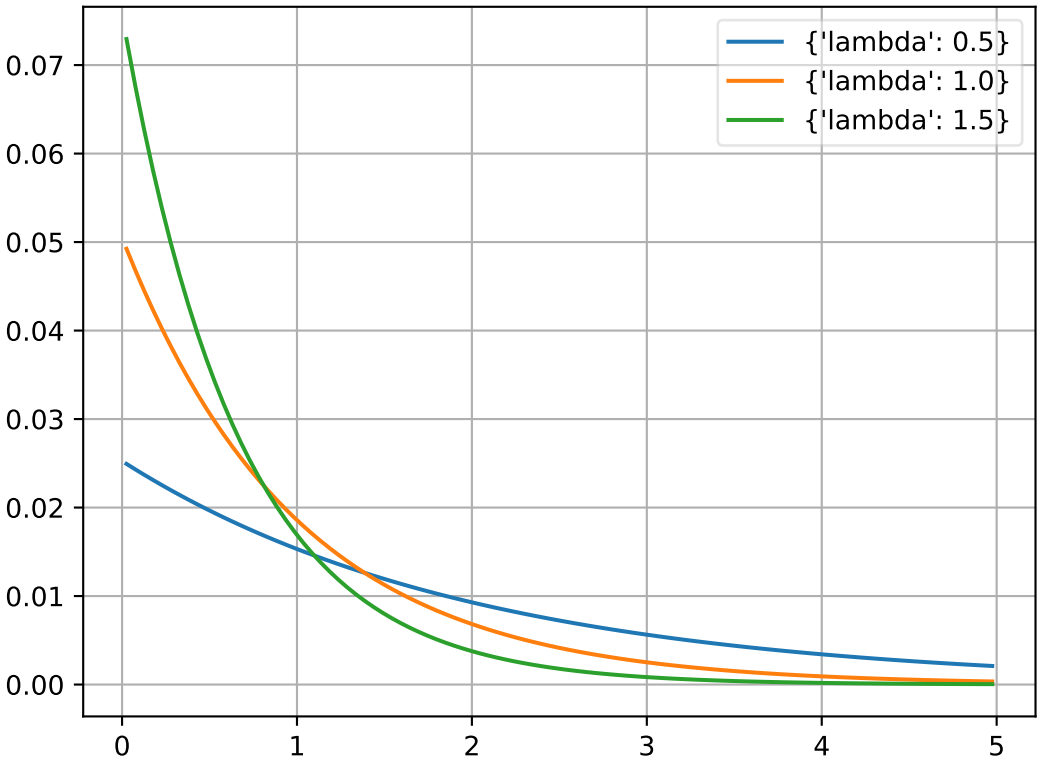
1.3.14 Exponential

probfit.pdf.**exponential**()
_Exponential(xname='x')

Exponential [1]_.

$$f(x; \tau) = \begin{cases} \exp(-\lambda x) & \text{if } \geq 0 \\ 0 & \text{if } < 0 \end{cases}$$


OrderedDict([('mean', 2), ('sigma', 1), ('nu', -4), ('tau', 0.5)])
OrderedDict([('mean', -3), ('sigma', 2), ('nu', 4), ('tau', 0.1)])
OrderedDict([('mean', 0), ('sigma', 3), ('nu', 2), ('tau', 0.9)])
0.05
0.04
0.03
0.02
0.01
0.00
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


{'lambda': 0.5}
{'lambda': 1.0}
{'lambda': 1.5}
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
1
2
3
4
5

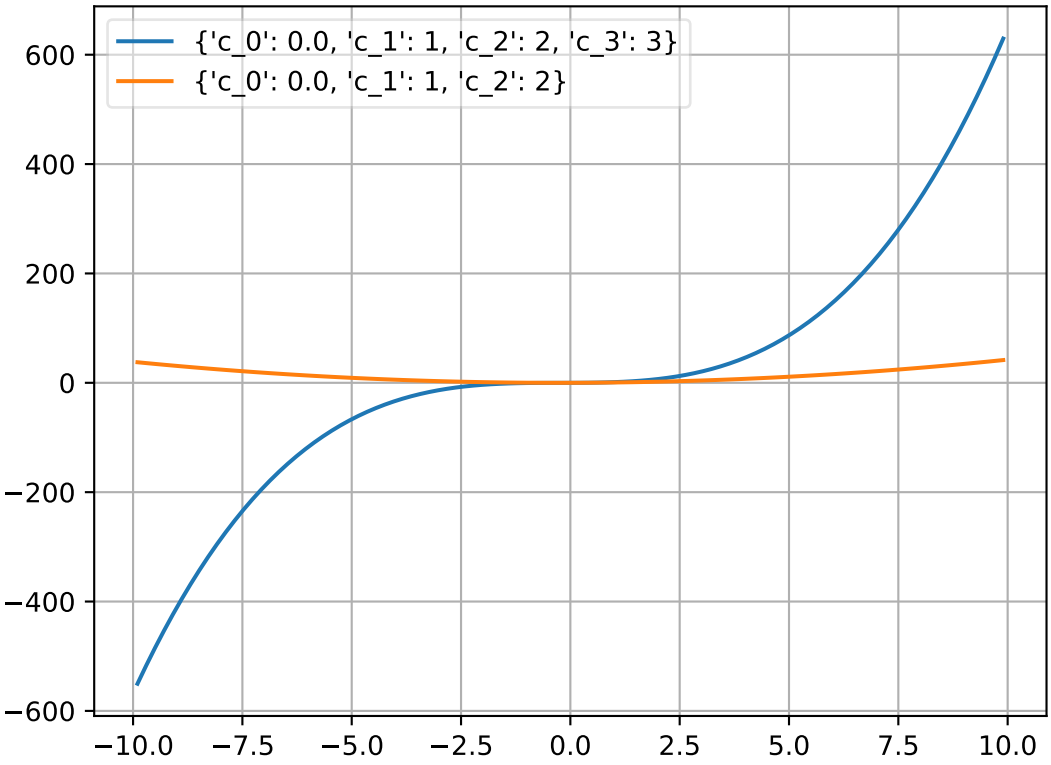
1.3.15 Polynomial

class probfit.pdf.**Polynomial**(*order*, *xname='x'*)

Polynomial.

$$f(x; c_i) = \sum_{i<\text{order}} c_i x^i$$

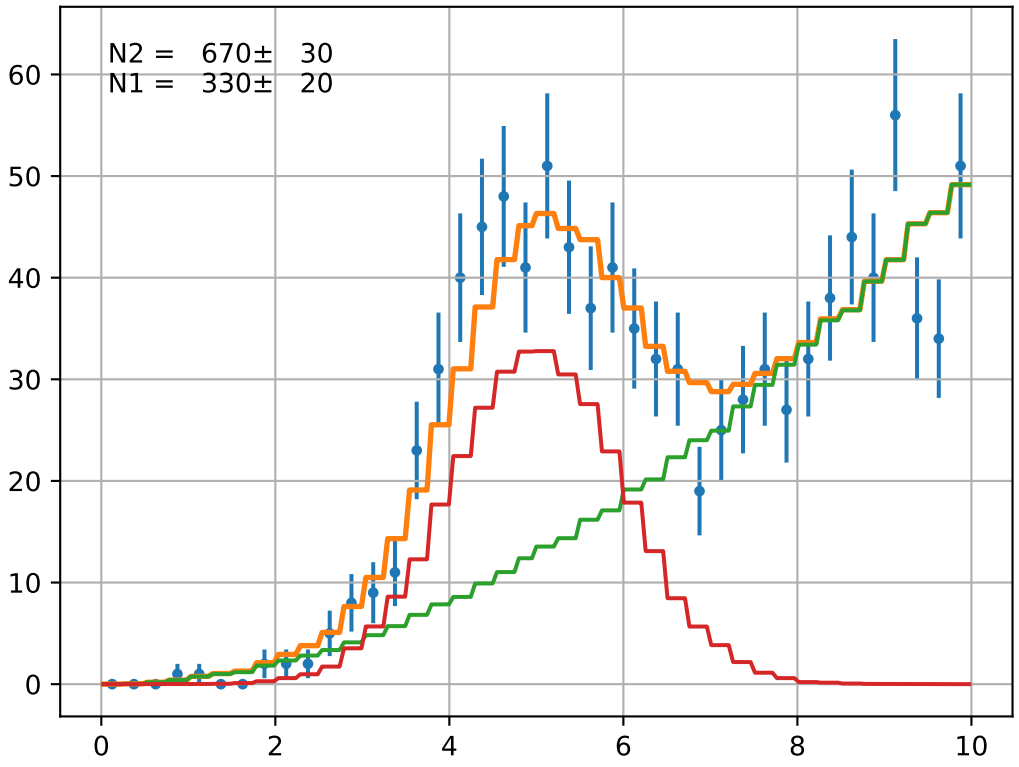
User can supply order as integer in which case it uses (c_0. . . .c_n+1) default or the list of coefficient name which the first one will be the lowest order and the last one will be the highest order. (order=1 is a linear function)



1.3.16 HistogramPdf

class probfit.pdf.**HistogramPdf**(*hy*, *binedges*, *xname='x'*)

A histogram PDF. User supplies a template histogram with bin contents and bin edges. The histogram does not have to be normalized. The resulting PDF is normalized.


N2 = 670± 30
N1 = 330± 20
60
50
40
30
20
10
0
0
2
4
6
8
10

1.4 Useful Utility Function

1.4.1 vector_apply

probfit.nputil.**vector_apply**(*f*, *x*, **arg*)

Apply **f** to array **x** with given arguments fast.

This is a fast version of:

```
np.array([f(xi,*arg) for xi in x ])
```

useful when you try to plot something.

1.4.2 draw_pdf

probfit.plotting.**draw_pdf**(*f*, *arg*, *bound*, *bins=100*, *scale=1.0*, *density=True*, *normed_pdf=False*, *ax=None*, ***kwds*)

draw pdf with given argument and bounds.

Arguments

- **f** your pdf. The first argument is assumed to be independent variable
- **arg** argument can be tuple or list
- **bound** tuple(xmin,xmax)
- **bins** number of bins to plot pdf. Default 100.
- **scale** multiply pdf by given number. Default 1.0.
- **density** plot density instead of expected count in each bin (pdf*bin width). Default True.
- **normed_pdf** Normalize pdf in given bound. Default False
- The rest of keyword argument will be pass to pyplot.plot

Returns

x, y of what's being plot

1.4.3 draw_compare_hist

probfit.plotting.**draw_compare_hist**(*f*, *arg*, *data*, *bins=100*, *bound=None*, *ax=None*, *weights=None*, *normed=False*, *use_w2=False*, *parts=False*, *grid=True*)

draw histogram of data with poisson error bar and f(x,*arg).

```
data = np.random.rand(10000)
f = gaussian
draw_compare_hist(f, {'mean':0,'sigma':1}, data, normed=True)
```

Arguments

- **f**
- **arg** argument pass to f. Can be dictionary or list.
- **data** data array

- **bins** number of bins. Default 100.
- **bound** optional boundary of plot in tuple form. If *None* is given, the bound is determined from min and max of the data. Default *None*
- **weights** weights array. Default None.
- **normed** optional normalized data flag. Default False.
- **use_w2** scaled error down to the original statistics instead of weighted statistics.
- **parts** draw parts of pdf. (Works with AddPdf and Add2PdfNorm). Default False.

1.4.4 draw_residual

`probfit.plotting.`**`draw_residual`**(*x*, *y*, *yerr*, *xerr*, *show_errbars=True*, *ax=None*, *zero_line=True*, *grid=True*, ***kwargs*)

Draw a residual plot on the axis.

By default, if show_errbars if True, residuals are drawn as blue points with errorbars with no endcaps. If show_errbars is False, residuals are drawn as a bar graph with black bars.

Arguments

- **x** array of numbers, x-coordinates
- **y** array of numbers, y-coordinates
- **yerr** array of numbers, the uncertainty on the y-values
- **xerr** array of numbers, the uncertainty on the x-values
- **show_errbars** If True, draw the data as a bar plot, else as an errorbar plot
- **ax** Optional matplotlib axis instance on which to draw the plot
- **zero_line** If True, draw a red line at $y = 0$ along the full extent in x
- **grid** If True, draw gridlines
- **kwargs** passed to `ax.errorbar` (if `show_errbars` is True) or `ax.bar` (if `show_errbars` if False)

Returns

The matplotlib axis instance the plot was drawn on.

CHAPTER

TWO

COOKBOOK

2.1 Tell probfit to use my analytical integral

probfit checks for a method `integrate(bound, nint, *arg)`. If such method is available it calls that method to compute definite integral. If not it falls back to simpson3/8 integration (cubic approximation).

```
from probfit import integrate1d
def line(x, m, c):
    return m*x+c

# compute integral of line from x=(0,1) using 10 intevals with m=1. and c=2.
# all probfit internal use this

# no integrate method available probfit use simpson3/8
print integrate1d(line, (0,1), 10, (1.,2.) )

# Let us illustrate the point by forcing it to have integral that's off by
# factor of two
def wrong_line_integral(bound, nint, m, c):
    a, b = bound
    return 2*(m*(b**2/2.-a**2/2.)+c*(b-a)) # I know this is wrong

line.integrate = wrong_line_integral
# line.integrate = lambda bound, nint, m, c: blah blah # this works too

# yes off by factor of two
print integrate1d(line, (0,1), 10, (1.,2.))
```

CHAPTER

THREE

DEVELOPMENT

Contributions to probfit are welcome. You should fork the repository, create a branch in your fork, and then open a pull request.

Developing probfit requires a few dependencies, in addition to those required for users. The following commands should create a suitable environment, assuming you've cloned your fork of probfit and are in the repository root. (You may wish to work inside a virtual environment to isolate these packages from your system install.)

```
$ pip install cython pytest pytest-mpl pylint flake8 sphinx sphinx_rtd_theme
$ make build
```

Installing Cython will allow you to build the C extensions. Pylint and flake8 are used for linting, pytest for testing, and Sphinx for generating the HTML documentation.

When developing, be sure to regularly run the test suite to see if anything's broken. The suite is run automatically against Python 2 and 3 when you open a pull request, and also when you push subsequent commits. It can be run locally with:

```
$ make test
$ make code-analysis
```

To build and view the documentation, run `make doc-show`.

For a list of everything you can do, run `make help`. If you run into any trouble, please open an issue.

CHAPTER

FOUR

DOWNLOAD & INSTALL

From pip:

```
pip install probfit
```

or get the latest development from github:

```
pip install git+https://github.com/scikit-hep/probfit.git
```

CHAPTER

FIVE

TUTORIAL

The tutorial consists of an IPython notebook in the tutorial directory. You can view it online too.

CHAPTER

SIX

COMMONLY USED API

Refer to *Full API Documentation* for complete reference.

6.1 Cost Functions

Refer to *Cost Function*.

`UnbinnedLH`(f, data[, weights, extended, ...])	Construct -log(unbinned likelihood) from callable *f* and data points *data*.
`BinnedLH`(f, data[, bins, weights, ...])	Create a Poisson Binned Likelihood object from given PDF **f** and **data** (raw points not histogram).
`Chi2Regression`(f, x, y[, error, weights])	Create χ^2 regression object.
`BinnedChi2`(f, data[, bins, weights, bound, ...])	Create Binned Chi2 Object.
`SimultaneousFit`([factors, prefix, skip_prefix])	Construct simultaneous fit from given cost functions.

6.2 Functors

Refer to *Functor*

`Normalized`(f, bound[, nint, warnfloat])	Transformed PDF in to a normalized version.
`Extended`(f[, extname])	Transformed given **f** into extended from.
`Convolve`(f, g, gbound[, nbins])	Make convolution from supplied **f** and **g**.
`AddPdf`([prefix, factors, skip_prefix])	Directly add PDF without normalization nor factor.
`AddPdfNorm`([facname, prefix, skip_prefix])	Add PDF with normalization factor.
`rename`(f, newarg)	Rename function parameters.

And corresponding decorator

`normalized`(bound[, nint])	Normalized decorator
`extended`([extname])	Extended decorator

6.3 Builtin Functions

Refer to *Builtin PDF*. This list can grow: implement your favorite function and send us pull request.

gaussian(double x, double mean, double sigma)	Normalized gaussian.
crystalball(double x, double alpha, ...)	Unnormalized crystal ball function.
doublecrystalball(double x, double alpha, ...)	Unnormalized double crystal ball function A gaussian core with two power tails
cruijff(double x, double m_0, ...)	Unnormalized cruijff function
cauchy(double x, double m, double gamma)	Cauchy distribution aka non-relativistic Breit-Wigner
rtv_breitwigner(double x, double m, double gamma)	Normalized Relativistic Breit-Wigner
doublegaussian(double x, double mean, ...)	Unnormalized double gaussian
johnsonSU	_JohnsonSU(xname='x')
argus(double x, double c, double chi, double p)	Unnormalized argus distribution
linear	_Linear()
poly2(double x, double a, double b, double c)	Parabola
poly3(double x, double a, double b, ...)	Polynomial of third order
novosibirsk(double x, double width, ...)	Unnormalized Novosibirsk
HistogramPdf(hy, binedges[, xname])	A histogram PDF.
Polynomial(order[, xname])	Polynomial.

6.4 Useful utility

You may find these functions useful in interactive environment.

vector_apply(f, x, *arg)	Apply **f** to array **x** with given arguments fast.
draw_pdf(f, arg, bound[, bins, scale, ...])	draw pdf with given argument and bounds.
draw_compare_hist(f, arg, data[, bins, ...])	draw histogram of data with poisson error bar and f(x,*arg).

CHAPTER

SEVEN

COOKBOOK

Cookbook recipies are at *Cookbook*.

CHAPTER

EIGHT

DEVELOPMENT

If you'd like to develop with the probfit source code, see the *Development* section.

INDEX

Symbols

A

B

C

D

E

G

H

J

L

N

P

R

S

U

V