Heppy

Sep 05, 2023

Contents:

The Heppy package provides useful data structures and tools for high energy physics.

CHAPTER 1

Histogram

1.1 Base histogram

The base histogram class from which histograms of a specific dimension inherit.

class heppy.**basehistogram**(*binedges*, *contents*, *areas=False*, *name=''*, *uncorr_variations={}*, *corr_variations={}*, *attributes={}*, *plot_attributes={}*)

Base class for one-dimensional and two-dimensional histograms that keep track of their various uncertainty contributions and arbitrary attributes (useful for labeling and plotting).

Parameters

- **binedges** (numpy.array, or tuple of numpy.array) – bin edges, including uppermost. For 1D histograms, a numpy.array. For 2D histograms, a tuple of two numpy.array (in the x and y direction, respectively).
- **contents** (numpy.array) – the "bin contents", which are either bin areas (= what ROOT calls "bin contents") or bin heights (= bin areas / bin sizes). See also argument areas.
- **areas** (bool) – if True, interpret given contents as bin areas, else as bin heights
- **name** (str) – a name for the histogram. This is only separate from the other attributes because it is so commonly used and is automatically created for histograms produced by mathematically combining two histograms. E.g. dividing two histograms with names 'foo' and 'bar' will return a histogram with name 'foo / bar'.
- **uncorr_variations** (dict) – dictionary of variations that are uncorrelated between bins (e.g. statistical uncertainty). Keys are variation names, values are np.array objects of the same dimension as the nominal contents.
- **corr_variations** (dict) – dictionary of variations that are fully correlated between bins (e.g. systematic uncertainty). Keys are variation names, values are np.array objects of the same dimension as the nominal contents.
- **attributes** (dict) – dictionary of completely arbitrary attributes that the user can provide/change/access. E.g. information about the data sample that produced the histogram.

- **plot_attributes** (dict) – dictionary of completely arbitrary that the user can provide/change/access. This one is more intended for information on how to visualise/plot the histogram. It is especially useful if working with *heppy.make_figure*, which will *assume* that all the plot_attributes correspond to keyword arguments that are understood by Matplotlib's `plot()` and/or `fill_between()` functions

extract_variation_histogram(*variation*, ***kwargs*)
Get a new histogram object that has a given variation as nominal. Useful e.g. for studying a particular systematic variation.

Parameters

- **variation** (str) – name of the variation
- ****kwargs** – get passed on to constructor of new histogram, e.g. useful to set a `name` for the new histogram.

Returns new *heppy.histogram* that has a given `variation` as nominal

Raises

- **KeyError** – if variation not found in either uncorrelated or correlated variations
- **RuntimeError** – if variation found in both uncorrelated or correlated variations

binsizes
Bin sizes.

For a one-dimensional histogram, returns an array of dimension (N, 1), where N is the number of bins. The elements represent the width of each bin.

For a two-dimensional histogram, returns an array of dimension (N, M), where N is the number of bins along the first axis ("x-axis") and M is the number of bins along the second axis ("y-axis"). The elements represent the area of each bin.

heights
Bin heights, equal to bin areas divided by the corresponding bin sizes

set_heights(*heights*)
Set bin heights to an array of the same dimension as the current areas or to a scalar

integral(*variations=None*, ***kwargs*)
Calculate the integral of the histogram.

Parameters

- **variations** (list of str or str) – if given, a tuple of the nominal integral and its upper and lower variation is calculated. This argument is passed to *histogram1d.net_variations()* and should be a list of considered variation names or the string `'all'`.
- ****kwargs** – additional keyword arguments that get passed to *histogram1d.net_variations()*

Returns the integral (as well as upper and lower variation if `variations` is given)

Return type float, or if `variations` are given, tuple of nominal as well as upper and lower variation

net_variations(*variations='all'*, *subtract_nominal=False*, *relative=False*)
Return upper and lower net areas variation of the histogram as a tuple.

@variations should be a sequence of considered variation names or the string 'all' @subtract_nominal: if True, return the differences with respect to the nominal areas @relative: if True, divide by the nominal areas

CAUTION: this method cannot yet deal with systematic uncertainties for which the up- and down-shift lie on the same side of the nominal. This is because the variations are fundamentally treated independently of each other, so there is no sense of the up- and down-shift being related to the same underlying uncertainty source.

errorbars (*variations='all'*)
Returns upper and lower error bars, defined as the absolute net variations (taking into account the given variations) with the nominal values subtracted.

__add__ (*other*)
Add another histogram or a scalar to this histogram.

Returns the result of the addition as a histogram.

Correlated variations are treated as fully correlated among the two histograms if they have the same name, otherwise they are treated as uncorrelated. Uncorrelated variations are treated as uncorrelated between the two histograms.

__sub__ (*other*)
Subtract another histogram or a scalar from this histogram.

Returns the result of the subtraction as a histogram.

Correlated variations are treated as fully correlated among the two histograms if they have the same name, otherwise they are treated as uncorrelated. Uncorrelated variations are treated as uncorrelated between the two histograms.

__mul__ (*other*)
Multiply another histogram or a scalar binwise with this histogram.

Returns the result of the binwise multiplication as a histogram.

Correlated variations are treated as fully correlated among the two histograms if they have the same name, otherwise they are treated as uncorrelated. Uncorrelated variations are treated as uncorrelated between the two histograms.

__truediv__ (*other*)
Divide by another histogram or a scalar binwise.

Returns the result of the binwise division as a histogram.

Correlated variations are treated as fully correlated among the two histograms if they have the same name, otherwise they are treated as uncorrelated.

CAUTION: Uncorrelated variations are treated as uncorrelated between the two histograms. If the uncorrelated variations represent statistical uncertainties, this means that the division treats the two histograms as statistically uncorrelated.

See also *histdiv()*

apply_inplace (*function*, *new_binedges=None*)
Call a function on the nominal areas as well as all varied areas (in `corr_variations` and `uncorr_variations`), modifying the existing histogram.

It is possible to convert the histogram to a different type (e.g. *histogram2d* → *histogram1d*) by giving new binedges of the desired new dimensionality. If the new binedges have a dimension other than 1D or 2D, the type will become *basehistogram*.

For a version of this method that leaves the original histogram unchanged and returns a copy with the function applied (and optionally new bin edges), see *basehistogram.apply()*. I HIGHLY recommend using that if the new histogram will be of different type to avoid confusion!

Caution: if you change the binning, it is your responsibility that uncertainties encoded in the variations are handled correctly.

Parameters

- **function** (function) – function taking a numpy.array as argument
- **new_binedges** – optional argument to set new binedges. If the binedges change the dimension of the histogram (e.g. from 2D to 1D), the histograms' type is changed accordingly

Example 1: taking the sine of the areas

```
import heppy as hp
import numpy as np
foo = hp.histogram1d(np.array([0., 1., 2., 3.]), [5., 6., 7])
foo.apply_inplace(np.sin)
```

Example 2: projecting a 2D histogram to its x-axis (integrating over the y-axis)

```
import heppy as hp
import numpy as np
binedges = (np.array([0., 10., 20.]), np.array([10., 20., 30.]))
areas = [[0.1, 0.2], [0.3, 0.4]]
foo = hp.histogram2d(binedges, areas, areas=True)
from functools import partial
project_x = partial(np.sum, axis=1)
foo.apply_inplace(project_x, new_binedges=foo.binedges[0])
```

Raises ValueError if the shape of the areas after the function is called on them does not match the shape of the bin edges (after setting them to new_binedges if given)

apply (*function*, *new_binedges=None*)
Same as *basehistogram.apply_inplace()*, except that the resulting histogram is returned (as an independent object) with the function applied, while the original histogram is not modified.

Returns histogram with function applied and possibly new bin edges (*histogram1d*, *histogram2d*, or *basehistogram*)

clip (*minimum=None*, *maximum=None*)
Clips the bin areas at a minimum and/or maximum value.

Similar to numpy.clip (see here).

Both the nominals and all correlated and uncorrelated variations of the histogram are clipped. *Note that the net variation of multiple individual variations considered together may still lie outside of the clip range!*

Parameters **minimum** – minimum accepted value; areas below this value will be set to the value.

If None, no minimum is imposed. :type minimum: float or None :param maximum: maximum accepted value; areas above this value will be set to the value. If None, no maximum is imposed. :type maximum: float or None

The behaviour is like that of numpy.clip, e.g. with regards to what happens if minimum is greater than maximum.

"Undocumented" feature: you can actually pass a Numpy `array_like` as `minimum` or `maximum`, not just a single `float`!

to_file (*outfilename*, *key*, *recreate=False*)
Write histogram to text file. Multiple histograms with different keys can be written to the same file.

Parameters

- **outfilename** (`str`) – name of the file that the histogram is written to
- **key** (`bool`) – name/key of the histogram inside the output file
- **recreate** – option to recreate the output file rather than append to it

1.2 One-dimensional histogram

A histogram class with bins along one axis.

class heppy.**histogram1d**(**args*, ***kwargs*)
Heppy one-dimensional histogram. This has functionality for rebinning, getting various representations for plotting (curve, points, errorbars, errorbands), as well as performing mathematical operations (these have only been implemented for one-dimensional histograms so far).

nbins
Returns the number of bins in the histogram.

Returns number of bins in the histogram

Return type `int`

binwidths
Bin widths is an alias for bin sizes in the case of a one-dimensional histogram

bin_index (*x*)
Returns the index of the bin that contains the given x-value.

Lower bin edges are included in a bin, upper bin edges are excluded (same as in the ROOT convention).

Parameters **x** (`float`) – x-value

Returns index of bin that contains the x-value

Return type `int`

Raises **ValueError** – if x-value lies outside of the outer bin edges of the histogram

Example:

```
>>> h = histogram1d([0., 1., 2.], [10., 11.])
>>> h.bin_index(0.5)
0
>>> h.bin_index(0.)
0
>>> h.bin_index(1.0)
1
>>> h.bin_index(2.0)
ValueError: Cannot find index of bin containing x = 2.0, which is outside of␣
↪histogram x-boundaries [0.0, 2.0)
>>> h.bin_index(-1.0)
ValueError: Cannot find index of bin containing x = -1.0, which is outside of␣
↪histogram x-boundaries [0.0, 2.0)
```

curve (*variation=''*)
Curve representation of histogram @variation: if given, return the curve for the variation of this name. Otherwise, return the nominal curve

points (*variation=''*, *shift=0.0*, *abs_shift=False*)
Point representation of histogram If @shift is given, the x-coordinates of the midpoints are given shifted by this absolute x-value (if @abs_shift=True) or relative fraction of the corresponding bin's width (if @abs_shift=False)

errorband (**args*, ***kwargs*)
Basically same as errorbars method, only in curve representation @*args and @**kwargs get passed on to self.net_variations()

rebin (*newedges*)
Rebin to @newedges Each element of @newedges should correspond to an existing binedge, i.e. only existing bins are merged

CAUTION: currently ASSUMES that each uncorrelated variation only has shifts in one direction of the nominal (i.e. it is either higher or lower everywhere)!

merge_bins (*xmin*, *xmax*)
Merge the bins falling into the given x-range into one bin

squash_highest_bin (*squash_above*, *new_xmax*)
Merge all bins from @squash_above upwards and set the highest bin edge to @new_xmax.

Does nothing if squash_above is >= the highest bin edge.

height (*bin_index*)
Returns the height of the given bin index with uncertainties.

Returns height of the indexed bin including its variations

Return type `heppy.value`

Usage example:

```
>>> import heppy as hp
>>> h = hp.histogram1d([0., 1., 3.], [10., 11.], corr_variations={'systematic_
↪_up' : [13., 11.5]})
>>> v = h.height(1)
>>> v.nominal
11.0
>>> v.corr_variations['systematic__up']
11.5
```

iterheights ()
Generates iterator over heights.

Returns bin heights including their variations

Return type `heppy.value`

Usage example:

```
>>> import heppy as hp
>>> h = hp.histogram1d([0., 1., 3.], [10., 11.], corr_variations={'systematic_
↪_up' : [13., 11.5]})
>>> for height in h.iterheights(): print(height.nominal, height.corr_
↪variations['systematic__up'])
10.0 13.0
11.0 11.5
```

iterbins()
Generates iterator over bins, yielding bin edges and heights.

Returns bin egdes and nominal bin height

Return type `tuple` of the following: `tuple` of two `float`, and one `float`

Usage example:

```
>>> import heppy as hp
>>> h = hp.histogram1d([0., 1., 3.], [10., 11.], corr_variations={'systematic_
↪_up' : [13., 11.5]})
>>> for binedges, height in h.iterbins(): print(binedges, height.nominal)
(0.0, 1.0) 10.0
(1.0, 3.0) 11.0
>>> for binedges, height in h.iterbins(): print(binedges, height.nominal,␣
↪height.corr_variations['systematic__up'])
(0.0, 1.0) 10.0 13.0
(1.0, 3.0) 11.0 11.5
```

cumulative(*side='left'*, *nan_to_num=True*)
Returns a new histogram containing the cumulative sums of the bin areas.

Analogous to Numpy's `cumsum` function.

Note: the plot attributes are copied over from `self` to the returned histogram.

Parameters

- **side** (`str`) – which side to integrate from, may be either `'left'` or `'right'`
- **nan_to_num** (`bool`) – treat NaN as 0 when taking the cumulative sum?

Returns histogram of the cumulative sum of the bin areas

Return type *heppy.histogram1d*

to_yoda(*identifier*, *metadata={}*)
Returns the histogram in YODA output format as a string.

See the websites of YODA and its main user Rivet for more information.

Parameters

- **identifier** (`str`) – in-file identifier for the histogram, e.g. `'/REF/ATLAS_2017_I1614149/d16-x01-y02'`
- **metadata** (`dict`) – optional dictionary of metadata. E.g. for Rivet use, one could have `metadata = {'IsRef' : 1, 'Path' : '/REF/ATLAS_2017_I1614149/d16-x01-y02', 'Title' : 'doi:10.17182/hepdata.80041.v2/t16'}`

Returns histogram formatted as YODA input string

Return type `str`

to_rivet(*identifier*, *metadata={}*)
Returns the histogram in YODA output format as a string.

See the websites of YODA and its main user Rivet for more information.

Parameters

- **identifier** (`str`) – in-file identifier for the histogram, e.g. `'/REF/ATLAS_2017_I1614149/d16-x01-y02'`

- **metadata** (dict) – optional dictionary of metadata. E.g. for Rivet use, one could have `metadata = {'IsRef' : 1, 'Path' : '/REF/ATLAS_2017_I1614149/d16-x01-y02', 'Title' : 'doi:10.17182/hepdata.80041.v2/t16'}`

Returns histogram formatted as YODA input string

Return type str

to_root(*nominal_label='nominal'*, *key_form='{name}_{variation}'*, *errors=None*)

Returns a dictionary of variation names mapped to ROOT TH1D's.

Parameters

- **nominal_label** (str) – dictionary key to give to the nominal histogram
- **key_form** (str) – template for keys in the returned dictionary. The names of the histograms will also be set to these values. It accepts the formatting fields *name* (for *self.name*) and variation (for the name of the variation)
- **errors** – optional array of bin errors. Must have the same dimensionality

as the histogram areas. Note that the same errors are used for all variation histograms. :type errors: np.array or None

Returns dictionary of ROOT histograms (values) mapped by variation name (keys)

Return type dict of str and ROOT.TH1D

to_root_file(*filename*, *recreate=False*, ***kwargs*)

Writes this histogram to a ROOT file as a set of TH1D's.

Parameters

- **filename** (str) – name or path to the ROOT file. May exist or not.
- **recreate** (bool) – only has an effect if the ROOT file exists already. If *True*, it will be overwritten. Otherwise, the new histograms will be added to the existing file.
- ****kwargs** – keyword arguments that get passed on to *histogram1d.to_root()*

__add__(*other*)

Add another histogram or a scalar to this histogram.

Returns the result of the addition as a histogram.

Correlated variations are treated as fully correlated among the two histograms if they have the same name, otherwise they are treated as uncorrelated. Uncorrelated variations are treated as uncorrelated between the two histograms.

__mul__(*other*)

Multiply another histogram or a scalar binwise with this histogram.

Returns the result of the binwise multiplication as a histogram.

Correlated variations are treated as fully correlated among the two histograms if they have the same name, otherwise they are treated as uncorrelated. Uncorrelated variations are treated as uncorrelated between the two histograms.

__sub__(*other*)

Subtract another histogram or a scalar from this histogram.

Returns the result of the subtraction as a histogram.

Correlated variations are treated as fully correlated among the two histograms if they have the same name, otherwise they are treated as uncorrelated. Uncorrelated variations are treated as uncorrelated between the two histograms.

__truediv__(*other*)
Divide by another histogram or a scalar binwise.

Returns the result of the binwise division as a histogram.

Correlated variations are treated as fully correlated among the two histograms if they have the same name, otherwise they are treated as uncorrelated.

CAUTION: Uncorrelated variations are treated as uncorrelated between the two histograms. If the uncorrelated variations represent statistical uncertainties, this means that the division treats the two histograms as statistically uncorrelated.

See also *histdiv()*

apply(*function*, *new_binedges=None*)
Same as *basehistogram.apply_inplace()*, except that the resulting histogram is returned (as an independent object) with the function applied, while the original histogram is not modified.

Returns histogram with function applied and possibly new bin edges (*histogram1d*, *histogram2d*, or *basehistogram*)

apply_inplace(*function*, *new_binedges=None*)
Call a function on the nominal areas as well as all varied areas (in `corr_variations` and `uncorr_variations`), modifying the existing histogram.

It is possible to convert the histogram to a different type (e.g. *histogram2d* → *histogram1d*) by giving new binedges of the desired new dimensionality. If the new binedges have a dimension other than 1D or 2D, the type will become *basehistogram*.

For a version of this method that leaves the original histogram unchanged and returns a copy with the function applied (and optionally new bin edges), see *basehistogram.apply()*. I HIGHLY recommend using that if the new histogram will be of different type to avoid confusion!

Caution: if you change the binning, it is your responsibility that uncertainties encoded in the variations are handled correctly.

Parameters

- **function** (`function`) – function taking a `numpy.array` as argument
- **new_binedges** – optional argument to set new binedges. If the binedges change the dimension of the histogram (e.g. from 2D to 1D), the histograms' type is changed accordingly

Example 1: taking the sine of the areas

```
import heppy as hp
import numpy as np
foo = hp.histogram1d(np.array([0., 1., 2., 3.]), [5., 6., 7])
foo.apply_inplace(np.sin)
```

Example 2: projecting a 2D histogram to its x-axis (integrating over the y-axis)

```
import heppy as hp
import numpy as np
binedges = (np.array([0., 10., 20.]), np.array([10., 20., 30.]))
areas = [[0.1, 0.2], [0.3, 0.4]]
foo = hp.histogram2d(binedges, areas, areas=True)
```

(continues on next page)

(continued from previous page)

```
from functools import partial
project_x = partial(np.sum, axis=1)
foo.apply_inplace(project_x, new_binedges=foo.binedges[0])
```

Raises `ValueError` if the shape of the areas after the function is called on them does not match the shape of the bin edges (after setting them to `new_binedges` if given)

binsizes

Bin sizes.

For a one-dimensional histogram, returns an array of dimension (N, 1), where N is the number of bins. The elements represent the width of each bin.

For a two-dimensional histogram, returns an array of dimension (N, M), where N is the number of bins along the first axis ("x-axis") and M is the number of bins along the second axis ("y-axis"). The elements represent the area of each bin.

clip(*minimum=None*, *maximum=None*)

Clips the bin areas at a minimum and/or maximum value.

Similar to `numpy.clip` (see here).

Both the nominals and all correlated and uncorrelated variations of the histogram are clipped. *Note that the net variation of multiple individual variations considered together may still lie outside of the clip range!*

Parameters **minimum** – minimum accepted value; areas below this value will be set to the value.

If `None`, no minimum is imposed. :type minimum: `float` or `None` :param maximum: maximum accepted value; areas above this value will be set to the value. If `None`, no maximum is imposed. :type maximum: `float` or `None`

The behaviour is like that of `numpy.clip`, e.g. with regards to what happens if `minimum` is greater than `maximum`.

"Undocumented" feature: you can actually pass a Numpy `array_like` as `minimum` or `maximum`, not just a single `float`!

errorbars(*variations='all'*)

Returns upper and lower error bars, defined as the absolute net variations (taking into account the given variations) with the nominal values subtracted.

extract_variation_histogram(*variation*, ***kwargs*)

Get a new histogram object that has a given variation as nominal. Useful e.g. for studying a particular systematic variation.

Parameters

- **variation** (`str`) – name of the variation
- ****kwargs** – get passed on to constructor of new histogram, e.g. useful to set a `name` for the new histogram.

Returns new *heppy.histogram* that has a given `variation` as nominal

Raises

- **KeyError** – if variation not found in either uncorrelated or correlated variations
- **RuntimeError** – if variation found in both uncorrelated or correlated variations

heights
Bin heights, equal to bin areas divided by the corresponding bin sizes

integral(*variations=None*, ***kwargs*)
Calculate the integral of the histogram.

Parameters

- **variations** (`list` of `str` or `str`) – if given, a tuple of the nominal integral and its upper and lower variation is calculated. This argument is passed to *histogram1d.net_variations()* and should be a list of considered variation names or the string `'all'`.
- ****kwargs** – additional keyword arguments that get passed to *histogram1d.net_variations()*

Returns the integral (as well as upper and lower variation if `variations` is given)

Return type `float`, or if `variations` are given, `tuple` of nominal as well as upper and lower variation

net_variations(*variations='all'*, *subtract_nominal=False*, *relative=False*)
Return upper and lower net areas variation of the histogram as a tuple.

@variations should be a sequence of considered variation names or the string 'all' @subtract_nominal: if True, return the differences with respect to the nominal areas @relative: if True, divide by the nominal areas

CAUTION: this method cannot yet deal with systematic uncertainties for which the up- and down-shift lie on the same side of the nominal. This is because the variations are fundamentally treated independently of each other, so there is no sense of the up- and down-shift being related to the same underlying uncertainty source.

set_heights(*heights*)
Set bin heights to an array of the same dimension as the current areas or to a scalar

to_file(*outfilename*, *key*, *recreate=False*)
Write histogram to text file. Multiple histograms with different keys can be written to the same file.

Parameters

- **outfilename** (`str`) – name of the file that the histogram is written to
- **key** (`bool`) – name/key of the histogram inside the output file
- **recreate** – option to recreate the output file rather than append to it

Convenience alias for one-dimensional histograms, since these are the most commonly encountered type in high-energy physics.

heppy.**histogram**
alias of `heppy.histogram.histogram1d`

1.3 Two-dimensional histogram

A histogram class with bins along two axes.

class heppy.**histogram2d**(**args*, ***kwargs*)
Heppy two-dimensional histogram. This currently has much more limited functionality than the 1D histogram class, although probably most (if not all) of the former's mathematical operations should also work for the 2D histogram (at least with minor modifications).

Note: only independent binnings of the two axes are supported (i.e. y-bins don't depend on x-bins and vice versa).

nbins

Returns tuple of number of bins along x- and y-axis

bin_index_x(*x*)
Returns the index of the x-axis bin that contains the given x-value.

Lower bin edges are included in a bin, upper bin edges are excluded (same as in the ROOT convention).

Parameters **x** (`float`) – x-value

Returns index of x-axis bin that contains the x-value

Return type `int`

Raises **ValueError** – if x-value lies outside of the outer bin edges of the histogram

bin_index_y(*y*)
Returns the index of the y-axis bin that contains the given y-value.

Lower bin edges are included in a bin, upper bin edges are excluded (same as in the ROOT convention).

Parameters **y** (`float`) – y-value

Returns index of y-axis bin that contains the y-value

Return type `int`

Raises **ValueError** – if y-value lies outside of the outer bin edges of the histogram

points()
Point representation of 2D histogram.

This involves flattening/ravelling the histogram bin midpoints and heights to one-dimensional arrays. The flattening is done in row-major, C-style order, with the y-axis index changing fastest and the x-axis index changing slowest.

Returns `tuple` of x-axis bin midpoints, y-axis bin midpoints, and heights

rebin(*newedges*)
Rebin 2D histogram. Correlated and uncorrelated variations will be recalculated to match the new bin edges.

CAUTION: currently ASSUMES that each uncorrelated variation only has shifts in one direction of the nominal (i.e. it is either higher or lower everywhere)!

Parameters **newedges** (`tuple` of two `numpy.array`) – new bin edges. Each new bin edge should correspond to an existing bin edge, i.e. only existing bins are merged

Raises `ValueError` if newedges is not of the correct type

as_1d(*name=''*)
Return a copied one-dimensional reinterpretation of this histogram. This only works if the histogram only has one bin in one of its dimensions. This dimension will then be ignored.

Parameters **name** (`str`) – name for the reinterpreted histogram

project(*axis*, *name=''*)
Project histogram to one axis by integrating over the other. Correlated and uncorrelated uncertainties are computed for the resulting one-dimensional histogram.

Parameters

- **axis** (`'x'` or `'y'`) – which axis to project onto, i.e. the *axis that is kept*

- **name** (str) – name for the projection histogram

Returns *heppy.histogram1d* representing the projection

Raises ValueError if invalid axis identifier is given

slice(*axis*, *bin_index*, *name=''*)
Returns 1D histogram of the distribution along one axis in a given bin of the other axis.

Parameters

- **axis** ('x' or 'y') – axis along which the slicing is done, i.e. the *axis that is kept*
- **bin_index** (int) – index of the bin on the *axis that is not kept*
- **name** (str) – name for the slice histogram

Returns 1D histogram of the slice

Return type *heppy.histogram1d*

height(*bin_index_x*, *bin_index_y*)
Returns the height of the given bin indices with uncertainties.

Parameters

- **bin_index_x** – bin index along x-axis
- **bin_index_x** – int
- **bin_index_y** – bin index along y-axis
- **bin_index_y** – int

Returns height of the indexed bin including its variations

Return type heppy.value

iterheights(*faster='y'*)
Generates iterator over heights.

Parameters **faster** (str; 'x' or 'y') – controls the iteration order by specifying along which axis the bin index changes faster

Returns bin heights including their variations

Return type heppy.value

iterbins()
Generates iterator over bins, yielding bin edges and heights.

Returns x-axis bin egdes, y-axis bin edges, and bin height with variations

Return type tuple of the following: tuple of two float, tuple of two float, and one heppy.value

Usage example:

```
>>> import heppy as hp
>>> import numpy as np
>>> heights = np.array([             # bin heights
        [1., 5.],
        [2., 6.],
        [3., 7.],
        ])
>>> x = np.array([-7., 0., 5., 50.]) # bin edges in x
```

(continues on next page)

(continued from previous page)

```
>>> y = np.array([-1., 0., 1.])       # bin edges in y
>>> h = hp.histogram2d((x, y), heights)
>>> for binedges_x, binedges_y, height in h.iterbins(): print(binedges_x,␣
↪binedges_y, height.nominal)
(-7.0, 0.0) (-1.0, 0.0) 1.0
(-7.0, 0.0) (0.0, 1.0) 5.0
(0.0, 5.0) (-1.0, 0.0) 2.0
(0.0, 5.0) (0.0, 1.0) 6.0
(5.0, 50.0) (-1.0, 0.0) 3.0
(5.0, 50.0) (0.0, 1.0) 7.0
```

to_root (*nominal_label='nominal'*, *key_form='{name}_{variation}'*, *errors=None*)
Returns a dictionary of variation names mapped to ROOT TH2D's.

Parameters

- **nominal_label** (str) – dictionary key to give to the nominal histogram
- **key_form** (str) – template for keys in the returned dictionary. The names of the histograms will also be set to these values. It accepts the formatting fields *name* (for *self.name*) and variation (for the name of the variation)
- **errors** – optional array of bin errors. Must have the same dimensionality

as the histogram areas. Note that the same errors are used for all variation histograms. :type errors: np.array or None

Returns dictionary of ROOT histograms (values) mapped by variation name (keys)

Return type dict of str and ROOT.TH2D

to_root_file (*filename*, *recreate=False*, ***kwargs*)
Writes this histogram to a ROOT file as a set of TH2D's.

Parameters

- **filename** (str) – name or path to the ROOT file. May exist or not.
- **recreate** (bool) – only has an effect if the ROOT file exists already. If *True*, it will be overwritten. Otherwise, the new histograms will be added to the existing file.
- ****kwargs** – keyword arguments that get passed on to *histogram2d.to_root()*

__add__ (*other*)
Add another histogram or a scalar to this histogram.

Returns the result of the addition as a histogram.

Correlated variations are treated as fully correlated among the two histograms if they have the same name, otherwise they are treated as uncorrelated. Uncorrelated variations are treated as uncorrelated between the two histograms.

__mul__ (*other*)
Multiply another histogram or a scalar binwise with this histogram.

Returns the result of the binwise multiplication as a histogram.

Correlated variations are treated as fully correlated among the two histograms if they have the same name, otherwise they are treated as uncorrelated. Uncorrelated variations are treated as uncorrelated between the two histograms.

__sub__ (*other*)
Subtract another histogram or a scalar from this histogram.

Returns the result of the subtraction as a histogram.

Correlated variations are treated as fully correlated among the two histograms if they have the same name, otherwise they are treated as uncorrelated. Uncorrelated variations are treated as uncorrelated between the two histograms.

__truediv__ (*other*)
Divide by another histogram or a scalar binwise.

Returns the result of the binwise division as a histogram.

Correlated variations are treated as fully correlated among the two histograms if they have the same name, otherwise they are treated as uncorrelated.

CAUTION: Uncorrelated variations are treated as uncorrelated between the two histograms. If the uncorrelated variations represent statistical uncertainties, this means that the division treats the two histograms as statistically uncorrelated.

See also *histdiv()*

apply (*function*, *new_binedges=None*)
Same as *basehistogram.apply_inplace()*, except that the resulting histogram is returned (as an independent object) with the function applied, while the original histogram is not modified.

Returns histogram with function applied and possibly new bin edges (*histogram1d*, *histogram2d*, or *basehistogram*)

apply_inplace (*function*, *new_binedges=None*)
Call a function on the nominal areas as well as all varied areas (in `corr_variations` and `uncorr_variations`), modifying the existing histogram.

It is possible to convert the histogram to a different type (e.g. *histogram2d* → *histogram1d*) by giving new binedges of the desired new dimensionality. If the new binedges have a dimension other than 1D or 2D, the type will become *basehistogram*.

For a version of this method that leaves the original histogram unchanged and returns a copy with the function applied (and optionally new bin edges), see *basehistogram.apply()*. I HIGHLY recommend using that if the new histogram will be of different type to avoid confusion!

Caution: if you change the binning, it is your responsibility that uncertainties encoded in the variations are handled correctly.

Parameters

- **function** (`function`) – function taking a `numpy.array` as argument
- **new_binedges** – optional argument to set new binedges. If the binedges change the dimension of the histogram (e.g. from 2D to 1D), the histograms' type is changed accordingly

Example 1: taking the sine of the areas

```
import heppy as hp
import numpy as np
foo = hp.histogram1d(np.array([0., 1., 2., 3.]), [5., 6., 7])
foo.apply_inplace(np.sin)
```

Example 2: projecting a 2D histogram to its x-axis (integrating over the y-axis)

```
import heppy as hp
import numpy as np
binedges = (np.array([0., 10., 20.]), np.array([10., 20., 30.]))
```

(continues on next page)

(continued from previous page)

```
areas = [[0.1, 0.2], [0.3, 0.4]]
foo = hp.histogram2d(binedges, areas, areas=True)
from functools import partial
project_x = partial(np.sum, axis=1)
foo.apply_inplace(project_x, new_binedges=foo.binedges[0])
```

Raises `ValueError` if the shape of the areas after the function is called on them does not match the shape of the bin edges (after setting them to `new_binedges` if given)

binsizes

Bin sizes.

For a one-dimensional histogram, returns an array of dimension (N, 1), where N is the number of bins. The elements represent the width of each bin.

For a two-dimensional histogram, returns an array of dimension (N, M), where N is the number of bins along the first axis ("x-axis") and M is the number of bins along the second axis ("y-axis"). The elements represent the area of each bin.

clip(*minimum=None*, *maximum=None*)

Clips the bin areas at a minimum and/or maximum value.

Similar to `numpy.clip` (see here).

Both the nominals and all correlated and uncorrelated variations of the histogram are clipped. *Note that the net variation of multiple individual variations considered together may still lie outside of the clip range!*

Parameters **minimum** – minimum accepted value; areas below this value will be set to the value.

If `None`, no minimum is imposed. :type minimum: `float` or `None` :param maximum: maximum accepted value; areas above this value will be set to the value. If `None`, no maximum is imposed. :type maximum: `float` or `None`

The behaviour is like that of `numpy.clip`, e.g. with regards to what happens if `minimum` is greater than `maximum`.

"Undocumented" feature: you can actually pass a Numpy `array_like` as `minimum` or `maximum`, not just a single `float`!

errorbars(*variations='all'*)

Returns upper and lower error bars, defined as the absolute net variations (taking into account the given variations) with the nominal values subtracted.

extract_variation_histogram(*variation*, ***kwargs*)

Get a new histogram object that has a given variation as nominal. Useful e.g. for studying a particular systematic variation.

Parameters

- **variation** (`str`) – name of the variation
- ****kwargs** – get passed on to constructor of new histogram, e.g. useful to set a `name` for the new histogram.

Returns new *heppy.histogram* that has a given `variation` as nominal

Raises

- **KeyError** – if variation not found in either uncorrelated or correlated variations
- **RuntimeError** – if variation found in both uncorrelated or correlated variations

heights
Bin heights, equal to bin areas divided by the corresponding bin sizes

integral(*variations=None*, ***kwargs*)
Calculate the integral of the histogram.

Parameters

- **variations** (`list` of `str` or `str`) – if given, a tuple of the nominal integral and its upper and lower variation is calculated. This argument is passed to *histogram1d.net_variations()* and should be a list of considered variation names or the string `'all'`.
- ****kwargs** – additional keyword arguments that get passed to *histogram1d.net_variations()*

Returns the integral (as well as upper and lower variation if `variations` is given)

Return type `float`, or if `variations` are given, `tuple` of nominal as well as upper and lower variation

net_variations(*variations='all'*, *subtract_nominal=False*, *relative=False*)
Return upper and lower net areas variation of the histogram as a tuple.

@variations should be a sequence of considered variation names or the string 'all' @subtract_nominal: if True, return the differences with respect to the nominal areas @relative: if True, divide by the nominal areas

CAUTION: this method cannot yet deal with systematic uncertainties for which the up- and down-shift lie on the same side of the nominal. This is because the variations are fundamentally treated independently of each other, so there is no sense of the up- and down-shift being related to the same underlying uncertainty source.

set_heights(*heights*)
Set bin heights to an array of the same dimension as the current areas or to a scalar

to_file(*outfilename*, *key*, *recreate=False*)
Write histogram to text file. Multiple histograms with different keys can be written to the same file.

Parameters

- **outfilename** (`str`) – name of the file that the histogram is written to
- **key** (`bool`) – name/key of the histogram inside the output file
- **recreate** – option to recreate the output file rather than append to it

1.4 Free functions

Free functions related to histograms.

heppy.**histdiv**(*a*, *b*, *corr=None*, *ignore_denominator_uncertainty=False*)
Sophisticated division of two histograms

Parameters

- **a** (*heppy.basehistogram*) – numerator histogram
- **b** (*heppy.basehistogram*) – denominator histogram
- **corr** – information on how a and b are correlated — NOT YET IMPLEMENTED, do not use

- **ignore_denominator_uncertainty** (bool) – switch to ignore the variations of the denominator histogram. If True, divide all variations of the numerator histogram by the nominal denominator histogram.

NOTE: the returned ratio histogram's bin heights are not given "per bin size", but take the role that the areas have for histograms that do not represent a ratio.

Returns ratio histogram a/b with variations treated as specified

Raises **NotImplementedError** – if corr is not None (remains to be implemented)

heppy.**from_file**(*infilename*, *key*)

Read histogram written out by heppy (using *heppy.basehistogram.to_file*).

Parameters

- **infilename** (str) – name of the file that the histogram should be read from
- **key** (str) – name/key of the histogram inside the input file

Returns *heppy.histogram1d* or *heppy.histogram2d*

heppy.**zeros_like**(*a*, *name=None*)

Returns a histogram of zeros with the same bin edges and variations as a.

The attributes and plot attributes are not kept.

Parameters **a** – the bin edges and variation names of a define these same

attributes of the returned histogram. :type a: *heppy.histogram1d* or *heppy.histogram2d* :param name: if given, this becomes the name attribute of the returned histogram. If Non, the returned histogram has the name "zeros like " + a.name :type name: str or None

Returns *heppy.histogram1d* or *heppy.histogram2d*

CHAPTER 2

Reading and writing in different formats

2.1 Heppy

Heppy histograms can be written to and read from the native text-based format using *heppy.basehistogram.to_file()* and *heppy.from_file()*.

2.2 ROOT

Reading from a ROOT file:

heppy.**readroot**(*rootfile*, *histpath*, *variation_paths={}*, *ignore_missing_variations=False*, ***kwargs*)

Reads a histogram (possibly with systematic variations) from a ROOT file.

Parameters

- **rootfile** (str) – path to ROOT file
- **histopath** (str) – path of the histogram inside the file
- **variation_paths** (dict of str : str) – optional dictionary of variation names (keys) and paths to the variation histograms inside the same ROOT file (values). NOTE: we can easily add the ability to read variation histograms also from other ROOT files than the nominal, get in touch if you want that.
- ****kwargs** – get passed on to histogram constructor. An important one is areas=False if retrieving ratios (e.g. efficiencies).

Converting to ROOT histograms and writing to ROOT files can be done using *heppy.histogram1d.to_root()* and *heppy.histogram1d.to_root_file()* (for 1D histograms) and *heppy.histogram2d.to_root()* and *heppy.histogram2d.to_root_file()* (for 2D histograms).

2.3 Rivet (YODA)

One-dimensional histograms can be written to the Rivet YODA format using *heppy.histogram1d.to_yoda()* or its synonym *heppy.histogram1d.to_rivet()*.

2.4 TRExFitter (YAML)

Reading TRExFitter YAML format:

heppy.**readtrex**(*trexfile*, *unfolded_format=False*)

Reads Heppy histogram from TRExFitter YAML format.

Requires PyYAML.

IMPORTANT: the uncertainties are treated as correlated uncertainties, since their actual correlations cannot be inferred from the histogram. So treat them with care: they're fine for plotting and looking at individual bins, but do not rebin the histogram or do statistical testing involving its shape and assume that you will find correct results.

Parameters

- **trexfile** (str) – path to input file.
- **unfolded_format** (bool) – set this to True if reading the YAML format that TRExFitter uses for unfolded cross sections.

2.5 ATLAS Isolation and Fake Forum (XML)

Values with uncertainties can be converted to an XML string that the ATLAS Isolation and Fake Forum fake background tool understands in its configuration file using :py:func‘heppy.Value.to_atlasiff‘. You can use this for writing out e.g. fake factors.

CHAPTER 3

Histogram stack

A collection of histograms. The total summed histogram and its variations may be accessed easily, and the stack may be plotted conveniently.

class heppy.**histostack**(*histograms*, *attributes={}*)

Stack of one-dimensional histograms

Parameters

- **histograms** (list of *heppy.histogram1d*) – histograms in the stack
- **attributes** (dict) – dictionary of completely arbitrary attributes that the user can provide/change/access. E.g. information on how to plot

total

Returns *heppy.histogram1d* that is the combination of all the stacked ones, with the combined uncertainty. If the stack has no histograms, returns None

iterbands()

Bands are 3-tuples of (curve representation) x-values as well as two subsequent curves that are useful as arguments to plt.fill_between(). E.g.:

```
for histogram, band in zip(stack.histograms, stack.iterbands()):
    ax.fill_between(*band, **histogram.attributes)
```

The bands are ordered such that the first histogram in the stack is at the top and the last at the bottom

CHAPTER 4

Value with uncertainties

class heppy.**Value**(*nominal*, *uncorr_variations={}*, *corr_variations={}*, *attributes={}*)

A single value with uncertainties.

Parameters

- **nominal** (float) – nominal value
- **uncorr_variations** (dict of str and float) – dictionary of variations that are uncorrelated between different heppy.value objects even when they have the same key
- **corr_variations** (dict of str and float) – dictionary of variations that are fully correlated between different heppy.value objects when they have the same key, and uncorrelated otherwise
- **attributes** (dict) – dictionary of completely arbitrary attributes that the user can provide/change/access. E.g. information about the data sample that produced the value

to_atlasiff(*attributes={}*, *up_suffix='__1up'*, *down_suffix='__1down'*)

Returns string representation in ATLAS IFF format.

This is the XML format used by the fake-lepton background tool of the ATLAS Isolation and Fakes Forum.

Parameters

- **attributes** (str) – dictionary of attributes to put in the bin-tag
- **up_suffix** – suffix in variation keys to designate an up variation
- **down_suffix** – suffix in variation keys to designate an down variation

Usage example:

```
>>> import heppy as hp
>>> nominal = 12.3
>>> uncorr_variations = {
        'stat__1up' : 12.4,
        'stat__1down' : 12.1,
        }
```

(continues on next page)

(continued from previous page)

```
>>> corr_variations={
        'efficiency__1up' : 13.1,
        'efficiency__1down' : 9.8,
        'energy_scale__1up' : 10.5,
        }
>>> v = hp.Value(nominal, uncorr_variations=uncorr_variations, corr_
↪variations=corr_variations)
>>> v.to_atlasiff(attributes={'pt' : '[20,inf]', '|eta|' : '[0.0,0.6]'})
'<bin pt="[20,inf]" |eta|="[0.0,0.6]"> 12.3 +0.1-0.2 (stat) -1.8+0.0 (energy_
↪scale) +0.8-2.5 (efficiency) </bin>'
```

net_variations(*variations='all'*, *subtract_nominal=False*, *relative=False*)

Return upper and lower net height variation of the value.

@variations should be a sequence of considered variation names or the string 'all' @subtract_nominal: if True, return the differences with respect to the nominal heights @relative: if True, divide by the nominal heights

CAUTION: this method cannot yet deal with systematic uncertainties for which the up- and down-shift lie on the same side of the nominal. This is because the variations are fundamentally treated independently of each other, so there is no sense of the up- and down-shift being related to the same underlying uncertainty source.

CHAPTER 5

Plotting

5.1 2D plotting (histograms as lines, points, or bands)

2D plotting, i.e. plotting of 1D histograms. A single histogram stack as well as any number of individual histograms visualised as curves, points, bands can be shown. Any number of panels (called "pads" in ROOT) can be included in a single figure.

The function for making a 2D plot is `make_figure`.

A special convenience function for plotting a breakdown of the uncertainties of a single histogram called `make_uncertainty_breakdown` is also provided.

`heppy.`**`make_figure`**(*panels*, *title=''*, *figsize=(8, 5)*, *write=''*, *xlims=None*, *xmax=None*, *legend_outside=False*)

Parameters

- **`panels`** (*`heppy.panel`* or `list` of *`heppy.panel`*) – panel(s) to visualise in the plot
- **`title`** (*`str`*) – plot title
- **`write`** (*`str`*) – may be changed to a filename, which will result in the figure being rendered and saved at the given location
- **`legend_outside`** (*`bool`*) – option to move the legend next to the plot panels. It also changes the legend style to try to make it look better next to the plot: no box, text wrapped at 30 characters (not tested with LaTeX rendering — proceed with caution), smaller text (fontsize='small')
- **`xlims`** (*tuple* of *float*, or *None*) – can be used to manually set lower and upper x-axis limits, e.g. xlims=(0.0, 2.0)

Returns tuple of the created plt.figure object and plt.axes objects. These can be assigned to variables by the user to allow further manipulations of the plot (style, contents, etc.)

class heppy.**panel**(*title=''*, *height=1.0*, *xlabel=''*, *ylabel=''*, *logx=False*, *logy=False*, *stack=None*, *curves=[]*, *bands=[]*, *points=[]*, *pointshift=0.0*, *scatters=[]*, *ylims=None*, *nolegend=False*, *legend_title=None*, *legend_loc=None*, *unbinned=[]*)

A panel holds histograms and other information that describe a panel of a plot

Parameters

- **title** (*str*) – title of the panel
- **height** (*float*) – relative height of the panel with respect to any other panels in a plot
- **xlabel** (*str*) – x-axis label
- **ylabel** (*str*) – y-axis label
- **logx** (*bool*) – plot x-axis on a logarithmic scale?
- **logy** (*bool*) – plot y-axis on a logarithmic scale?
- **stack** (*heppy.histostack*, or *None*) – histogram stack to plot
- **curves** (list of *heppy.histogram1d*) – histograms to plot as curves
- **bands** (list of *heppy.histogram1d*) – histograms to plot as (uncertainty) bands
- **points** (list of *heppy.histogram1d*) – histograms to plot as points located at the centre of each of their bins
- **pointshift** (*float*) – distance to shift points by horizontally to avoid overlap and improve readability. This functionality is poorly tested and may be broken
- **scatters** (list of *heppy.histogram1d*) – this is a bit of an oddball. You can use it to plot 2D scatters that aren't really histograms. The x-values are lower bin edges (the uppermost binedge is not used for anything) while the y-values are the areas.
- **ylims** (*tuple* of *float*, or *None*) – can be used to manually set lower and upper y-axis limits, e.g. ylims=(0.0, 2.0)
- **legend_title** (*str*) – legend title
- **legend_loc** (whatever Matplotlib accepts for the `loc` keyword arg of legend) – legend title
- **unbinned** (*list of tuples, each tuple has two np.arrays and a plot attribute dict*) – unbinned curves, given as tuple of x and y values (for matplotlib.pyplot.plot)

heppy.**make_uncertainty_breakdown**(*histogram*, *separator='__'*, *ylims=None*, *xlabel=''*, ***kwargs*)

@histogram: heppy.histogram object for which the uncertainty breakdown figure will be made @separator: string that separates high/low (up/down, ...) indictator from the rest of the uncertainty name, e.g. "jet_energy_scale__1up" and "jet_energy_scale__1down" uses the separator "__" @ylims: may be set to a tuple/list of lower and upper y-axis limits, e.g. ylims=(0.0, 2.0) @**kwargs: get passed on to make_figure()

5.2 3D plotting (histogram as heatmap)

3D plotting, i.e. plotting of 2D histograms. Only a single histogram can be put into each plot. The histogram contents can be printed onto the histogram in a nicely formatted way for better readability.

heppy.**make_heatmap**(*histogram*, *areas=False*, *title=''*, *figsize=(8, 5)*, *colorbar=True*, *text_format=<function TextFormatter.brief>*, *text_precision=3*, *text_autocolor=True*, *black_text_below=None*, *monowidth=False*, *write=''*, *xmax=None*, ***kwargs*)

Make a heatmap plot of a two-dimensional histogram

Parameters

- **histogram** (*heppy.histogram2d*) – two-dimensional histogram to visualise
- **title** (str) – plot title
- **figsize** (tuple of float) – figure size
- **colorbar** (bool) – show color bar or not
- **text_format** (str, function or None) – string template or function returning the text to be printed inside each bin. The following format keys (if string) or keyword arguments (if function) will be provided: {nominal}, {uncert_up} and {uncert_down} for total uncertainty, {stat_up} and {stat_down} for statistical uncertainty, {syst_up} and {syst_down} for systematic uncertainty. All uncertainties are given as non-negative numbers. The class *heppy.TextFormatter* provides a set of useful and somewhat adaptable predefined formatter functions.
- **monowidth** (bool) – if True, all bins are shown as equally wide/high, with the bin edges written in the label.
- **write** (str) – may be changed to a filename, which will result in the figure being rendered and saved to disk
- ****kwargs** – keyword arguments that get passed on to plt.hist2d

The following keys in histogram.plot_attributes can be used to set axis labels:

- "xlabel" – x-axis label
- "ylabel" – y-axis label

class heppy.**TextFormatter**

Predefined functions to format bin content text printed on heatmap. The user can alternatively write their own such functions.

Contents will be printed with a precision of (up to) three significant digits. If you want to set a different precision, you can create your own adapted formatting function as the following example illustrates:

```
import functools
# set the number of significant digits (e.g. to 4)
formatter = functools.partial(TextFormatter.nominal, significants=4)
# or set the fixed absolute precision (e.g. to 3 digits after the decimal point)
formatter = functools.partial(TextFormatter.nominal, fixedprec=3)
# then use as: heppy.make_heatmap(..., text_format=formatter, ...)
```

Here significants represents the maximum number of significant digits considered (default: 3), while fixedprec represents the fixed absolute precision considered, e.g. 1 for a precision of 0.1 or -1 for a precision of 10 (default: None). If fixedprec is given, significants is ignored.

Hint: when writing a custom formatting function, any unnecessary keyword arguments can be absorbed into a **kwargs catch-all parameter to keep the function signature shorter and tidier.

static nominal(*significants=3*, *fixedprec=None*, *nominal=None*, ***ignore*)

Returns LaTeX string of nominal value.

static brief(*significants=3*, *fixedprec=None*, *nominal=None*, *uncert_up=None*, *uncert_down=None*, ***ignore*)

Returns LaTeX string of nominal value and total uncertainty.

The format is $\mathrm{nominal} \pm \sigma$, where σ is the uncertainty from the uncorrelated variations and the correlated variations. Asymmetric uncertainties are supported and will be shown as $\overset{\sigma^{\mathrm{up}}}{\sigma^{\mathrm{down}}}$.

static statsyst(*significants=3*, *fixedprec=None*, *nominal=None*, *stat_up=None*, *stat_down=None*, *syst_up=None*, *syst_down=None*, ***ignore*)

Returns LaTeX string of nominal value and statistical and systematic uncertainty.

The format is $\mathrm{nominal} \pm \sigma_{\mathrm{stat}} \pm \sigma_{\mathrm{syst}}$, where σ_{stat} is the uncertainty from the uncorrelated variations and σ_{syst} is the uncertainty from the correlated variations. Asymmetric uncertainties are supported and will be shown as $\overset{\sigma^{\mathrm{up}}_{\mathrm{syst}}}{\sigma^{\mathrm{down}}_{\mathrm{syst}}}$ etc.

CHAPTER 6

Systematic-uncertainty tools

Heppy provides tools to explore and treat the systematic variations of histograms. This notably means combining multiple systematic variations into a single combined systematic. Some examples include:

- Combining the members of a PDF set into the PDF uncertainty
- Adding several uncorrelated uncertainties in quadrature to get a resulting net uncertainty
- Combining a set of bootstrap replicas into a final uncertainty
- Taking the envelope of perturbative QCD scale variations to get an estimate of the fixed-order calculation uncertainty
- etc.

The basic usage is:

1. Create any number of `heppy.uncertainty.model` instances that specify *which* variations should be combined and *how*.
2. Call `heppy.uncertainty.combine_copy`, taking as arguments a histogram and a list of uncertainty models, to return a copy of the histogram where the combination models have been applied to the variations.

`heppy.uncertainty.`**`combine_add_quad`**(*array*, *nominal*)

`heppy.uncertainty.`**`combine_add_lin`**(*array*, *nominal*)

`heppy.uncertainty.`**`combine_symm_rms`**(*array*, *nominal*)

`heppy.uncertainty.`**`combine_asym_rms`**(*array*, *nominal*)

`heppy.uncertainty.`**`combine_envelope`**(*array*, *nominal*)

`heppy.uncertainty.`**`combine_asym_hessian`**(*array*, *nominal*)

`heppy.uncertainty.`**`combine_symm_hessian`**(*array*, *nominal*)

`heppy.uncertainty.`**`combine_asym_hessian_pairwise`**(*array*, *nominal*)

`heppy.uncertainty.`**`combine_symm_hessian_pairwise`**(*array*, *nominal*)

class heppy.uncertainty.**model**(*name*, *keys*, *strategy*, *reference=None*, *postprocess=None*, *suffixes=('__hi', '__lo')*, *controlplot=None*, *matches_required=None*)

Model for combining multiple variations into one uncertainty.

Contains information of which variations to combine how and what to call the result.

apply(*histogram*, *controlplot_location=None*)

WARNING: SIDE EFFECTS - this method will change the @histogram.corr_variations dictionary. @controlplots: if a directory (end with '/') or prefix is given, control plots will be stored there for models that have them enabled (model.controlplot != None)

heppy.uncertainty.**remove_same_sign_shifts**(*histogram, suffixes=['_1up', '_1down', '_up', '_down']*)

Drop smaller same-sign correlated variation shifts from the nominal for any group of variations whose names differ only by (any number of occurrences of) any of the strings in `matches`

heppy.uncertainty.**combine_copy**(*histogram, models, ignore_missing=False, controlplot_location=None, drop_same_sign_shifts=False, suffixes=['_1up', '_1down', '_up', '_down']*)

@histogram: the input histogram. The return value will be a copy of this histogram, with the desired variation combinations applied. @models: iterable of the models to be applied @ignore_missing: if True, do not throw an exception if the input variations specified in a model are missing, but simply ignore the model @controlplot_location: if a directory (end with '/') or prefix is given, control plots will be stored there for models that have them enabled (model.controlplot != None) :param drop_same_sign_shifts: if True, _correlated_ variation names that differ only by any suffix given in argument `suffixes` are grouped together. If more than one of these grouped variations has a shift with respect to the nominal in a given bin, only the largest shift in that bin is kept. The other shifts are set to zero (i.e. the variation is set to equal the nominal in the bin). A common case where this is useful is to avoid double-counting the same source of systematic uncertainty in bins where the "up" and "down" variation point in the same direction with respect to the nominal in some bin(s). Uncorrelated variations are not affected by this option. :type drop_same_sign_shifts: `bool` :param suffixes: see argument `drop_same_sign_shifts` for explanation :type suffixes: `list` of `str`

IMPORTANT NOTE: it is possible to apply combination models whose input variations are only produced in the same ca

the variations don't yet need to exist in the @histogram.corr_variations when passing @histogram to combine_copy.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

Symbols

A

B

C

E

F

H

I

M

N

P

R

S

T

V

Z