# xhistogram Documentation

***Release 0.2.0+82.ge01291b.dirty***

**xhistogram developers**

**Jun 28, 2021**

# CONTENTS

Histograms (a.k.a "binning") are much more than just a visualization tool. They are the foundation of a wide range of scientific analyses including [joint] probability distributions and coordinate transformations. Xhistogram makes it easier to calculate flexible, complex histograms with multi-dimensional data. It integrates (optionally) with Dask, in order to scale up to very large datasets and with Xarray, in order to consume and produce labelled, annotated data structures. It is useful for a wide range of scientific tasks.

CHAPTER

ONE

# WHY A NEW HISTOGRAM PACKAGE?

The main problem with the standard `histogram` function in numpy and dask is that they automatically act over the entire input array (i.e. they "flatten" the data). Xhistogram allows you to choose which axes / dimensions you want to preserve and which you want to flatten. It also allows you to combine N arbitrary inputs to produce N-dimensional histograms. A good place to start is the *Xhistogram Tutorial*.

CHAPTER

TWO

# CONTENTS

## 2.1 Installation

### 2.1.1 Requirements

xhistogram is compatible with python 3. It requires numpy and, optionally, xarray.

### 2.1.2 Installation from Conda Forge

The easiest way to install xhistogram along with its dependencies is via conda forge:

```
conda install -c conda-forge xhistogram
```

### 2.1.3 Installation from Pip

An alternative is to use pip:

```
pip install xhistogram
```

This will install the latest release from pypi.

### 2.1.4 Installation from GitHub

xhistogram is under active development. To obtain the latest development version, you may clone the source repository and install it:

```
git clone https://github.com/xgcm/xhistogram.git
cd xhistogram
python setup.py install
```

or simply:

```
pip install git+https://github.com/xgcm/xhistogram.git
```

Users are encouraged to fork xhistogram and submit issues and pull requests.

## 2.2 Xhistogram Tutorial

Histograms are the foundation of many forms of data analysis. The goal of xhistogram is to make it easy to calculate weighted histograms in multiple dimensions over n-dimensional arrays, with control over the axes. Xhistogram builds on top of xarray, for automatic coordiantes and labels, and dask, for parallel scalability.

### 2.2.1 Toy Data

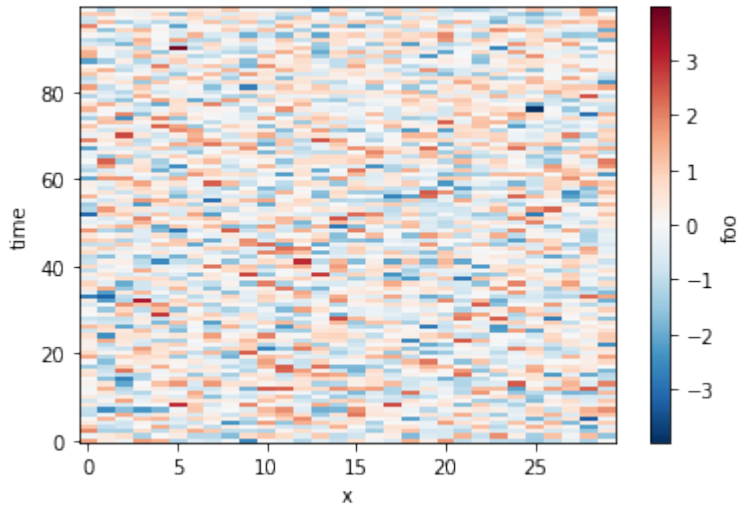
We start by showing an example with toy data. First we use xarray to create some random, normally distributed data.

#### 1D Histogram

[1]:
```
import xarray as xr
import numpy as np
%matplotlib inline

nt, nx = 100, 30
da = xr.DataArray(np.random.randn(nt, nx), dims=['time', 'x'],
                  name='foo') # all inputs need a name
display(da)
da.plot()
```

```
<xarray.DataArray 'foo' (time: 100, x: 30)>
array([[ 1.5593684 ,  0.40145489, -0.22479739, ..., -0.3437886 ,
        -1.23356416,  0.03077635],
       [-0.2228109 ,  0.38783241,  0.68898411, ..., -0.16182289,
         1.23321392, -1.59096389],
       [ 1.92575725, -1.25776679, -1.30329336, ..., -0.51350828,
        -1.49556751,  0.51878971],
       ...,
       [-0.77886827,  0.10927865, -0.02355692, ..., -0.0774592 ,
         0.67645978,  0.58709225],
       [ 1.01767561,  1.07267062,  0.13102546, ..., -0.42234066,
         1.48014365, -0.0097595 ],
       [-0.05547442, -1.26186425,  0.10945666, ...,  0.37304738,
         1.46653078, -1.2716321 ]])
Dimensions without coordinates: time, x
```

[1]:
```
<matplotlib.collections.QuadMesh at 0x7f1a8fc81a00>
```

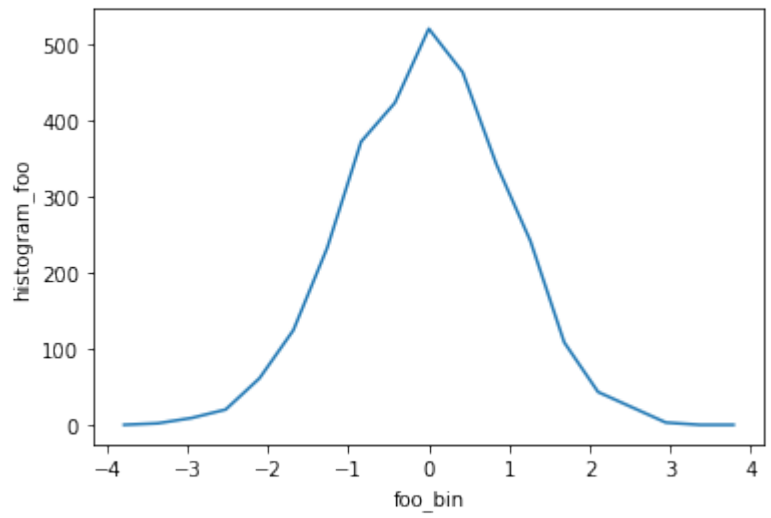



By default xhistogram operates on all dimensions of an array, just like numpy. However, it operates on xarray DataArrays, taking labels into account.

[2]:
```
from xhistogram.xarray import histogram

bins = np.linspace(-4, 4, 20)
h = histogram(da, bins=[bins])
display(h)
h.plot()
```

```
<xarray.DataArray 'histogram_foo' (foo_bin: 19)>
array([  1,   3,  10,  21,  62, 125, 233, 372, 423, 520, 463, 342, 242,
       109,  44,  24,   4,   1,   1])
Coordinates:
  * foo_bin  (foo_bin) float64 -3.789 -3.368 -2.947 -2.526 ... 2.947 3.368 3.789
```

[2]:
```
[<matplotlib.lines.Line2D at 0x7f1a8fb525e0>]
```




**TODO:** - Bins needs to be a list; this is annoying, would be good to accept single items - The `foo_bin` coordinate is the estimated bin center, not the bounds. We need to add the bounds to the coordinates, but we can as long as we are

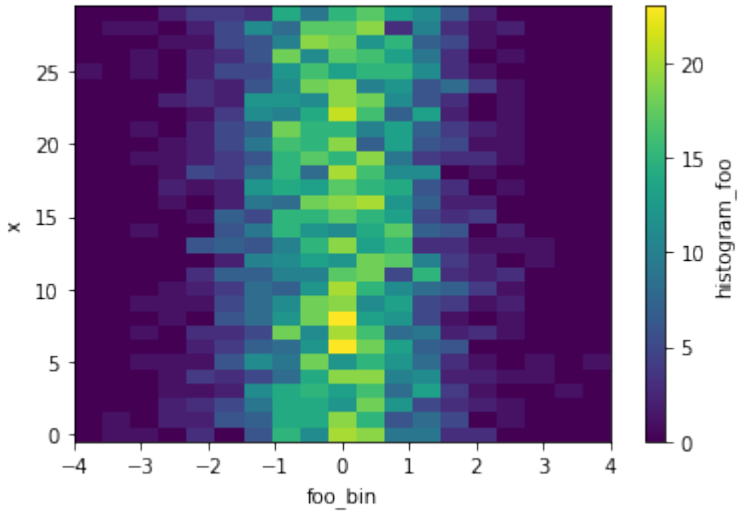
returning DataArray and not Dataset.

Both of the above need GitHub Issues

### Histogram over a single axis

[3]:
```
h_x = histogram(da, bins=[bins], dim=['time'])
h_x.plot()
```

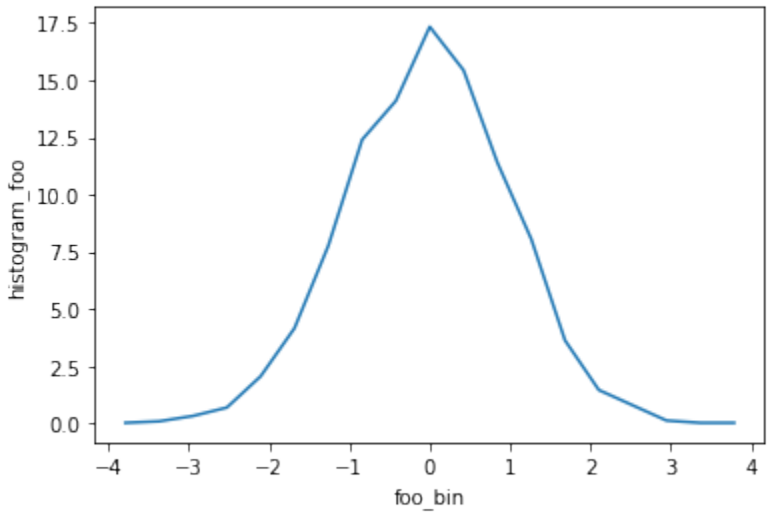
[3]:
```
<matplotlib.collections.QuadMesh at 0x7f1a8faf4610>
```




**TODO:** - Relax / explain requirement that dims is always a list

[4]:
```
h_x.mean(dim='x').plot()
```

[4]:
```
[<matplotlib.lines.Line2D at 0x7f1a8fa28b50>]
```



#### Weighted Histogram

Weights can be the same shape as the input:

[5]:
```
weights = 0.4 * xr.ones_like(da)
histogram(da, bins=[bins], weights=weights)
```

[5]:
```
<xarray.DataArray 'histogram_foo' (foo_bin: 19)>
array([  0.4,   1.2,   4. ,   8.4,  24.8,  50. ,  93.2, 148.8, 169.2,
       208. , 185.2, 136.8,  96.8,  43.6,  17.6,   9.6,   1.6,   0.4,
         0.4])
Coordinates:
  * foo_bin  (foo_bin) float64 -3.789 -3.368 -2.947 -2.526 ... 2.947 3.368 3.789
```

Or can use Xarray broadcasting:

[6]:
```
weights = 0.2 * xr.ones_like(da.x)
histogram(da, bins=[bins], weights=weights)
```

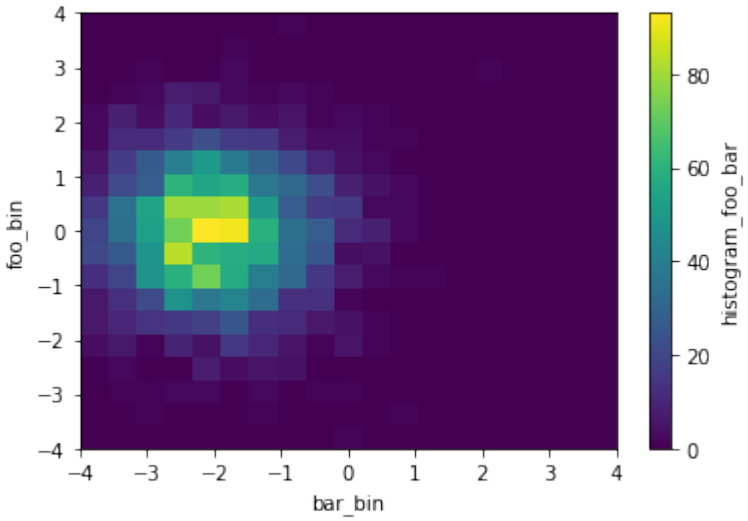
[6]:
```
<xarray.DataArray 'histogram_foo' (foo_bin: 19)>
array([  0.2,   0.6,   2. ,   4.2,  12.4,  25. ,  46.6,  74.4,  84.6,
       104. ,  92.6,  68.4,  48.4,  21.8,   8.8,   4.8,   0.8,   0.2,
         0.2])
Coordinates:
  * foo_bin  (foo_bin) float64 -3.789 -3.368 -2.947 -2.526 ... 2.947 3.368 3.789
```

### 2.2.2 2D Histogram

Now let's say we have multiple input arrays. We can calculate their joint distribution:

[7]:
```
db = xr.DataArray(np.random.randn(nt, nx), dims=['time', 'x'],
                  name='bar') - 2

histogram(da, db, bins=[bins, bins]).plot()
```

[7]:
```
<matplotlib.collections.QuadMesh at 0x7f1a8f9a7b20>
```



### 2.2.3 Real Data Example

#### Ocean Volume Census in TS Space

Here we show how to use xhistogram to do a volume census of the ocean in Temperature-Salinity Space

First we open the World Ocean Atlas dataset from the opendap dataset (http://apdrc.soest.hawaii.edu/dods/public_data/WOA/WOA13/1_deg/annual).

Here we read the annual mean Temparature, Salinity and Oxygen on a 5 degree grid.

[8]:
```
# Read WOA using opendap
Temp_url = 'http://apdrc.soest.hawaii.edu:80/dods/public_data/WOA/WOA13/5_deg/annual/temp
↪'
Salt_url = 'http://apdrc.soest.hawaii.edu:80/dods/public_data/WOA/WOA13/5_deg/annual/salt
↪'
Oxy_url = 'http://apdrc.soest.hawaii.edu:80/dods/public_data/WOA/WOA13/5_deg/annual/doxy'

ds = xr.merge([
    xr.open_dataset(Temp_url).tmn.load(),
    xr.open_dataset(Salt_url).smn.load(),
    xr.open_dataset(Oxy_url).omn.load()])
ds
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xhistogram/conda/latest/lib/python3.9/
↪site-packages/xarray/coding/times.py:119: SerializationWarning: Ambiguous reference␣
↪date string: 1-1-1 00:00:0.0. The first value is assumed to be the year hence will be␣
↪padded with zeros to remove the ambiguity (the padded reference date string is: 0001-1-
↪1 00:00:0.0). To remove this message, remove the ambiguity by padding your reference␣
↪date strings with zeros.
  warnings.warn(warning_msg, SerializationWarning)
/home/docs/checkouts/readthedocs.org/user_builds/xhistogram/conda/latest/lib/python3.9/
↪site-packages/xarray/coding/times.py:527: SerializationWarning: Unable to decode time␣
↪axis into full numpy.datetime64 objects, continuing using cftime.datetime objects␣
↪instead, reason: dates out of range
  dtype = _decode_cf_datetime_dtype(data, units, calendar, self.use_cftime)
/home/docs/checkouts/readthedocs.org/user_builds/xhistogram/conda/latest/lib/python3.9/
↪site-packages/numpy/core/_asarray.py:102: SerializationWarning: Unable to decode time␣
↪axis into full numpy.datetime64 objects, continuing using cftime.datetime objects␣
↪instead, reason: dates out of range
  return array(a, dtype, copy=False, order=order)
```

[8]:
```
<xarray.Dataset>
Dimensions:  (lat: 36, lev: 102, lon: 72, time: 1)
Coordinates:
  * time     (time) object -001-01-15 00:00:00
  * lev      (lev) float64 0.0 5.0 10.0 15.0 ... 5.2e+03 5.3e+03 5.4e+03 5.5e+03
  * lat      (lat) float64 -87.5 -82.5 -77.5 -72.5 -67.5 ... 72.5 77.5 82.5 87.5
  * lon      (lon) float64 -177.5 -172.5 -167.5 -162.5 ... 167.5 172.5 177.5
Data variables:
    tmn      (time, lev, lat, lon) float32 nan nan nan nan ... nan nan nan nan
    smn      (time, lev, lat, lon) float32 nan nan nan nan ... nan nan nan nan
    omn      (time, lev, lat, lon) float32 nan nan nan nan ... nan nan nan nan
Attributes:
    long_name:  statistical mean sea water temperature [degc]
```

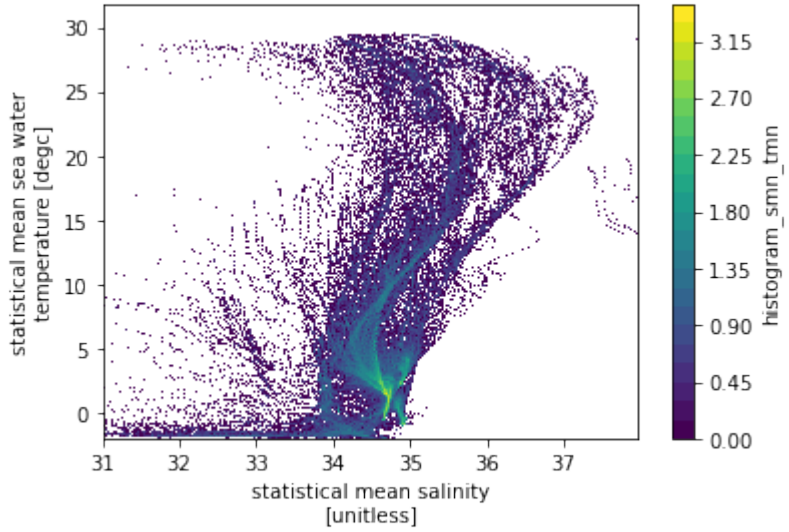
Use histogram to bin data points. Use canonical ocean T/S ranges, and bin size of $0.1^0C$, and $0.025psu$. Similar ranges and bin size as this review paper on Mode Waters: https://doi.org/10.1016/B978-0-12-391851-2.00009-X .

[9]:
```
sbins = np.arange(31,38, 0.025)
tbins = np.arange(-2, 32, 0.1)
```

[10]:
```
# histogram of number of data points
# histogram of number of data points
hTS = histogram(ds.smn, ds.tmn, bins=[sbins, tbins])
np.log10(hTS.T).plot(levels=31)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xhistogram/conda/latest/lib/python3.9/
↪site-packages/xarray/core/computation.py:739: RuntimeWarning: divide by zero␣
↪encountered in log10
  result_data = func(*input_data)
```

[10]:
```
<matplotlib.collections.QuadMesh at 0x7f1a8f650310>
```




However, we would like to do a volume census, which requires the data points to be weighted by volume of the grid box.

$$dV = dz * dx * dy \tag{2.1}$$

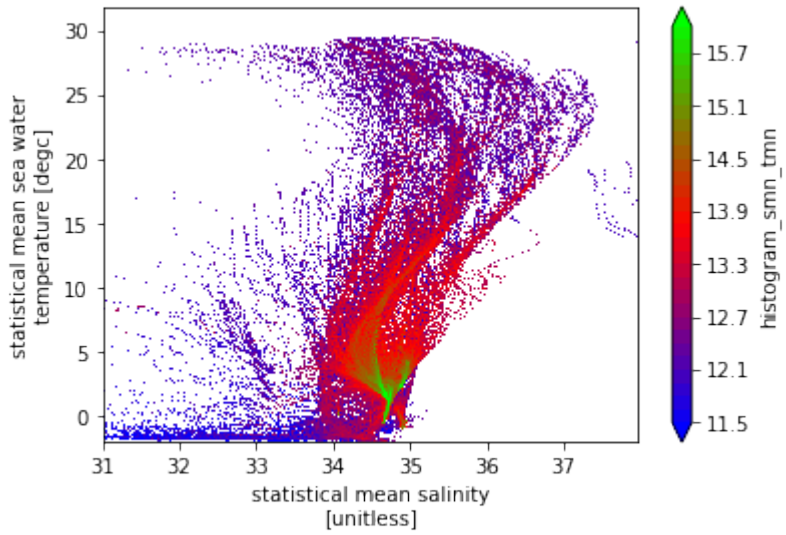
[11]:
```
# histogram of number of data points weighted by volume resolution
# Note that depth is a non-uniform axis

# Create a dz variable
dz = np.diff(ds.lev)
dz =np.insert(dz, 0, dz[0])
dz = xr.DataArray(dz, coords= {'lev':ds.lev}, dims='lev')

# weight by volume of grid cell (resolution = 5degree, 1degree=110km)
dVol = dz * (5*110e3) * (5*110e3*np.cos(ds.lat*np.pi/180))

# Note: The weights are automatically broadcast to the right size
```

(continues on next page)

(continued from previous page)

```
hTSw = histogram(ds.smn, ds.tmn, bins=[sbins, tbins], weights=dVol)
np.log10(hTSw.T).plot(levels=31, vmin=11.5, vmax=16, cmap='brg')
```

[11]:
```
<matplotlib.collections.QuadMesh at 0x7f1a8f59a250>
```




The ridges of this above plot are indicative of T/S classes with a lot of volume, and some of them are indicative of Mode Waters (example Eighteen Degree water with T$\sim 18^oC$, and S$\sim 36.5psu$.

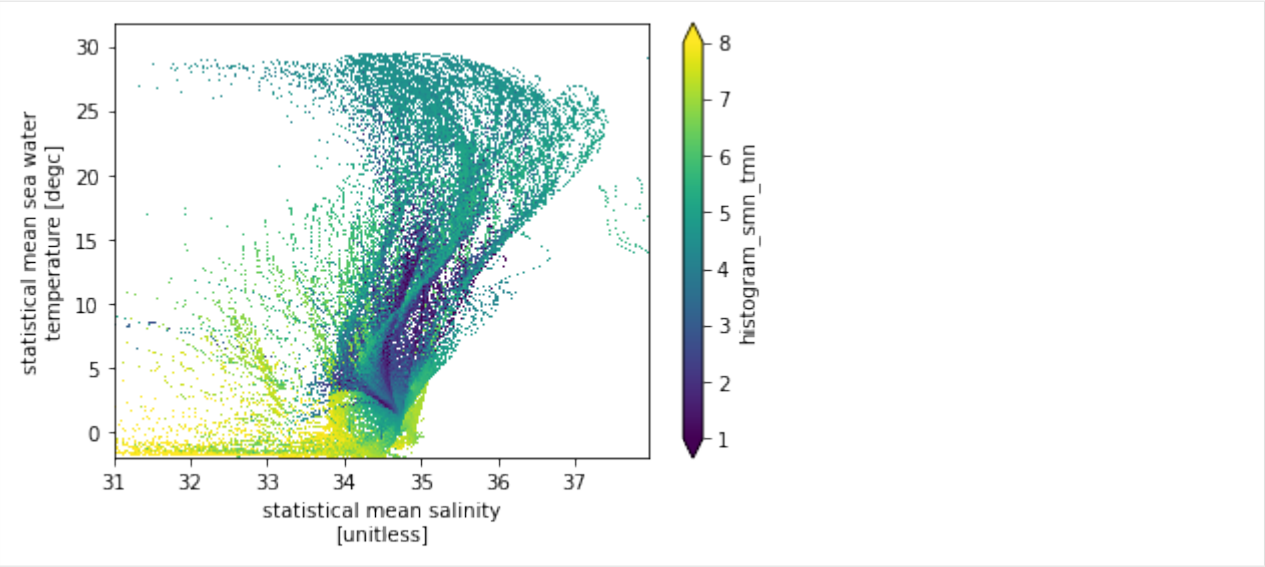
### Averaging a variable

Next we calculate the mean oxygen value in each TS bin.

$$\overline{A}(m,n) = \frac{\sum_{T(x,y,z)=m, S(x,y,z)=n}(A(x,y,z)dV)}{\sum_{T(x,y,z)=m, S(x,y,z)=n} dV}. \tag{2.2}$$

[12]:
```
hTSO2 = (histogram(ds.smn.where(~np.isnan(ds.omn)),
                   ds.tmn.where(~np.isnan(ds.omn)),
                   bins=[sbins, tbins],
                   weights=ds.omn.where(~np.isnan(ds.omn))*dVol)/
                histogram(ds.smn.where(~np.isnan(ds.omn)),
                          ds.tmn.where(~np.isnan(ds.omn)),
                          bins=[sbins, tbins],
                          weights=dVol))

(hTSO2.T).plot(vmin=1, vmax=8)
```

[12]:
```
<matplotlib.collections.QuadMesh at 0x7f1a8f4df130>
```




Some interesting patterns in average oxygen emerge. Convectively ventilated cold water have the highest oxygen and mode waters have relatively high oxygen. Oxygen minimum zones are interspersed in the middle of volumetic ridges (high volume waters).

**NOTE**: NaNs in weights will make the weighted sum as nan. To avoid this, call `.fillna(0.)` on your weights input data before calling `histogram()`.

### 2.2.4 Dask Integration

Should just work, but need examples.

## 2.3 API

### 2.3.1 Core Module

Numpy API for xhistogram.

`xhistogram.core.`**`histogram`**(**args*, *bins=None*, *range=None*, *axis=None*, *weights=None*, *density=False*, *block_size='auto'*)

Histogram applied along specified axis / axes.

**Parameters**

**args** [array_like] Input data. The number of input arguments determines the dimensionality of the histogram. For example, two arguments produce a 2D histogram. All args must have the same size.

**bins** [int, str or numpy array or a list of ints, strs and/or arrays, optional] If a list, there should be one entry for each item in `args`. The bin specifications are as follows:

- If int; the number of bins for all arguments in `args`.
- If str; the method used to automatically calculate the optimal bin width for all arguments in `args`, as defined by numpy *histogram_bin_edges*.
- If numpy array; the bin edges for all arguments in `args`.

- If a list of ints, strs and/or arrays; the bin specification as above for every argument in `args`.

When bin edges are specified, all but the last (righthand-most) bin include the left edge and exclude the right edge. The last bin includes both edges.

A TypeError will be raised if args contains dask arrays and bins are not specified explicitly as an array or list of arrays. This is because other bin specifications trigger computation.

**range** [(float, float) or a list of (float, float), optional] If a list, there should be one entry for each item in `args`. The range specifications are as follows:

- If (float, float); the lower and upper range(s) of the bins for all arguments in `args`. Values outside the range are ignored. The first element of the range must be less than or equal to the second. *range* affects the automatic bin computation as well. In this case, while bin width is computed to be optimal based on the actual data within *range*, the bin count will fill the entire range including portions containing no data.
- If a list of (float, float); the ranges as above for every argument in `args`.
- If not provided, range is simply `(arg.min(), arg.max())` for each arg.

**axis** [None or int or tuple of ints, optional] Axis or axes along which the histogram is computed. The default is to compute the histogram of the flattened array

**weights** [array_like, optional] An array of weights, of the same shape as *a*. Each value in *a* only contributes its associated weight towards the bin count (instead of 1). If *density* is True, the weights are normalized, so that the integral of the density over the range remains 1.

**density** [bool, optional] If `False`, the result will contain the number of samples in each bin. If `True`, the result is the value of the probability *density* function at the bin, normalized such that the *integral* over the range is 1. Note that the sum of the histogram values will not be equal to 1 unless bins of unity width are chosen; it is not a probability *mass* function.

**block_size** [int or 'auto', optional] A parameter which governs the algorithm used to compute the histogram. Using a nonzero value splits the histogram calculation over the non-histogram axes into blocks of size `block_size`, iterating over them with a loop (numpy inputs) or in parallel (dask inputs). If `'auto'`, blocks will be determined either by the underlying dask chunks (dask inputs) or an experimental built-in heuristic (numpy inputs).

**Returns**

**hist** [array] The values of the histogram.

**bin_edges** [list of arrays] Return the bin edges for each input array.

**See also:**

**`numpy.histogram`, `numpy.bincount`, `numpy.searchsorted`**

### 2.3.2 Xarray Module

Xarray API for xhistogram.

`xhistogram.xarray.`**`histogram`**(**args*, *bins=None*, *range=None*, *dim=None*, *weights=None*, *density=False*, *block_size='auto'*, *keep_coords=False*, *bin_dim_suffix='_bin'*)

Histogram applied along specified dimensions.

**Parameters**

**args** [xarray.DataArray objects] Input data. The number of input arguments determines the dimensonality of the histogram. For example, two arguments prodocue a 2D histogram. All args must be aligned and have the same dimensions.

**bins** [int, str or numpy array or a list of ints, strs and/or arrays, optional] If a list, there should be one entry for each item in `args`. The bin specifications are as follows:

- If int; the number of bins for all arguments in `args`.
- If str; the method used to automatically calculate the optimal bin width for all arguments in `args`, as defined by numpy *histogram_bin_edges*.
- If numpy array; the bin edges for all arguments in `args`.
- If a list of ints, strs and/or arrays; the bin specification as above for every argument in `args`.

When bin edges are specified, all but the last (righthand-most) bin include the left edge and exclude the right edge. The last bin includes both edges.

A TypeError will be raised if args contains dask arrays and bins are not specified explicitly as an array or list of arrays. This is because other bin specifications trigger computation.

**range** [(float, float) or a list of (float, float), optional] If a list, there should be one entry for each item in `args`. The range specifications are as follows:

- If (float, float); the lower and upper range(s) of the bins for all arguments in `args`. Values outside the range are ignored. The first element of the range must be less than or equal to the second. *range* affects the automatic bin computation as well. In this case, while bin width is computed to be optimal based on the actual data within *range*, the bin count will fill the entire range including portions containing no data.
- If a list of (float, float); the ranges as above for every argument in `args`.
- If not provided, range is simply `(arg.min(), arg.max())` for each arg.

**dim** [tuple of strings, optional] Dimensions over which which the histogram is computed. The default is to compute the histogram of the flattened array.

**weights** [array_like, optional] An array of weights, of the same shape as *a*. Each value in *a* only contributes its associated weight towards the bin count (instead of 1). If *density* is True, the weights are normalized, so that the integral of the density over the range remains 1. NaNs in the weights input will fill the entire bin with NaNs. If there are NaNs in the weights input call `.fillna(0.)` before running `histogram()`.

**density** [bool, optional] If `False`, the result will contain the number of samples in each bin. If `True`, the result is the value of the probability *density* function at the bin, normalized such that the *integral* over the range is 1. Note that the sum of the histogram values will not be equal to 1 unless bins of unity width are chosen; it is not a probability *mass* function.

**block_size** [int or 'auto', optional] A parameter which governs the algorithm used to compute the histogram. Using a nonzero value splits the histogram calculation over the non-histogram axes into blocks of size `block_size`, iterating over them with a loop (numpy inputs) or in parallel (dask inputs). If `'auto'`, blocks will be determined either by the underlying dask chunks (dask inputs) or an experimental built-in heuristic (numpy inputs).

**keep_coords** [bool, optional] If `True`, keep all coordinates. Default: `False`

**bin_dim_suffix** [str, optional] Suffix to append to input arg names to define names of output bin dimensions

**Returns**

**hist** [xarray.DataArray] The values of the histogram. For each bin, the midpoint of the bin edges is given along the bin coordinates.

## 2.4 Contributor Guide

This package is in very early stages. Lots of work is needed.

You can help out just by using `xhistogram` and reporting issues.

The following sections cover some general guidelines for maintainers and contributors wanting to help develop `xhistogram`.

### 2.4.1 Feature requests, suggestions and bug reports

We are eager to hear about any bugs you have found, new features you would like to see and any other suggestions you may have. Please feel free to submit these as issues.

When suggesting features, please make sure to explain in detail how the proposed feature should work and to keep the scope as narrow as possible. This makes features easier to implement in small PRs.

When report bugs, please include:

- Any details about your local setup that might be helpful in troubleshooting, specifically the Python interpreter version, installed libraries, and `xhistogram` version.
- Detailed steps to reproduce the bug, ideally a Minimal, Complete and Verifiable Example.
- If possible, a demonstration test that currently fails but should pass when the bug is fixed.

### 2.4.2 Write documentation

Adding documentation is always helpful. This may include:

- More complementary documentation. Have you perhaps found something unclear?
- Docstrings.
- Example notebooks of `xhistogram` being used in real analyses.

The `xhistogram` documentation is written in reStructuredText. You can follow the conventions in already written documents. Some helpful guides can be found here and here.

When writing and editing documentation, it can be useful to see the resulting build without having to push to Github. You can build the documentation locally by running:

```
$ # Install the packages required to build the docs in a conda environment
$ conda env create -f doc/environment.yml
$ conda activate xhistogram_doc_env
$ # Install the latest xhistogram
$ pip install --no-deps -e .
$ cd doc/
$ make html
```

This will build the documentation locally in `doc/_build/`. You can then open `_build/html/index.html` in your web browser to view the documentation. For example, if you have `xdg-open` installed:

```
$ xdg-open _build/html/index.html
```

To lint the reStructuredText documentation files run:

```
$ doc8 doc/*.rst
```

### 2.4.3 Preparing Pull Requests

1. Fork the xhistogram GitHub repository. It's fine to use `xhistogram` as your fork repository name because it will live under your username.
2. Clone your fork locally, connect your repository to the upstream (main project), and create a branch to work on:

   ```
   $ git clone git@github.com:YOUR_GITHUB_USERNAME/xhistogram.git
   $ cd xhistogram
   $ git remote add upstream git@github.com:xgcm/xhistogram.git
   $ git checkout -b your-bugfix-feature-branch-name master
   ```

   If you need some help with Git, follow this quick start guide
3. Install dependencies into a new conda environment:

   ```
   $ conda env create -f ci/environment-3.9.yml
   $ conda activate xhistogram_test_env
   ```
4. Install xhistogram using the editable flag (meaning any changes you make to the package will be reflected directly in your environment):

   ```
   $ pip install --no-deps -e .
   ```
5. Start making your edits. Please try to type annotate your additions as much as possible. Adding type annotations to existing unannotated code is also very welcome. You can read about Python typing here.
6. Break your edits up into reasonably sized commits:

   ```
   $ git commit -a -m "<commit message>"
   $ git push -u
   ```

   It can be useful to manually run pre-commit as you make your edits. `pre-commit` will run checks on the format and typing of your code and will show you where you need to make changes. This will mean your code is more likely to pass the CI checks when you push it:

   ```
   $ pip install pre_commit # you only need to do this once
   $ pre-commit run --all-files
   ```
7. Run the tests (including those you add to test your edits!):

   ```
   $ pytest xhistogram
   ```

   You can also test that your contribution and tests increased the test coverage:

   ```
   $ coverage run --source xhistogram -m py.test
   $ coverage report
   ```

8. Add a new entry describing your contribution to the *Release History* in `doc/contributing.rst`. Please try to follow the format of the existing entries.
9. Submit a pull request through the GitHub website.

   Note that you can create the Pull Request while you're working on your PR. The PR will update as you add more commits. `xhistogram` developers and contributors can then review your code and offer suggestions.

### 2.4.4 Release History

#### v0.3.0

- Add support for histograms over non-float dtypes (e.g. datetime objects) GH25. By Dougie Squire.
- Refactor histogram calculation to use dask.array.blockwise when input arguments are dask arrays, resulting in significant performance improvements GH49. By Ryan Abernathy, Tom Nicholas and Gabe Joseph.
- Fixed bug with density calculation when NaNs are present GH51. By Dougie Squire.
- Implemented various options for users for providing bins to xhistogram that mimic the numpy histogram API. This included adding a range argument to the xhistogram API GH13. By Dougie Squire.
- Added a function to check if the object passed to xhistogram is an xarray.DataArray and if not, throw an error. GH14. By Yang Yunyi.

#### v0.2.0

- Added FutureWarning for upcoming changes to core API GH13. By Dougie Squire.
- Added documentation on how to deal with NaNs in weights GH26. By Shanice Bailey.
- Move CI to GitHub actions GH32. By James Bourbeau.
- Add documentation for contributors. By Dougie Squire.
- Add type checking with mypy GH32. By Dougie Squire.

#### v0.1.3

- Update dependencies to exclude incompatible dask version GH27. By Ryan Abernathey.

#### v0.1.2

- Aligned definition of `bins` with `numpy.histogram` GH18. By Dougie Squire.

#### v0.1.1

Minor bugfix release

- Imroved documentation examples. By Dhruv Balwada.
- Fixed issue GH5 related to incorrect dimension order and dropping of dimension coordinates. By Ryan Abernathey.

### v0.1

First release

# PYTHON MODULE INDEX

# INDEX

## H



## M



## X