boost-histogram and hist

Henry Schreiner

April 15, 2019

Histograms in Python

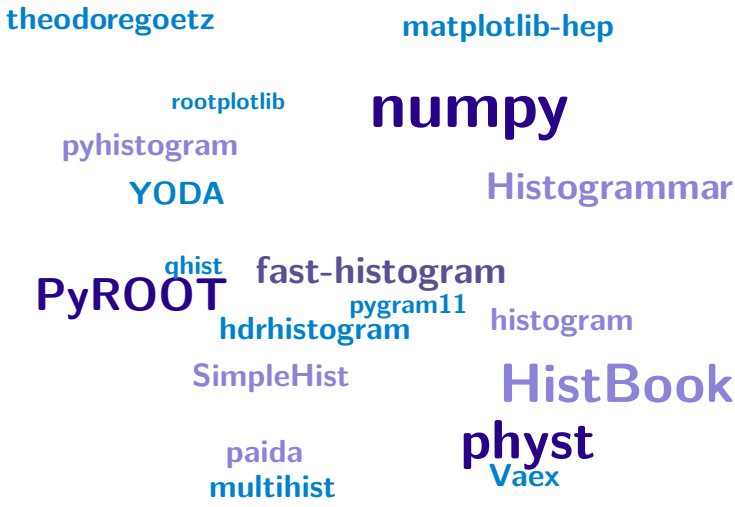
Current state of histograms in Python

Core library: numpy

- Historically slow
- No histogram object
- Plotting is separate

Other libraries

- Narrow focus: speed, plotting, or language
- Many are abandoned
- Poor design, backends, distribution



Design

- A histogram should be an object
- Manipulation and plotting should be easy

Flexibility

- Axes options: sparse, growing, labels
- Storage: integers, weights, errors...

Performance

- Fast single threaded filling
- Multithreaded filling (since it's 2019)

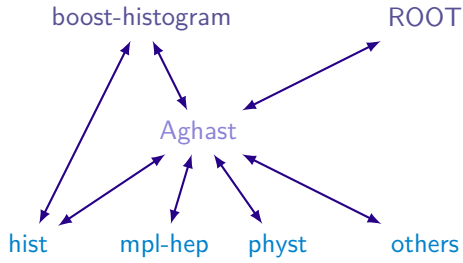
Distribution

- Easy to use anywhere, pip or conda
- Should have wheels, be easy to build, etc.

Core histogramming libraries

Universal adaptor

Front ends (plotting, etc)



Boost::Histogram (C++14)

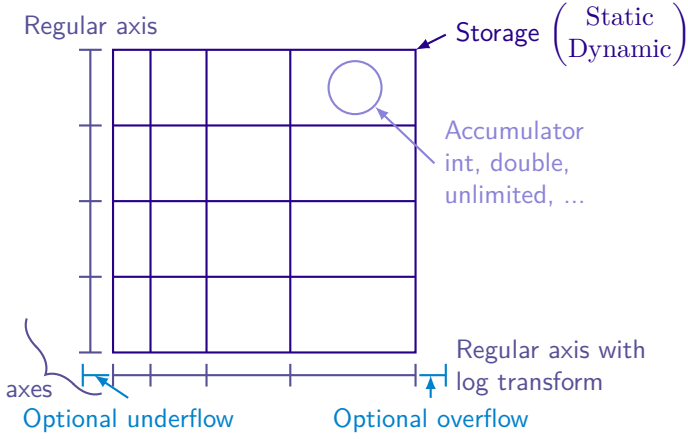
- Multidimensional templated header-only histogram library: /boostorg/histogram
- Designed by Hans Dembinski, inspired by ROOT, GSL, and histbook

Histogram

- Axes
- Storages
- Accumulators

Axes types

- Regular, Circular
- Variable
- Integer
- Category



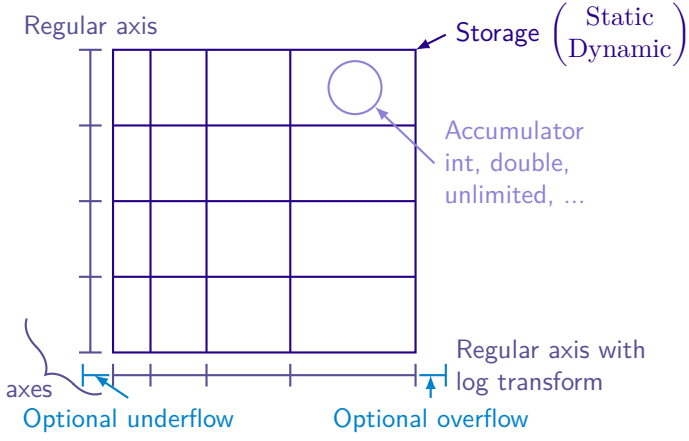
- Multidimensional templated header-only histogram library: /boostorg/histogram
- Designed by Hans Dembinski, inspired by ROOT, GSL, and histbook

Histogram

- Axes
- Storages
- Accumulators

Axes types

- Regular, Circular
- Variable
- Integer
- Category



Boost 1.70 now released with Boost::Histogram!

boost-histogram (Python)

- Boost::Histogram developed with Python in mind
- Original bindings based on Boost::Python
  - Hard to build and distribute
  - Somewhat limited
- New bindings: /scikit-hep/boost-histogram
  - 0-dependency build (C++14 only)
  - State-of-the-art PyBind11

Design | Flexibility | Speed | Distribution

- Supports Python 2.7 and 3.4+
- 260+ unit tests run on Azure on Linux, macOS, and Windows
- Up to 16 axes supported (may go up or down)
- 1D, 2D, and ND histograms all have the same interface

Tries to stay close to the original Boost::Histogram where possible.

C++

```cpp
#include <boost/histogram.hpp>
namespace bh = boost::histogram;

auto hist = bh::make_histogram(
  bh::axis::regular<>{2, 0, 1, "x"},
  bh::axis::regular<>{4, 0, 1, "y"});

hist(.2, .3);
```

Python

```python
import boost.histogram as bh

hist = bh.make_histogram(
  bh.axis.regular(2, 0, 1, metadata="x"),
  bh.axis.regular(4, 0, 1, metadata="y"))

hist(.2, .3)
```

Combine two histograms
```
hist1 + hist2
```

Scale a histogram
```
hist * 2.0
```

Project a 3D histogram to 2D
```
hist.project(0,1) # select axis
```

Sum a histogram contents
```
hist.sum()
```

Access an axis
```
axis0 = hist.axis(0)
axis0.edges() # The edges array
axis0.bin(1) # The bin accessors
```

Fill 2D histogram with values or arrays
```
hist(x, y)
```

Fill copies in 4 threads, then merge
```
hist.fill_threaded(4, x, y)
```

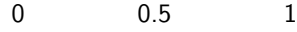
Fill in 4 threads (atomic storage only)
```
hist.fill_atomic(4, x, y)
```

Convert to Numpy, 0-copy
```
hist.view()
# Or
np.asarray(hist)
```

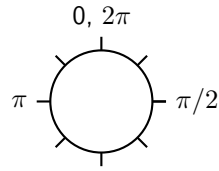
Flexibility: Axis

boost-histogram (Python)

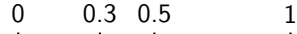
- `bh.axis.regular`
  - `bh.axis.regular_uoflow`
  - `bh.axis.regular_noflow`
  - `bh.axis.regular_growth`
- `bh.axis.circular`
- `bh.axis.regular_log`
- `bh.axis.regular_sqrt`
- `bh.axis.regular_pow`
- `bh.axis.integer`
- `bh.axis.integer_noflow`
- `bh.axis.integer_growth`
- `bh.axis.variable`
- `bh.axis.category_int`
- `bh.axis.category_int_growth`



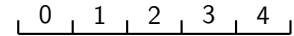
`bh.axis.regular(10,0,1)`



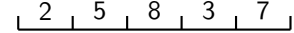
`bh.axis.circular(8,0,2*np.pi)`



`bh.axis.variable([0,.3,.5,1])`



`bh.axis.integer(0,5)`



`bh.axis.category_int([2,5,8,3,7])`

Flexibility: Storage types

- `bh.storage.int`
- `bh.storage.double`
- `bh.storage.unlimited` (WIP)
- `bh.storage.atomic_int`
- `bh.storage.weight` (WIP)
- `bh.storage.profile` (WIP, needs sampled fill)
- `bh.storage.weighted_profile` (WIP, needs sampled fill)

The following measurements are with:

1D

- 100 regular bins
- 10,000,000 entries

2D

- 100x100 regular bins
- 1,000,000 entries

See my histogram performance post for measurements of other libraries.

Performance: macOS, dual core, 1D

Type	Storage	Fill	Time	Speedup
Numpy	uint64		149.4 ms	1x
Any	int		236 ms	0.63x
Regular	int		86.23 ms	1.7x
Regular	aint	1	132 ms	1.1x
Regular	aint	2	168.2 ms	0.89x
Regular	aint	4	143.6 ms	1x
Regular	int	1	84.75 ms	1.8x
Regular	int	2	51.6 ms	2.9x
Regular	int	4	42.39 ms	3.5x

Type	Storage	Fill	Time	Speedup
Numpy	uint64		121 ms	1x
Any	int		261.5 ms	0.46x
Regular	int		142.2 ms	0.85x
Regular	aint	1	319.1 ms	0.38x
Regular	aint	48	272.9 ms	0.44x
Regular	int	1	243.4 ms	0.5x
Regular	int	6	94.76 ms	1.3x
Regular	int	12	71.38 ms	1.7x
Regular	int	24	52.26 ms	2.3x
Regular	int	48	43.01 ms	2.8x

Performance: KNL, 64 core, 1D (anaconda)

boost-histogram (Python)

Type	Storage	Fill	Time	Speedup
Numpy	uint64		716.9 ms	1x
Any	int		1418 ms	0.51x
Regular	int		824 ms	0.87x
Regular	aint	1	871.7 ms	0.82x
Regular	aint	4	437.1 ms	1.6x
Regular	aint	64	198.8 ms	3.6x
Regular	aint	128	186.8 ms	3.8x
Regular	aint	256	195.2 ms	3.7x
Regular	int	1	796.9 ms	0.9x
Regular	int	2	430.6 ms	1.7x
Regular	int	4	247.6 ms	2.9x
Regular	int	64	88.77 ms	8.1x
Regular	int	128	98.08 ms	7.3x
Regular	int	256	112.2 ms	6.4x

Performance: macOS, dual core, 2D

Type	Storage	Fill	Time	Speedup
Numpy	uint64		121.1 ms	1x
Any	int		37.12 ms	3.3x
Regular	int		18.5 ms	6.5x
Regular	aint	1	20.21 ms	6x
Regular	aint	2	14.17 ms	8.5x
Regular	aint	4	10.23 ms	12x
Regular	int	1	17.86 ms	6.8x
Regular	int	2	9.41 ms	13x
Regular	int	4	6.854 ms	18x

Type	Storage	Fill	Time	Speedup
Numpy	uint64		87.27 ms	1x
Any	int		41.42 ms	2.1x
Regular	int		21.67 ms	4x
Regular	aint	1	38.61 ms	2.3x
Regular	aint	6	19.89 ms	4.4x
Regular	aint	24	9.556 ms	9.1x
Regular	aint	48	8.518 ms	10x
Regular	int	1	36.5 ms	2.4x
Regular	int	6	8.976 ms	9.7x
Regular	int	12	5.318 ms	16x
Regular	int	24	4.388 ms	20x
Regular	int	48	5.839 ms	15x

Performance: KNL, 64 core, 2D (anaconda)

boost-histogram (Python)

Type	Storage	Fill	Time	Speedup
Numpy	uint64		439.5 ms	1x
Any	int		250.6 ms	1.8x
Regular	int		135.6 ms	3.2x
Regular	aint	1	142.2 ms	3.1x
Regular	aint	4	52.71 ms	8.3x
Regular	aint	32	12.05 ms	36x
Regular	aint	64	16.5 ms	27x
Regular	aint	256	43.93 ms	10x
Regular	int	1	141.1 ms	3.1x
Regular	int	2	70.78 ms	6.2x
Regular	int	4	36.11 ms	12x
Regular	int	64	18.93 ms	23x
Regular	int	128	36.09 ms	12x
Regular	int	256	55.64 ms	7.9x

System	1D max speedup	2D max speedup
macOS 1 core	1.7 x	6.5 x
macOS 2 core	3.5 x	18 x
Linux 1 core	0.85 x	4 x
Linux 24 core	2.8 x	20 x
KNL 1 core	0.87 x	3.2 x
KNL 64 core	8.1 x	36 x

- Note that Numpy 1D is well optimized (last few versions)
- Anaconda versions may provide a few more optimizations to Numpy
- Mixing axes types in boost-histogram can reduce performance by 2-3x

- We *must* provide excellent distribution.
  - If anyone writes `pip install boost-histogram` and it fails, we have failed.
- Docker ManyLinux1 GCC 8.3: /scikit-hep/manylinuxgcc

Wheels

- manylinux1 32, 64 bit (ready)
- manylinux2010 64 bit (planned)
- macOS 10.9+ (wip)
- Windows 32, 64 bit, Python 3.6+ (wip)
  - Is Python 2.7 Windows needed?

Source

- SDist (ready)
- Build directly from GitHub (done)

Conda

- conda package (planned, easy)

```
python -m pip install \
    git+https://github.com/scikit-hep/boost-histogram.git@develop
```

- Add shortcuts for axis types, fill out axis types
- Allow view access into unlimited storage histograms
- Add `from_numpy` and numpy style shortcut(s)
- Filling
  - Samples
  - Weights
  - Non-numerical fill (if possible)
- Add profile, weighted_profile histograms
- Add reduce operations
- Release to PyPI
- Add some docs and read the docs support

First alpha

Release planned this week

Let's discuss API! (On GitHub issues or gitter)

- Download: `pip install boost-histogram` (WIP)
- Use: `import boost.histogram as bh`
- Create: `hist = bh.histogram(bh.axis.regular(12,0,1))`
- Fill: `hist(values)`
- Access values, convert to numpy, etc.

Documentation

- The documentation will also need useful examples, feel free to contribute!

hist

`hist` is the 'wrapper' piece that does plotting and interacts with the rest of the ecosystem.

Plans

- Easy plotting adaptors (mpl-hep)
- Serialization formats (ROOT, HDF5)
- Auto-multithreading
- Statistical functions (Like TEfficiency)
- Multihistograms (HistBook)
- Interaction with fitters (ZFit, GooFit, etc)
- Bayesian Blocks algorithm from SciKit-HEP
- Command line histograms for stream of numbers

Call for contributions

- What do you need?
- What do you want?
- What would you like?

Join in the development! This should combine the best features of other packages.

Questions?

Questions?

- Supported by IRIS-HEP, NSF OAC-1836650