Histogramming

Henry Schreiner

October 17, 2019

iris hep
Institute for Research & Innovation in Software for High Energy Physics

PRINCETON UNIVERSITY

Overview

- Part 1: Overview of histograms
  - Components of a Histogram
  - Histograms in Python
  - Boost.Histogram in C++14
  - Introducing: `boost-histogram` for Python
  - Outlook, with `hist` and `aghast`
- Part 2: Hands-on with `boost-histogram`

What is a histogram?

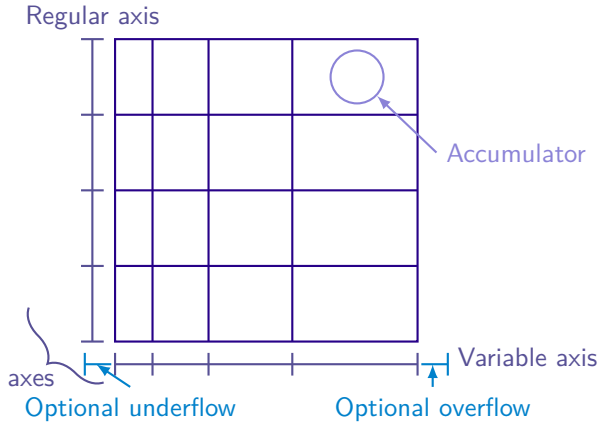
- A histogram is a set of accumulators over data in ranges
  - Usually continuous in Physics, could also be categories
  - Accumulators often are a sum of values - can contain other components
- Input values are digitized by axes (AKA binnings)
  - Categories
  - Real values
    - Variable sized bins (usually give edges)
    - Regular binning (#bins, start, stop)
  - May have special features (overflow, circular, etc.)

Histogram components

A 'histogram is a collection of 1+ axes and an accumulator.

Performance

- Variable axis - list of edges is most general but requires a sorted search.
- Regular axis: regular spacing



Histograms in (classic) PyROOT

- 1D Regular

```
h = ROOT.TH1D("", "", 10, 0, 1)
h.fillN(arr)
```

- 1D Variable

```
h = ROOT.TH1D("", "", (1,2,3,4,5,6))
h.fillN(arr)
```

- 2D Regular

```
h = ROOT.TH2D("", "", 10, 0, 1, 20, 0, 2)
h.fillN(arr)
```

Histogram in Numpy

- 1D Regular

```
bins, edges = np.histogram(arr, bins=10, range=(0,1))
```

- 1D Variable

```
bins, edges = np.histogram(arr, bins=(1,2,3,4,5,6))
```

- 2D regular

```
b, e1, e2 = np.histogram2d(x, y, bins=(10,20), range=((0,1),(0,2)))
```

Numpy Pros and Cons

Pros

- Comes with Numpy
- Good for interactive operations (auto binning)
- *Reasonably* fast
- Density option, weight support too

Cons

- Manipulation of plain arrays
- One time fill
- 2D+ not optimized for regular binning
- 1D, 2D, and ND syntax variations
- MPL had to mimic: `plt.hist`

PyROOT Pros and Cons

Pros

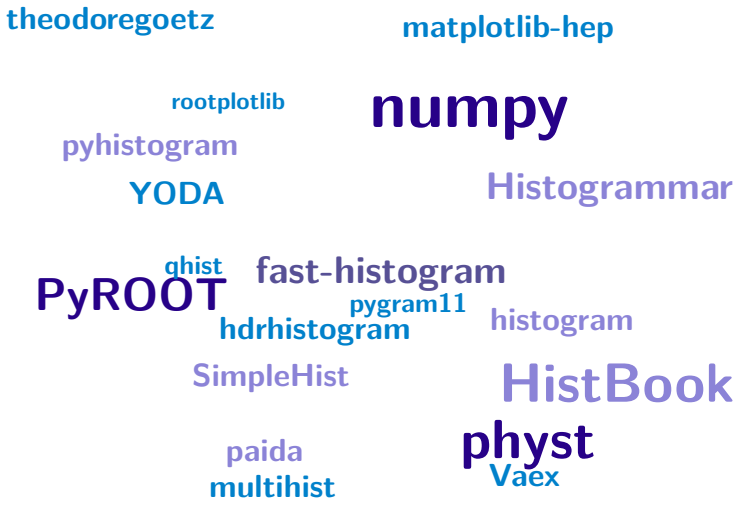
- Full histogram object
- Iterative fill option
- Weights option
- Can track sum of weights too

Cons

- ROOT requirement (Conda-forge helps)
- Can be slow in Python (and C++)
- Poor interactive exploration
- Odd syntax, odd memory model
- Max 3D

Histogram Libraries

- Narrow focus: speed, plotting, or language
- Many are abandoned
- Often issues with design, backends, distribution
- No/little *interaction*



Physt

- Histograms as objects
- Pure Python - Dropped Python 2 this year :)
- Very slow fills (slower than numpy)

```
hist = histogram(heights)
hist.plot(show_values=True)
```

- Powerful plotting
- Easy conversion to Pandas and many more (ROOT through uproot)
- Special histograms, like polar histograms

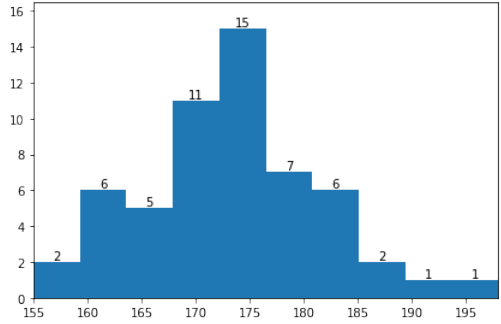


Figure 1: Physt example default plot

Fast-Histogram

- Exactly like numpy, but faster
  - C kernel
  - Takes advantage of regular binning
  - Can be 20-25x faster for 2D histograms
  - Missing some features / combinations

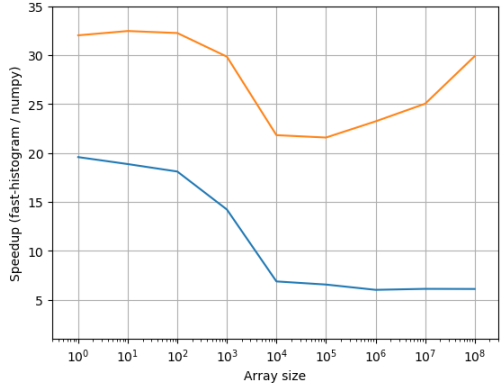


Figure 2: Fast Histogram 2d comparison with Numpy

HistBook (archived)

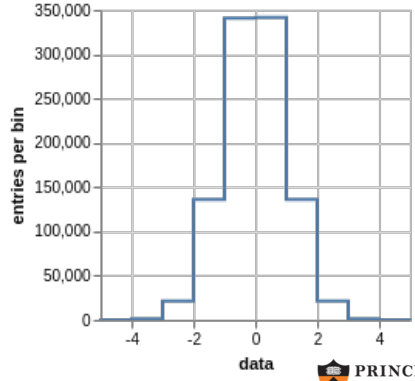
histbook

The first Scikit-HEP library for histograms

- Designed for shared axis histogram collections
- Plotting with Vega-Light

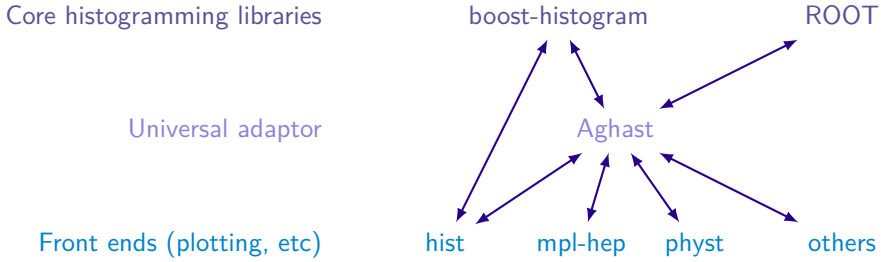
Now deprecated and in archive mode, functionality may return in Hist (see next slides).

```
>>> array = np.random.normal(0, 1, 1000000)
>>> histogram = Hist(bin("data", 10, -5, 5))
>>> histogram.fill(data=array)
>>> histogram.step("data").to(canvas)
```



SciKit-HEP Histogramming plan

- boost-histogram: Fast filling and manipulation (core library)
- hist: Simple analysis frontend
- aghast: Conversions between histogram libraries
- UHI: Unified Histogram Indexing: A way for histograms to be indexed cross-library (boost-histogram and hist to begin with)



Boost.Histogram C++14

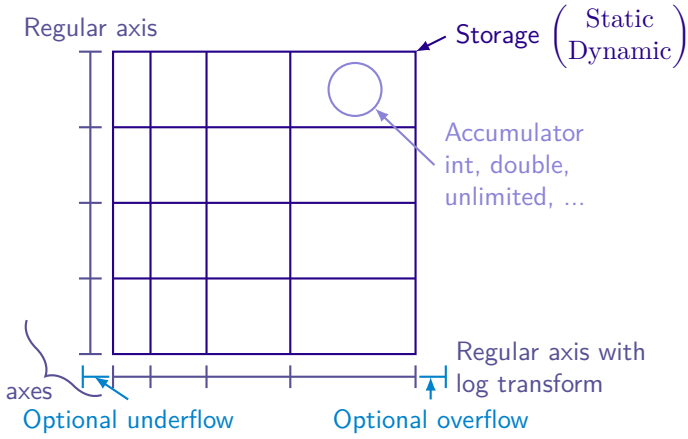
- Multidimensional templated header-only histogram library: /boostorg/histogram
- Designed by Hans Dembinski, inspired by ROOT and GSL

Histogram

- Axes
- Storage

Axes types

- Regular, Circular
- Variable
- Integer
- Category



Boost.Histogram example

```cpp
#include <boost/histogram.hpp>
#include <boost/histogram/ostream.hpp>
#include <random>

int main() {
    namespace bh = boost::histogram;

    auto hist = bh::make_histogram(bh::axis::regular<>{20, -3, 3});

    std::default_random_engine eng;
    std::normal_distribution<double> dist(0, 1);
    for(int n = 0; n < 10'000; ++n)
        hist(dist(eng));

    std::cout << hist << std::endl;
    return 0;
}
```

Boost.Histogram example (output)

```
histogram(regular(20, -3, 3, options=underflow | overflow))
                  +------------------------------------------------------------
+
[-inf,   -3) 9    |                                                           |
[  -3, -2.7) 19   |=                                                          |
[-2.7, -2.4) 36   |==                                                         |
[-2.4, -2.1) 110  |=====                                                      |
[-2.1, -1.8) 191  |=========                                                  |
[-1.8, -1.5) 275  |=============                                              |
[-1.5, -1.2) 518  |=========================                                  |
[-1.2, -0.9) 644  |==============================                             |
[-0.9, -0.6) 914  |===========================================                |
[-0.6, -0.3) 1107 |====================================================       |
[-0.3,    0) 1183 |========================================================== |
[   0,  0.3) 1185 |========================================================== |
[ 0.3,  0.6) 1120 |=====================================================      |
[ 0.6,  0.9) 874  |=========================================                  |
[ 0.9,  1.2) 663  |===============================                            |
[ 1.2,  1.5) 491  |=======================                                    |
[ 1.5,  1.8) 322  |===============                                            |
[ 1.8,  2.1) 172  |========                                                   |
[ 2.1,  2.4) 79   |====                                                       |
[ 2.4,  2.7) 38   |==                                                         |
[ 2.7,    3) 28   |=                                                          |
[   3,  inf) 22   |=                                                          |
                  +------------------------------------------------------------
+
```

boost-histogram: Python bindings

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 filling
- Compiled composable manipulations

Distribution

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

Intro to the Python bindings

- 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

Design

- 500+ unit tests run on Azure on Linux, macOS, and Windows

Resembles the original Boost.Histogram where possible, with changes where needed for Python performance and idioms.

C++14

```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); // Fill will also be
hist(.4, .5); // availble in 1.7.2
hist(.3, .2);
```

Python

```python
import boost.histogram as bh

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

hist.fill(
    [.2, .4, .3],
    [.3, .5, .2])
```

Design: Manipulations

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

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

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

Access an axis
```
ax = hist.axis(0)
ax.edges   # The edges array
ax.centers # Centers of bins
ax.widths  # Width of each bin
```

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

Convert contents to Numpy array
```
hist.view()
```

Convert to Numpy style histogram tuple
```
hist.to_numpy()
```

Pickle supported (multiprocessing)
```
pickle.dumps(hist, -1)
```

Copy/deepcopy supported
```
hist2 = copy.deepcopy(hist)
```

Unified Histogram Indexing (UHI)

The language here (`bh.loc`, etc) is defined in such a way that any library can provide them - "Unified".

Access

```
v = h[b]              # Returns bin contents, indexed by bin number
v = h[bh.loc(b)]      # Returns the bin containing the value
v = h[bh.underflow] # Underflow and overflow can be accessed with special tags
```

Setting

```
h[b] = v
h[bh.loc(b)] = v
h[bh.underflow] = v
```

Unified Histogram Indexing (UHI) (2)

```
h == h[:]                # Slice over everything
h2 = h[a:b]              # Slice of histogram (includes flow bins)
h2 = h[:b]               # Leaving out endpoints is okay
h2 = h[bh.loc(v):]       # Slices can be in data coordinates, too
h2 = h[::bh.project]     # Sum an axis (name may change)
h2 = h[::bh.rebin(2)]    # Modification operations (rebin)
h2 = h[a:b:bh.rebin(2)] # Modifications can combine with slices
h2 = h[a:b, ...]         # Ellipsis work just like normal numpy
```

- Docs are here
- Description may move to a new repository

Performance

- Factor of 2 faster than 1D regular binning in Numpy 1.17
  - Currently no specialization, just a 1D regular fill
  - Could be optimized further
- Factor of 6-10 faster than 2D regular binning Numpy

Distribution

- We *must* provide excellent distribution.
  - If anyone writes `pip install boost-histogram` and it fails, we have failed.
- Docker ManyLinux1 GCC 9.2: /scikit-hep/manylinuxgcc
- Used in /scikit-hep/iMinuit, see /scikit-hep/azure-wheel-helpers

Wheels

- manylinux1 32 and 64 bit, Py 2.7 & 3.5–3.7
- manylinux2010 64 bit, Py 2.7 & 3.5–3.8
- macOS 10.9+ 64 bit, Py 2.7 & 3.6–3.8
- Windows 32 and 64 bit, Py 2.7 & 3.6–3.7

Source

- SDist
- Build directly from GitHub

Conda

- conda-forge package planned

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

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 via Aghast (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.

Aghast

Aghast is a histogramming library that does not fill histograms and does not plot them.

- A memory format for histograms, like Apache Arrow
- Converts to and from other libraries
- Uses flatbuffers to hold histograms
- Indexing ideas inspired the UHI

Binnings

IntegerBinning ▪ RegularBinning ▪ HexagonalBinning ▪ EdgesBinning ▪ IrregularBinning ▪ CategoryBinning ▪ SparseRegularBinning ▪ FractionBinning ▪ PredicateBinning ▪ VariationBinning

End of part 1

Now, we will go hands on with the first beta of boost-histogram!

Support

- Supported by IRIS-HEP, NSF OAC-1836650