Boost-Histogram for Analysis Systems

Team: Henry Schreiner, Hans Dembinski*, Jim Pivarski
Institutions: Princeton University, *TU Dortmund

Institute for Research & Innovation in Software for High Energy Physics

PRINCETON UNIVERSITY

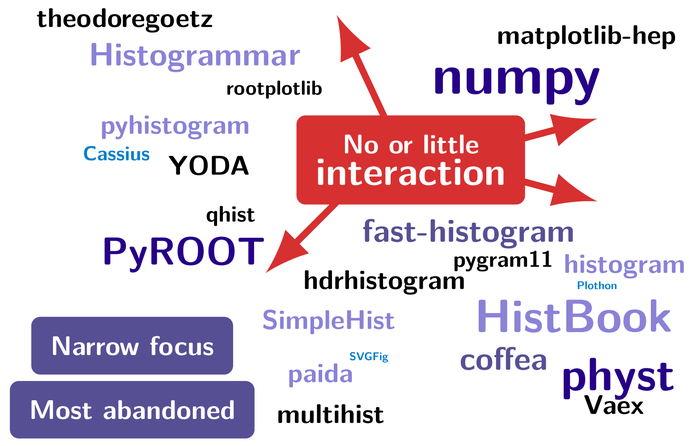
A universal need

Histograms are a universal tool used across disciplines. However, for HEP, rather that just being a useful visualization tool, advanced histograms are often integral to the entire analysis. This is why we have some of the most highly developed histogram tools in C++ in ROOT, and why we need a high quality Python histogramming package.

Boost.Histogram for C++14 was developed by a HEP physisist and accepted as a general tool into the Boost C++ libraries, the most respected third-party library collection in the world. In close collaboration with the author, we have developed boost-histogram for Python.

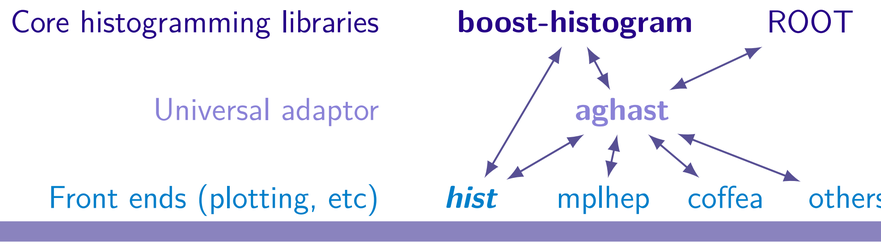
Many attempts at a solution

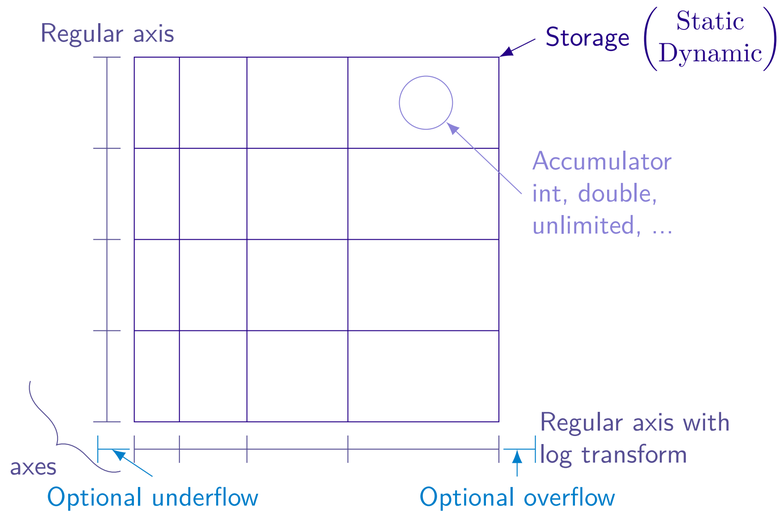
There are many histogramming libraries for Python, but all of the fall short in the key areas we care about: **Design**, **Flexibility**, **Performance**, and **Distribution**. Furthermore, they do not talk to each other.



Our solution

We have proposed the following family of libraries. Unlike previous attempts, we are building a modular solution and an adaptor. Bold indicates the library is related to this project, italics is a planned package.





Storages

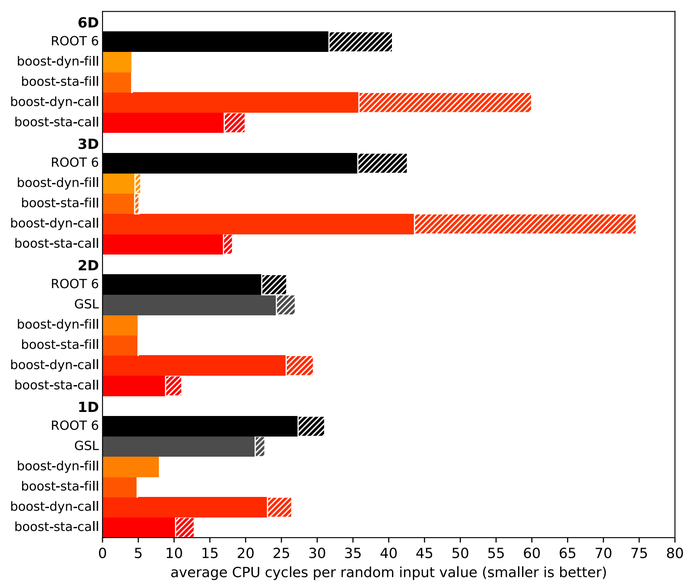
Double: Multipurpose, supports fractional weights.
Int64: Good for simple counts.
AtomicInt64: Threadsafe.
Unlimited: Starts as int8 and grows or converts to double as needed.
WeightedSum: Holds sum of weights squared (ROOT's optional weight tracking).
Mean: A "Profile" histogram, where means reather than sums are kept.
WeightedMean: Like Mean, but tracks the additional variance from varying weights.

Histograms are made up of components:a storage and 0 or more axis. All the ROOT histogram types (TH1D, TH2D, TH3D, THND, TProfile1D, etc, can be represented using just a single histogram type, along with axes types and configuration not possible in ROOT. Everything is available in the formats Python users expect; data can be accessed without copy in any Python library.

Performance

100 bins, 10M events
8-core 2.4 GHz i9 macOS
Numpy: **146 ms**
PyROOT: **123 ms**
boost: **65 ms** or **58 ms**
boost MT: **10.3 ms**

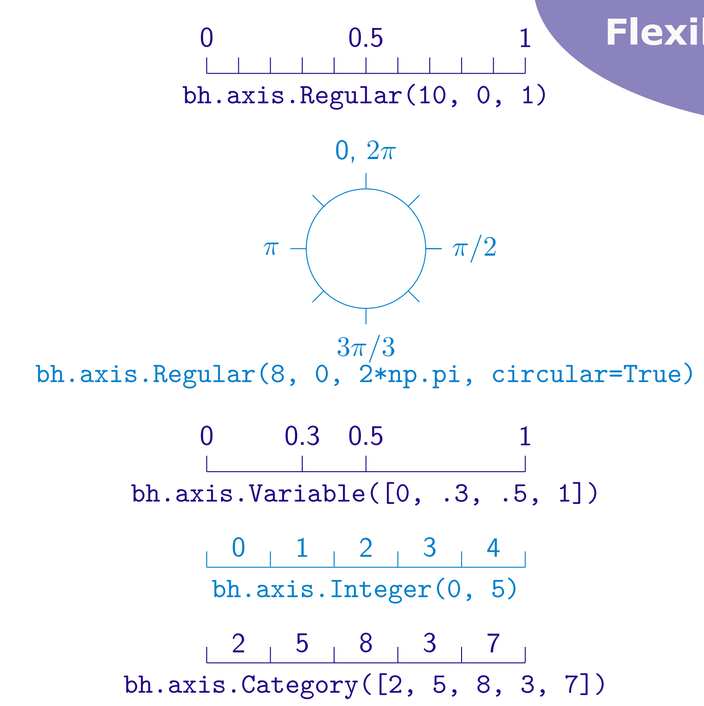
100x100 bins, 10M events
8-core 2.4 GHz i9 macOS
Numpy: **1.18 s**
PyROOT: **157 ms**
boost: **93 ms** or **76 ms**
boost MT: **14.7 ms**



Comparison of fill performance against ROOT and GSL, in C++ using Boost.Histogram. The dyn-fill bars correspond to to boost-histogram in Python.

Flexibility

```
bh.axis.Regular
    under/overflow
    growth=True
    circular=True
    transform=Log()
    transform=Sqrt()
    transform=Pow(v)
    <any C transform>
bh.axis.Integer
    under/overflow
    growth=True
bh.axis.Variable
    under/overflow
    growth=True
bh.axis.IntCategory
bh.axis.StrCategory
    growth=True
```



Distribution

```
pip install boost-histogram
```

Wheels

Windows	macOS	Linux (1 & 2010)
32-bit / 64-bit	64-bit	32-bit / 64-bit
2.7 3.6 3.7 3.8	2.7 3.6 3.7 3.8	2.7 3.5 3.6 3.7 3.8

```
conda install boost-histogram --channel conda-forge
```

Conda

Windows	macOS	Linux
64-bit	64-bit	64-bit / ARM / PowerPC
3.6 3.7 3.8	2.7 3.6 3.7 3.8	2.7 3.6 3.7 3.8

+ support for source builds, with only C++14 requirement

The Azure CI-based wheel build system designed for boost-histogram is now being used in several other Scikit-HEP projects.

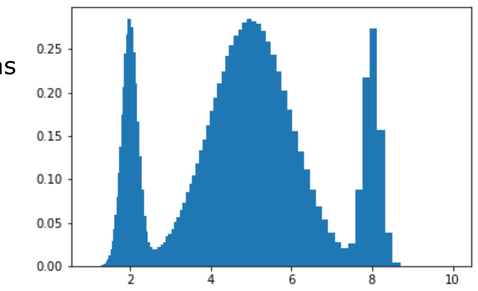
Unified Histogram Indexing (UHI)

```
v = h[b]            # Returns bin contents, indexed by bin number
v = h[loc(b)]       # Returns the bin containing the value
v = h[loc(b) + 1]   # Returns the bin above the one containing the value
v = h[underflow]    # Underflow and overflow can be accessed with special tags

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[loc(v):]        # Slices can be in data coordinates, too
h2 = h[::rebin(2)]     # Modification operations (rebin)
h2 = h[a:b:rebin(2)]   # Modifications can combine with slices
h2 = h[::sum]          # Projection operations
h2 = h[a:b:sum]        # Adding endpoints to projection operations
h2 = h[0:len:sum]      #   removes under or overflow from the calculation
h2 = h[v, a:b]         #   A single value v is like v:v+1:sum
h2 = h[a:b, ...]       # Ellipsis work just like normal numpy

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

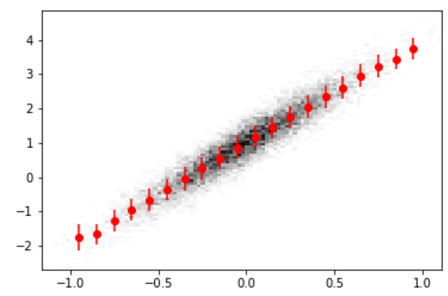
Analysis using axes

What traditionally would be multiple histograms can be described as axes in a single histogram!

```
value_ax = bh.axis.Regular(100, -5, 5)
bool_ax = bh.axis.Integer(0, 2,
                          underflow=False,
                          overflow=False)
run_number_ax = bh.axis.IntCategory([], growth=True)

hist = bh.Histogram(value_ax, bool_ax, run_number_ax)

hist.fill(values, bools, run_numbers)

hist_true = hist[:, True, ::bh.sum] # Classic 1D hist
```



Log transform axis example



Profile histogram example

The future plans

Boost-histogram is ready for broad use; final polishing work is being down to enable smooth behavior when mixing types, etc. Boost-histogram has a well defined scope; it does not plot histograms or convert them; it has no dependencies. Aghast handles conversions, and Hist will assist in plotting and other common analysis tasks.

This project is supported by the National Science Foundation under Cooperative Agreement OAC-1836650 (IRIS-HEP) and grant OAC-1450377 (DIANA/HEP). Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.