A universal need

Histograms are a universal tool used across disciplines. However, for HEP, rather than 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 physicist 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 fell 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.

Core histogramming libraries

- Scikit-HEP
- HistBook
- coffea
- physt
- Vex

Applications

- PyROOT
- ROOT
- mplhep
- coffeepy
- boost-histogram
- ROOT

Team: Henry Schreiner, Hans Dembinski*, Jim Pivarski

Institutions: Princeton University, *TU Dortmund

We have some of the most highly developed histogram tools in C++ in ROOT, and why we need a high quality Python histogramming package.

### 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[under/overflow];` # Underflow and overflow can be accessed with special tags
- `v = h[static/dynamic];` # Slice over everything
- `v = h[...];` # Slice of histogram (includes flow bins)
- `v = h[loc(v)];` # Slices can be in data coordinates, too
- `v = h[redim(b)];` # Modification operations (rebin)
- `v = h[redim(b)];` # Modifications can combine with slices
- `v = h[...];` # Projection operations
- `v = h[...];` # Adding endpoints to projection operations
- `h = [v];` # A single value v is like v:v:v
- `h = [v, a:b];` # Ellipsis’s work just like normal numpy

### 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.Bool(2, underflowFalse, overflowFalse)
run_number_ax = bh.axis.IntCategory([], growth=True)

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

hist.fill([0.5, 1.0], [True], [100])
```

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.