A Lightweight Histogramming Interface Layer

C. Leggett[1]

Lawrence Berkeley National Laboratory, USA

Abstract

A troubling theme that pervades modern data analysis tools available in the world of high energy and nuclear physics today, is that the user is usually constrained to a specific platform and file format. PAW, ROOT, and HepTuple are all culprits in this regard. Furthermore, modern tools such as ROOT, JAS and OpenScientist seek to blend the lines between data analysis tools, and visualization tools, which often should be kept discrete, requiring the use of large libraries.

We present a statistical data interface layer, which treats histograms and ntuples as discrete, lightweight objects, allowing simple manipulations, and the ability to either save the objects in traditional file formats such as HBOOK or ROOT, or to pass them on to applications. This functionality is provided by a class library, that can be linked to existing code with minimal modifications, enabling the histogramming of virtually any data type.

Keywords: histogram,lightweight,template,C++

1 Introduction

Histograms have become one of the primary tools used in the analysis of data from HENP experiments. They are so deeply entrenched in our mindset that it is difficult to conceive of an analysis that does not make use of them. As the software developed to aid in the analysis of HENP data has evolved over the decades, histogramming tools have become almost inextricably intermingled in all the various levels of this software, from the point of data collection, through the analysis modules, all the way to the visualization programmes. They have gained functionality and features which transcend their underlying nature as purely statistical entities, to the point where they are no longer suitable for certain applications.

It should be remembered that at a fundamental level, histograms are entities that hold statistical information about a particular process. They should not be concerned with the thickness of the line with which they are viewed, or the parameters of a fit. The act of accumulating and storing statistical data is completely distinct from the act of manipulating it and from the act of visualizing it. In the attempt to provide vast functionality, modern tools have tended to blur the lines between these various actions, making it very difficult to use histograms as containers of purely statistical information. Even with the use of shared libraries, linking in current histogramming packages can result in very large memory footprints, which may overtax critical components.

Furthermore, many histogramming packages restrict the user to a single type of file format. If that format is used, then the associated visualization tool must also be used. While some conversion utilities do exist, they are often unidirectional, and don't permit the user to freely chose between competing formats.

We have designed a C++ class library that deals with histograms at their fundamental level. These histograms can be booked, and filled, in both binned and unbinned variants, and minor

operations such as addition and subtraction have been provided. No attempt has been made to provide a visualization tool, or complicated manipulation methods. The result is a very lightweight histogram object with minimal overhead that can be passed between various modules, and with the use of a provided file manager class, saved to and read from disk in a variety formats. This will also allow the package to be used as interchange medium, to convert between various histogram file formats.

2 Design Concerns

The two aspects of the design that we felt to be most essential were ease of use for the end user, and minimal resource overhead. It should not be necessary for the user to write ten or twenty lines of code just to create and fill a histogram. Sensible defaults are used for all parameters, though they can be overridden as desired. This resulted in the implementation of an automatically binned histogram, where the user does not even need to specify the range or bin sizes of the histogram - they are computed on the fly after sufficient statistics have been collected. As well, in order to minimize the size of the histogrammed object, the internal statistical information can be kept as either floats or doubles, with the choice being made on a histogram by histogram basis.

One of the questions that we confronted in the design was the use of templates. Even today, C++ compilers sometimes have trouble dealing with templates. However, it was felt that their benefits outweighed their problems, and a decision was made to use them. This permitted us to implement a very useful feature in the histogramming class - the ability to histogram complicated objects, instead of merely simple types such as ints, floats, and doubles. In order to make use of this feature, the user must supply a function that quantifies the object to be histogrammed.

3 Usage

Three standard types of histogram binning have been implemented: BINNED, UNBINNED and AUTO.

- **BINNED** histograms can either be created with fixed bin widths, or an array of bin edges can be supplied for variable width bins.
- **UNBINNED** histograms are intended for only small samples, and can be converted to BINNED histograms at any time.
- **AUTO**matically binned histograms start out as UNBINNED histograms, and are automatically converted to BINNED histograms after sufficient statistics have been accumulated. The conversion either creates fixed bin width histograms, or calculates the bin edges such that the same number of entries fill each bin.

  When the histograms are filled, a weight for each entry can also specified.

  Various examples of creating histograms are shown below:

```
#include "Histogram.hpp"

Histogram hist1<>;              // An automatically binned histogram
                                // of floats

Histogram hist2<>(100, -10., 20.);   // 100 bins of fixed width between
                                     // -10 and 20
```

```
Histogram hist3<float,double> (xvec); // variable bin widths supplied
                                      // by xvec, statistics kept as
                                      // doubles

// Automatically binned histogram of Muon objects, ordered according
// to the function (Muon.px() - Muon.py())

class Muon;
float MyQuantFunction(const Muon &M) {
  return ( M.px()-M.py() );
}

Histogram hist4<Muon>;
hist4.SetQuantFunction( MyQuantFunction );


// Filling

hist.Fill(X);        // fill histogram with X
hist.Fill(Y,.3);     // fill with weight 0.3

Muon M;
hist.Fill(M);        // fill with muon object, according to previously
                     // defined quantization function


// Various other methods
hist.Rebin(nch);                        // Rebin histogram with nch bins
hist.Resize(xlo, xhi, nch);             // Resize histogram

X = hist.Contents(ichan);               // Bin contents of channel ichan
C = hist.BinCenter(ichan);              // Center of channel ichan

n = hist.Entries();                     // Equivalent entries
m = hist.Overflow();                    // Number of overflow entries

M = hist.Mean();                        // Mean of histogram
R = hist.RMS();                         // RMS of histogram
```

4 I/O

In order to completely separate the histogram object from the persistent storage format, the histogram object knows nothing about histogram file storage formats, such as ROOT or HBOOK. Instead a histogram file manager is used to read in histograms from disk, and write them out in various formats. This also minimizes the histogram class size, as no extraneous libraries need be linked, reducing the required resources. Since the file manager can both read and write to all the file formats, this also provides the ability to convert histograms between any two file formats.

At this time, only a limited selection of file format flavours has been implemented, namely HBOOK, ROOT, and XDR. It is expected that this list will grow as time goes progresses to include all the standard histogram formats.

5 Conclusions

With this histogramming package, we have completely separated the histogram object from all the non-statistical aspects that have become inextricably intermingled in most other histogramming tools. This provides a clear delineation between data gathering, analysis and visualization tools. Not only does this permit computational resources to be minimized, but also facilitates the use of any desired tool at each level of the analysis process - for example the user is no longer constrained to use a particular visualization tool if its associated file format is chosen. This results in greater modularization of the code, and allows for various components to be replaced as they become obsolete or fall out of favour.

Since the file manager is distinct from the histogram layer, it can easily be amended as new formats are developed. Likewise, accessory packages can be developed by users for specific needs, such as specialized fitting or statistics routines. Since they are separate components, they would only be linked in as needed, keeping the resource usage to a minimum.