Working with Histograms

ROOT Training
Working with Histograms

• We have seen:
  – what is a histogram;
  – how to create a one-dimensional histogram;
  – how to draw the histogram

• We will see and work on with the exercises
  – how to extract information from an histogram
  – how to manipulate histograms
  – histograms in multi-dimension
  – what is a profile histogram
  – weighted histograms
  – sparse histograms

• More detailed on the ROOT Graphics Pad and the Canvas
• TH1 is the base class for all Histogram classes.
• TH1, TH2, TH3 are generic. Specialized ones (e.g. TH1D) should be used for constructing the objects.
  – Most used classes are TH1(2,3)F and TH1(2,3)D.
• For non-equidistant bins use a separate constructor:

```cpp
TH1D * h1 = new TH1D("h1","Example histogram",nbins,xbins);
```

• `xbins`: array of (nbins+1) values with the bin-edges
Axis and Bins

- The histogram class has an axis class which contains the bins

```cpp
TAxis * axis = h1->GetXaxis();
```
- one can query the number of bins, lower/upper bin edge from the axis:

```cpp
axis->GetNbins();
axis->GetBinLowEdge(bin_number);
axis->GetBinCenter(bin_number);
axis->GetBinUpEdge(bin_number);
```
- for the corresponding bin number given x value use

```cpp
bin_number = axis->FindBin(x_value);
```
- one can set the axis range (e.g. for zooming the histogram)

```cpp
axis->SetRange(firstbin, lastbin);
```
- N.B.: axis bin number starts from 1
  - bin number 0 is the **Underflowbin**
  - bin number nbin+1 is the **Overflowbin**
Histogram Bin Content and Error

• To extract content and error from bin number \( ibin \) of the histogram \( h1 \):

  ```
  root [1] h1->GetBinContent(ibin)
  (const Double_t)1.10000000000000000e+01
  root [2] h1->GetBinError(ibin)
  (const Double_t)3.31662479035539981e+00
  ```

• By default histograms have a bin error equal to \( \sqrt{N} \), where \( N \) is the bin content.
  – It is assumed that the observed bin content follows a Poisson distribution.
  – Can also display Poisson confidence interval

  ```
  root [1] h1->SetBinErrorOption(TH1::kBinningOption)
  root [2] h1->GetBinErrorUp(ibin);
  root [3] h1->GetBinErrorLow(ibin);
  ```
Histogram Operations

• Histogram scaling (normalization)
  – useful for plotting histogram in the same pad
  – useful for seeing histogram as an estimate of a probability density function (PDF)

```cpp
scale = 1.0/h->Integral();
h1->Scale(scale);
```

N.B. : After scaling the error will not be correct. We will see later how to have correct errors
Histogram Operations

• Add Histograms:
  – merge two histograms which have same axis:

```c
TH1D *h3 = new TH1D("h3","h1+h2",nbin,xmin,xmax);
h3->Add(h1,h2);
```

  – can also be used to subtract histograms

```c
h3->Add(h1,h2,1.,-1.);
```

• Divide Histograms:

```c
TH1D *h3 = new TH1D("h3","h1/h2",nbin,xmin,xmax);
h3->Divide(h1,h2);
```

  – if h1 is a subset of h2, the bin content of h3 is binomially distributed

  • ➤ better to use **TEfficiency** class
Weighted Histograms

- Histogram can be filled with a “weight”
  - observations do not contribute equally, but some of them contribute more or less than others;
  - the weight expresses how much an observation contributes.
    - Filling a 1D histogram with observation x and weight w:
      ```cpp
      h1->Fill(x,w);
      ```
    - Filling a 2D histogram with observation x,y and weight w:
      ```cpp
      h1->Fill(x,y,w);
      ```

- A weighted histogram will have and display as:
  - bin content = sum of all the weights accumulated in the bin;
  - bin error = \( \sqrt{W^2} \): \( W^2 \) = sum of the weight square in the bin.
    (in ROOT versions <= 5.34, if one has called TH1::Sumw2() before filling the histogram)
Histogram Re-Binning

- Rebin: merge bins together.
  - Possible to merge adjacent bins in a given group,
    - e.g. an histogram with 100 bins can be re-binned in a histogram with 25 bins.
  - Possible to merge bins using new bin edges provided by the user, i.e. making a new a variable bin histograms.
    - the new bin edges must correspond to existing bin edges. It is not possible to split bins.
  - Histogram errors and statistics are re-computed according to the new binning.

```c
TH1 * hrb = h1->Rebin(4,"hrebinned");
```

Rebin original histogram (40 bins) in a new one with 10 bins grouping 4 bins together (ngroup =4)
2D Histograms

- Frequency distribution of (X,Y) observations.
  - Construct 2D histogram specifying the number of bins in X axis, the minimum and maximum of axis range and the same for the Y axis

```c
TH2D * h2 = new TH2D("h2","Example 2D Hist",40,-4.,4.,40,-4.,4);
```

- fill 2D histogram with 10000 x,y normal data

```c
for (int i = 0; i<10000; ++i) {
    double x = gRandom->Gaus(0,1);
    double y = gRandom->Gaus(1,2);
    h2->Fill(x,y);
}
```

- use `h2->Draw()` for drawing, several options are available
  - Color, Contour, lego, surface and box plots
Drawing 2D histograms

• Color plots:

```cpp
h2->Draw("COLZ");
```

“Z” means drawing the color palette for the bin content (i.e. the Z axis)

• Contour plots:

```cpp
h2->Draw("CONTZ");
```

• Lego plots:

```cpp
h2->Draw("LEGO");
```
Profile Histograms

• A 2D Histogram can be projected into a 1D histogram

\[
\text{TH1} \ast \text{hX} = \text{h2} \rightarrow \text{ProjectionX}();
\]

– when projecting on the X axis the bins with the same x value are all summed together.

• A Profile histogram is a different way of projecting 2D data.

– for each bin in x the sample mean of the y values is plotted

\[
\text{TProfile} \ast \text{hp} = \text{new} \\
\text{TProfile}(\text{“hp”,“hp”,nbin,xmin,xmax}); \\
\text{for} \ (\text{int} \ i = 0; \ i<10000; \ ++i) \ { \\
\quad \text{double} \ x = \text{gRandom} \rightarrow \text{Gaus}(0,1); \\
\quad \text{double} \ y = \text{gRandom} \rightarrow \text{Gaus}(1,2); \\
\quad \text{hp} \rightarrow \text{Fill}(x,y); \\
\}
\]

hp->Draw();

– various options for displaying errors
– default uses error in the sample mean (RMS/\sqrt{N})
Generating Pseudo Data

- Random number generation in ROOT is done using the TRandom classes
  - Three pseudo-random number generator exists, TRandom1, TRandom2 and TRandom3. TRandom is the base class.
    - TRandom3 (a Mersenne-Twister generator) is used by default (it has a very long period, $\sim 10^{6000}$ and it is very fast).
  - Seeding is controlled using TRandom::SetSeed( seed).
    - When using seed = 0, independent random streams can be generated (the seed is based on a UUID number).
  - TRandom::Rndm() generates uniform number in the [0,1] range.
  - Random numbers can be also generated using the global static variable gRandom

```c
root [0] gRandom->Rndm()
(Double_t)9.99741748906672001e-01
```
Random Number Distributions

- The class `TRandom` provides methods to generate numbers according to some pre-defined distributions

<table>
<thead>
<tr>
<th>Distributions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Double_t Uniform(Double_t x1,Double_t x2)</code></td>
<td>Uniform random numbers between (x_1, x_2)</td>
</tr>
<tr>
<td><code>Double_t Gaus(Double_t mu,Double_t sigma)</code></td>
<td>Gaussian random numbers.</td>
</tr>
<tr>
<td></td>
<td>Default values: (\mu=0, \sigma=1)</td>
</tr>
<tr>
<td><code>Double_t Exp(Double_t tau)</code></td>
<td>Exponential random numbers with mean tau.</td>
</tr>
<tr>
<td><code>Double_t Landau(Double_t mean,Double_t sigma)</code></td>
<td>Landau distributed random numbers.</td>
</tr>
<tr>
<td></td>
<td>Default values: (mean=0, \sigma=1)</td>
</tr>
<tr>
<td><code>Double_t BreitWigner(Double_t mean,Double_t gamma)</code></td>
<td>Breit-Wigner distributed random numbers.</td>
</tr>
<tr>
<td></td>
<td>Default values mean=0, gamma=1</td>
</tr>
<tr>
<td><code>Int_t Poisson(Double_t mean)</code></td>
<td>Poisson random numbers</td>
</tr>
<tr>
<td><code>Double_t PoissonD(Double_t mean)</code></td>
<td></td>
</tr>
<tr>
<td><code>Int_t Binomial(Int_t ntot,Double_t prob)</code></td>
<td>Binomial Random numbers</td>
</tr>
<tr>
<td><code>Circle(Double_t &amp;x,Double_t &amp;y,Double_t r)</code></td>
<td>Generate a random 2D point ((x, y)) in</td>
</tr>
<tr>
<td></td>
<td>a circle of radius (r)</td>
</tr>
<tr>
<td><code>Sphere(Double_t &amp;x,Double_t &amp;y,Double_t &amp;z,Double_t r)</code></td>
<td>Generate a random 3D point ((x, y, z)) in</td>
</tr>
<tr>
<td></td>
<td>a sphere of radius (r)</td>
</tr>
<tr>
<td><code>Rannor(Double_t &amp;a,Double_t &amp;b)</code></td>
<td>Generate a pair of Gaussian random numbers with</td>
</tr>
<tr>
<td></td>
<td>(\mu=0) and (\sigma=1)</td>
</tr>
</tbody>
</table>
Random Number Distributions (2)

- Random numbers can be generated according to what-ever distribution using accept-rejection techniques (often not very efficient) or by using the inverse of the cumulative (integral) distribution

- Integral \( g(x) \) of distribution \( f(x) \) (probability density function "PDF"):
  - and inverse \( i(y) \) of \( g(x) \)
  - random number \( 0 < y^* < 1 \)
  - return \( i(y^*) \)
  - \( i(y^*) \) is distributed like \( f(x) \)

- ROOT has the method \( \text{TF1::GetRandom}() \), which uses this technique to generate random numbers from a generic function object
Time for Exercises!

Put in practice the concepts to which you were just exposed: read the instructions and solve the histogram exercises.

see https://twiki.cern.ch/twiki/bin/view/Main/ROOTLaPlataTutorial#Working_with_Histograms
The ROOT graphics is build around the *Graphics Pad* concept (class `TPad`). A Graphics Pad is a linked list of primitives of any type (graphs, histograms, shapes, tracks, etc.). It is a kind of *display list* as shown on the following picture:

Adding an element into a Graphics Pad is done by the `Draw()` method of each classes.

On the previous picture the `Draw()` method has been called on: a 1D histogram, a 2D histogram, a graph and finally a pie chart.

All these objects are now stored in the Graphics Pad’s display list.
A Graphics Pad is painted by calling sequentially the `Paint()` method of each object in the list of primitives, as shown on the following picture:

The Graphics Pad's (re)painting does not need to be done explicitly by the ROOT user. It is done automatically at the end of a macro execution or when a Graphics Pad has been modified.

In some cases a pad need to be painted during a macro execution. To force the pad painting `gPad->Update()` should be performed.
The Graphics Canvas

A canvas is the graphics window in which the ROOT graphics will be displayed. It can be created using the class TCanvas. One can create as many canvases as needed during a ROOT session.

A TCanvas usually contains at least one TPad. Most of the time it contains several, each of them having its own coordinate system. A simple way to quickly make several pads in a canvas is to use the method Divide() like in the following example:

```c
TCanvas *c = new TCanvas("c","my canvas", 600,400);
c->Divide(2,1);
c->cd(1);
hpx->Draw();
c->cd(2);
hpxpy->Draw("lego");
c->cd(1);
TText *T = new TText(-1.,400.,"Hello !");
T->Draw();
```
Summary

• We have learned more about ROOT Histograms
  – how to access their information
  – how to perform operations on histograms
• Multi-dimensional histograms and projections
  – what is a profile histogram
• Remember:
  – all graphics options for plotting all histogram types are documented in the \texttt{THistPainter} class:
    • \url{http://root.cern.ch/root/html/THistPainter.html}