PHYS511L Lab 5: Attenuation Length

Spring 2016

1 Introduction

This lab walks through calculations used in testing and assessing a time of flight scintillator counter in addition to showing more capabilities of ROOT for visualizing and computing with higher dimensional data. The quantity we are concerned with is the *attenuation length* of the counter. The counter is a bar of scintillating material with a photomultiplier tube (PMT) attached to each end. Whenever a particle interacts with the counter and creates photons in the counter, the position of the origin of this light can be determined by using the simultaneous time information gathered by the PMTs once the effective speed of light c_{eff} is known. This calculation combined with very precise clocks allows the measurement of the time and location at which a particle interacts with the counter. Time-of-flight (TOF) detectors are built by assembling arrays of these counters, creating a kind of net for tracking outgoing particles from a scattering reaction or other process.

However, not all materials are equally-well suited to be used for TOF counters. One such measure of the quality of a TOF counter is the previously mentioned attenuation length. The counter material causes a loss of photons which can be seen by the PMTs depending on the distance the source is from each PMT. If N is the number of photons that reach a given PMT and D is the distance the source is from the PMT surface, then

$$N = N_0 e^{-D/\lambda} \tag{1}$$

where N_0 is proportional to the total number of photons generated during the scintillation and λ is the attenuation length. Therefore, if we had data of N versus D for a given PMT we could determine the attenuation length via fitting this data with an exponential function. The first step to doing this is learning how to find the interaction position for a particle passing through the counter.

1.1 Finding the Source

Let's say that a single particle has passed through the counter vertically in a straight line, causing enough light to be detected by both PMTs. Letting L be the length of the counter and assuming an x-axis along the counter with zero being in the center of the counter, if the position of the interaction were given by x and the time of this interaction were given by t, then the times at which the PMTs would detect the light would be given by

$$t_L = t + \frac{L/2 + x}{c_{\text{eff}}} \tag{2}$$

and

$$t_R = t + \frac{L/2 - x}{c_{\text{eff}}} \tag{3}$$

where t_L and t_R are the detection times for the left and right PMTs respectively. Notice that if we calculate

$$\frac{\Delta t}{2} = \frac{t_L - t_R}{2} = \frac{x}{c_{\text{eff}}} \tag{4}$$

then we can find the position of the light source through

$$x = c_{\text{eff}} \frac{\Delta t}{2} = c_{\text{eff}} \frac{t_L - t_R}{2}. \tag{5}$$

1.2 Effective Speed of Light

To calculate c_{eff}, we make use of the above source-finding technique in addition to the known length of the bar and the distribution of $\Delta t/2$. Ideally, $\Delta t/2$ should have a uniform distribution with clear boundaries for the minimum and maximum $\Delta t/2$ values. If these boundaries were given by $\Delta t_{\text{min}}/2$ and $\Delta t_{\text{max}}/2$, then c_{eff} could be computed via

$$c_{\text{ceff}} = \frac{L}{\Delta t_{\text{max}}/2 - \Delta t_{\text{min}}/2} \tag{6}$$

where again L is the counter length. However, due to many factors in the measurement process the $\Delta t/2$ distribution is convoluted and has tails. Figure 1 shows an example $\Delta t/2$ distribution.

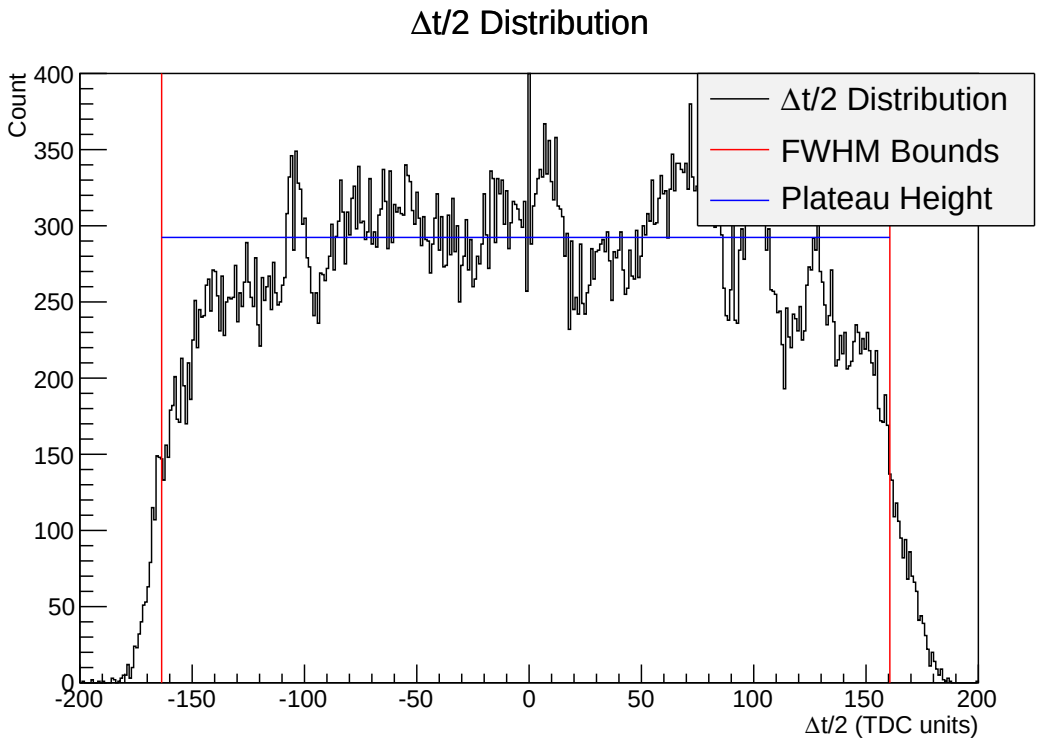

Figure 1: Δt distribution with FWHM bounds and plateau height.

The plateau behavior in the center is still visible, and in fig. 1, the height of this plateau is denoted by the blue line. Due to the tails, the actual minimum and maximum Δt values do not make a good estimate of the range of the distribution. Instead, we can use the full width at half maximum (FWHM) of the distribution as an estimator for $\Delta t_{\max}/2 - \Delta t_{\min}/2$, resulting in

$$c_{\text{eff}} = \frac{L}{\text{FWHM}}. \tag{7}$$

1.3 Determining the Attenuation Length

In principle, determining the attenuation length would be as simple as finding an appropriate λ through fitting N versus x for each PMT, and ideally each PMT should give the same result. To measure N, an analog to digital converter (ADC) is used to integrate the current through a PMT's anode. This charge is proportional to N, and can be used in place of N for the exercises that follow. However, as shown in fig. 2 and fig. 3, the number of photons N is statistically distributed for each position x.

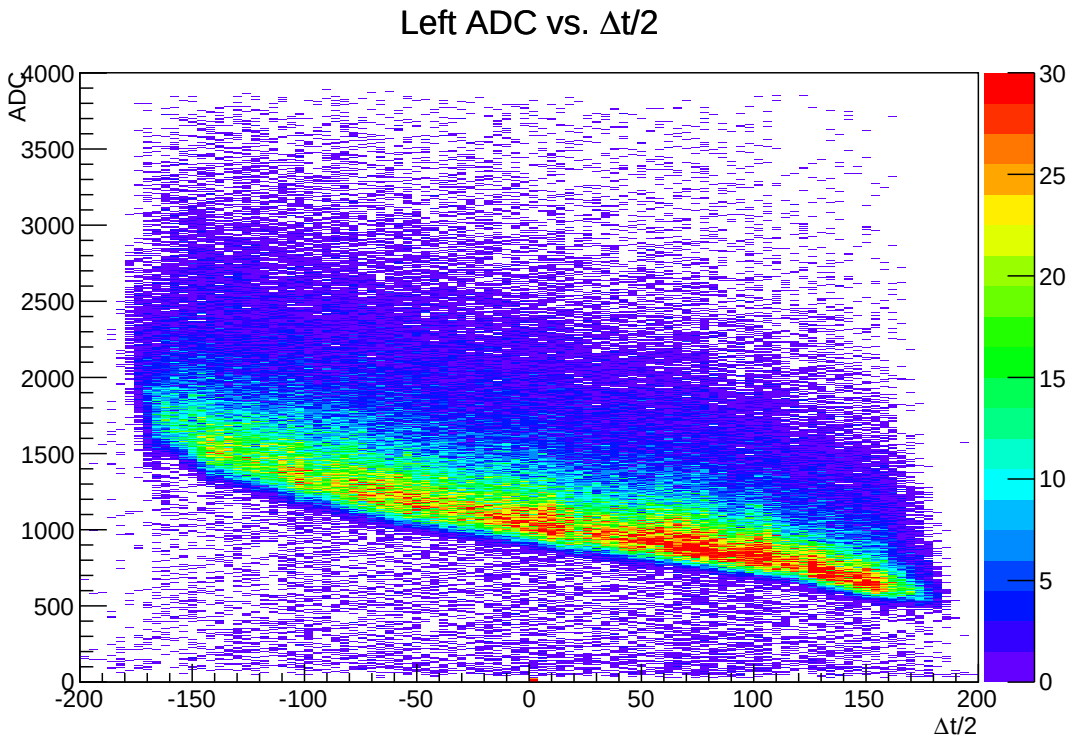

Figure 2: ADC channel versus Δt distribution. ADC channel is proportional to N. Notice that this is not a perfectly thin curve and therefore that N is statistically distributed for any given position along the counter.

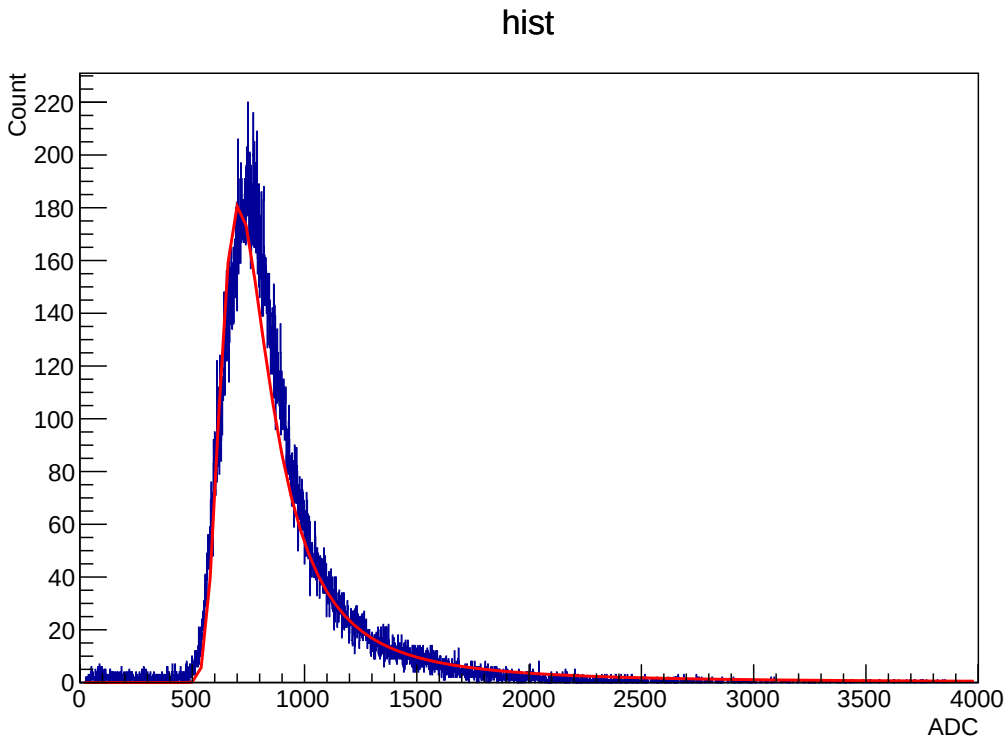

Figure 3: ADC channel distribution for 40 TDC unit bin around $\Delta t/2 = 80$ TDC units. Red line is the Landau fit to this ADC distribution.

In practice, the number of photons that are detected by a PMT from a given position is distributed according to the Landau distribution,

$$\rho_{\text{Landau}}(N) = \frac{1}{\pi}\int_0^\infty e^{-t\log t - xt}\sin(\pi t)dt, \tag{8}$$

which is well approximated by

$$\rho_{\text{Landau}}(N) \approx \frac{1}{2\pi}\exp\left(-\frac{1}{2}\left(x + e^{-x}\right)\right). \tag{9}$$

(ROOT comes with this function built-in.) The Landau distribution is strange in that it does not actually have a mean, but by defining a scaled (A), shifted (m), and stretched (σ) version as

$$\tilde{\rho}_{\text{Landau}}(x; A, m, \sigma) = A\rho_{\text{Landau}}((x-m)/\sigma), \tag{10}$$

the m parameter can be used as the central or characteristic value. (In ROOT nomenclature m is the "MPV" parameter.) Therefore, by collecting N distributions from specific positions along the counter, fitting each with a Landau distribution, extracting m, and using m for the estimate of N, the N versus x plot can be obtained and then fitted against an exponential.

Without infinite statistics, it is not possible to collect ADC distributions from a single point, so we need to divide the counter into subsections which are small enough to allow a good Landau fit but large enough to allow enough statistics in the individual ADC distribution. As stated below in the instructions, cutting $\Delta t/2$ into 40 TDC unit increments is appropriate for the data provided to you.

2 Lab Tasks

- Retrieve previously collected data from `http://boson.physics.sc.edu/~gothe/511-S16/rootlab/PHYS511L-S16/attlen.root`. The tree name is "attlen", and the branches are "adc_l" for the left ADC, "adc_r" for the right ADC, "tdc_l" for the left TDC, and "tdc_r" for the right TDC.
- For each PMT:
  - Calculate the effective speed of light (c_{eff}) from each PMT's perspective.
  - Using a cut width of 40 TDC units, plot the central ADC value m versus the position x along the counter on top of a plot of the 2-D histogram of m and x. Make sure not to overlap the cut ranges; you should cut in ranges like [-400,-360],[-360,-320],...
  - Include sample ADC distribution plots from the 40 TDC unit cuts (at least one from each PMT) along with the Landau fits in your lab report.
  - Calculate the attenuation length from each PMT's perspective. Make sure to use the appropriate effective speed of light in each case.
  - Plot the exponential fits you used for the attenuation length calculation in the same m versus x plot above.
- Using the average of the two attenuation length estimates as the attenuation length for the counter, compare the attenuation length with the actual length of the counter. Does this make physical sense? How should a good counter's attenuation length compare to its actual length? What about a bad counter's attenuation length?

3 Helpful Information

- TDC units are 25 ps. Don't forget to convert your c_{eff} units using this information!
- The counter length L for the data in this experiment is 120 cm.
- 2-D histograms are very similar to 1-D histograms in terms of how you use them. The example code below shows how:

```
//Initialization takes the following form:
TH2D hist("name","title",
          X_nbins,X_low,X_high,
          Y_nbins,Y_low,Y_high);

//So, for example,
TH2D* hist = new TH2D("adc_hist","adc_hist",
                      //delta t/2
                      100,-200,200,
                      //ADC
                      2001,-0.5,4000.5);
//would give you an appropriately binned
//ADC vs. delta t/2 histogram
```

```
//To add data to this histogram, you still use
//the Fill function, but with an extra argument
//since you now fill 2 values at the same time
//instead of 1:
hist.Fill(x,y);

//Once you've filled your histogram, you can
//draw it, but you need to supply an extra
//argument to Draw so that it knows you want to
//view it on a 2-D plane instead of in 3-D
//space:
hist.Draw("colz");

//In addition to this, sometimes a histogram has
//strong peaks which make it difficult to see what
//we're interested in (which is the case for this
//lab's data).  In these cases, it makes sense to
//adjust the z-axis of the histogram:
hist.GetZaxis()->SetRangeUser(0,30);

//which would set the z-axis range to [0,30].
```

- Using ROOT to process histograms is not difficult once you know the ins and outs of the library, but for the sake of expediency, here are some functions to help you with calculating the FWHM and the peak height of the $\Delta t/2$ distribution:

```
//Finds average y value over peak plateau region.  Avoids the
//non-physical peak at deltat = 0
double deltat_peak_y(TH1D* hist,
                     double xlo=-50,
                     double xhi=50)
{
  double npoints = 0;
  double sum = 0;
  double xpeak = 0;
  double peak = -1;
  long nbins = hist->GetNbinsX();
  for(int bin = 1; bin < nbins; ++bin) {
    double x = hist->GetXaxis()->GetBinCenter(bin);
    if(// avoid 0 bin
       !(-1e-5 < x && x < 1e-5) &&
       //Stay inside plateau
       x > xlo &&
       x < xhi) {
      double content = hist->GetBinContent(bin);
```

```
      sum += content;
      npoints++;
    }
  }
  return sum/npoints;
}

//Finds the left x value for the half-peak.
double deltat_halfpeak_left_x(TH1D* hist)
{
  // can't be positive so 1 makes a good default
  double xleft = 1;
  double left_halfpeak = 0;
  double mindiff = 1e9;
  const double halfpeak = 0.5*deltat_peak_y(hist);
  long nbins = hist->GetNbinsX();
  //Traverse from left to right
  for(int bin = 1; bin <= nbins; ++bin) {
    double x = hist->GetXaxis()->GetBinCenter(bin);
    if(x < -1e-5) { // stay left of 0 bin
      double content = hist->GetBinContent(bin);
      double diff = abs(content - halfpeak);
      if(mindiff > diff) {
        mindiff = diff;
        left_halfpeak = content;
        xleft = x;
      }
    }
  }
  cout << "Left Half-Peak: (x=" << xleft
       << ",y=" << left_halfpeak << ")"
       << endl;
  return xleft;
}

//Finds the right x value for the half-peak.
double deltat_halfpeak_right_x(TH1D* hist)
{
  // can't be negative so -1 makes a good default
  double xright = -1;
  double right_halfpeak = 0;
  double mindiff = 1e9;
  const double halfpeak = 0.5*deltat_peak_y(hist);
  long nbins = hist->GetNbinsX();
  //Traverse from right to left
```

```
  for(int bin = nbins; bin >= 1; --bin) {
    double x = hist->GetXaxis()->GetBinCenter(bin);
    if(1e-5 < x) { // stay right of 0 bin
      double content = hist->GetBinContent(bin);
      double diff = abs(content - halfpeak);
      if(mindiff > diff) {
        mindiff = diff;
        right_halfpeak = content;
        xright = x;
      }
    }
  }
  cout << "Right Half-Peak: (x=" << xright
       << ",y=" << right_halfpeak << ")"
       << endl;
  return xright;
}
```

- Due to the large number of points you need to sample the ADC m value at, it is best to define a function for calculating the ADC histogram for each cut; for example,

```
TH1D* pointcut_left_adc(TChain* tree,
                        // center of delta t / 2 cut
                        double cut_center,
                        // width of delta t / 2 cut
                        double cut_width)
{
  double cut_low = cut_center - 0.5*cut_width;
  double cut_high = cut_center + 0.5*cut_width;

  double adc_l;
  tree->SetBranchAddress("adc_l",&adc_l);
  double tdc_l;
  tree->SetBranchAddress("tdc_l",&tdc_l);
  double tdc_r;
  tree->SetBranchAddress("tdc_r",&tdc_r);

  TH1D* hist = new TH1D("hist","hist",4001,-0.5,4000.5);

  int nrows = tree->GetEntries();
  for(int i = 0; i < nrows; i++) {
    tree->GetEvent(i);
    double deltat_over_2 = 0.5*(tdc_l - tdc_r);
    if((cut_low <= deltat_over_2) &&
       (cut_high <= deltat_over_2)) {
      hist->Fill(adc_l);
```

```
    }
  }
  return hist;
}
```

You can copy and modify this function to have another function for cutting on the right ADC values.

- In the same spirit as the other labs, here is a function for fitting a histogram against a Landau distribution:

```
// Landau fit function.  Returns the central (MPV) value
double fit_landau(TH1D* hist)
{
  TF1* landau
    = new TF1("landau",
              "[0]*TMath::Landau(x,[1],[2])",
              0.5, //to protect against non-physical peak
              4000);

  //Guess at normalization
  landau->SetParameter(0,hist->GetMaximum());
  //Guess at MPV:
  landau->SetParameter(1,1000);
  //and sigma:
  landau->SetParameter(2,1);
  hist->Fit(landau);
  return landau->GetParameter(1);
}
```

I'll leave it to you to define one for the exponential function.

- A data type that would be extremely useful to your tasks is the TGraph. TGraphs support scatter plots, and can be fitted and drawn in the same way as histograms. The following code shows how to use a TGraph:

```
//A graph is just a pair of arrays, one
//for the X values and one for the Y
//values.  So, if we had data in these:

double xs[5] = {1., 2., 3., 4., 5.};
double ys[5] = {1., 4., 9., 16., 25.};

//we could create a TGraph like this:
TGraph graph(5,xs,ys);

//and then draw it.  Note the special draw
//options "A*" for plotting as points
```

```
graph->Draw("A*");

//or fit it with a parabola
graph->Fit("pol2");
```

So, if you were to collect your N versus m data in a graph you could directly fit it against the exponential and plot it on the 2-D histogram plot.

- As a last bit of help, we needed a lot of headers for this lab's code, so here is the header code I needed when doing this lab:

```
#include <TMath.h>
#include <TLegend.h>
#include <TCanvas.h>
#include <TLine.h>
#include <TAxis.h>
#include <TTree.h>
#include <TChain.h>
#include <TFile.h>
#include <TH1D.h>
#include <TH2D.h>
#include <TF1.h>
#include <cmath>
#include <string>
#include <iostream>
using namespace std;
```