Analyzing-Trees-Section2

June 26, 2018

0.1 Plotting from the command line: TTree::Draw

`TTree::Draw` is the universal command to plot tree data. You might call it the Swiss Army Knife of interactive analysis with ROOT. It's signature is effectively

```
TTree::Draw(const char *varexp, const char *selection="", Option_t *option="", Long64_t nentries
```

Here * `varexp` is basically a formula of what you'd like to plot. In the simplest case, it is just the name of a tree variable. * `selection` is an expression that must be true (unequal zero) for `varexp` to be drawn. * `option` is a drawing option for the output histogram or scatter plot * `nentries` indicates how many entries (events) to process * `firstentry` is the number of the first entry to process

For a comprehensive description and many more options, see the documentation at https://root.cern.ch/doc/v612/classTTree.html

Let's run a handful of examples with real Hall A data.

Before doing that, run the following cell. This so-called "jsroot magic" is useful for making ROOT plots more interactive in a notebook:

In [ ]:
```
//%jsroot on
```

Create a canvas object where our plots will appear. We only have to do that in a notebook session so that we can explicitly Draw() the canvas later.

In [1]:
```
TCanvas c1;
```

Let's open a different ROOT file this time that, in addition to what we had before, also contains all the variables from plane u1 of the vertical drift chambers (VDCs)

In [2]:
```
f = TFile::Open("/data/ROOTfiles/g2p_3132_vdc_u1.root","READ");
```

```
Warning in <TClass::Init>: no dictionary for class THaEvent is available
Warning in <TClass::Init>: no dictionary for class THaEventHeader is available
Warning in <TClass::Init>: no dictionary for class THaRun is available
Warning in <TClass::Init>: no dictionary for class THaCodaRun is available
Warning in <TClass::Init>: no dictionary for class THaRunBase is available
Warning in <TClass::Init>: no dictionary for class THaRunParameters is available
```

Again, please ignore the warnings about "no dictionary for class"; this is an artifact of using plain ROOT. (One day, we might have a Notebook interface for the Hall A analyzer.)

As with all of the Hall A and Hall C ROOT files, the main event-by-event data tree is called simply **T**:

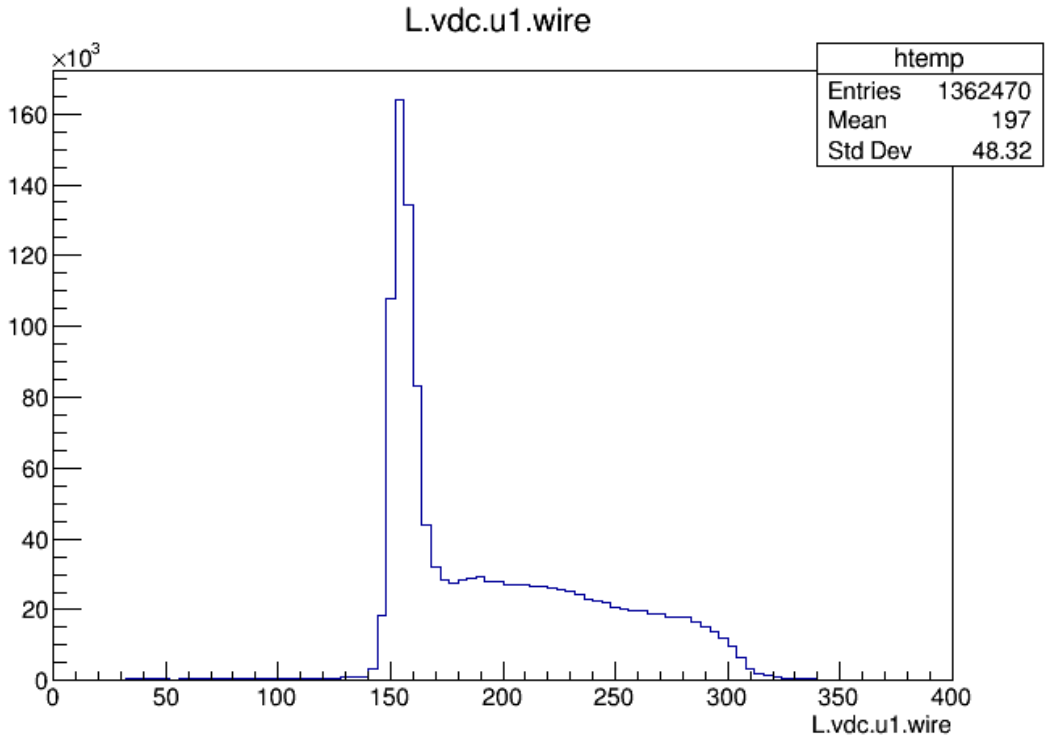
```
In [3]: f->ls();
```

```
TFile**         /data/ROOTfiles/g2p_3132_vdc_u1.root
 TFile*         /data/ROOTfiles/g2p_3132_vdc_u1.root
  KEY: THaRun        Run_Data;2        g2p run 3132 optics data
  KEY: TTree         T;1        Hall A Analyzer Output DST
  KEY: TH2F        Llsl;1        L u1 slope vs. local slope
  KEY: TH2F        Llsz;1        L u1 slope vs. cluster size
```

0.1.1 One-dimensional histograms

Let's create a basic histogram that does not need any cuts: The wire numbers of the VDC plane:

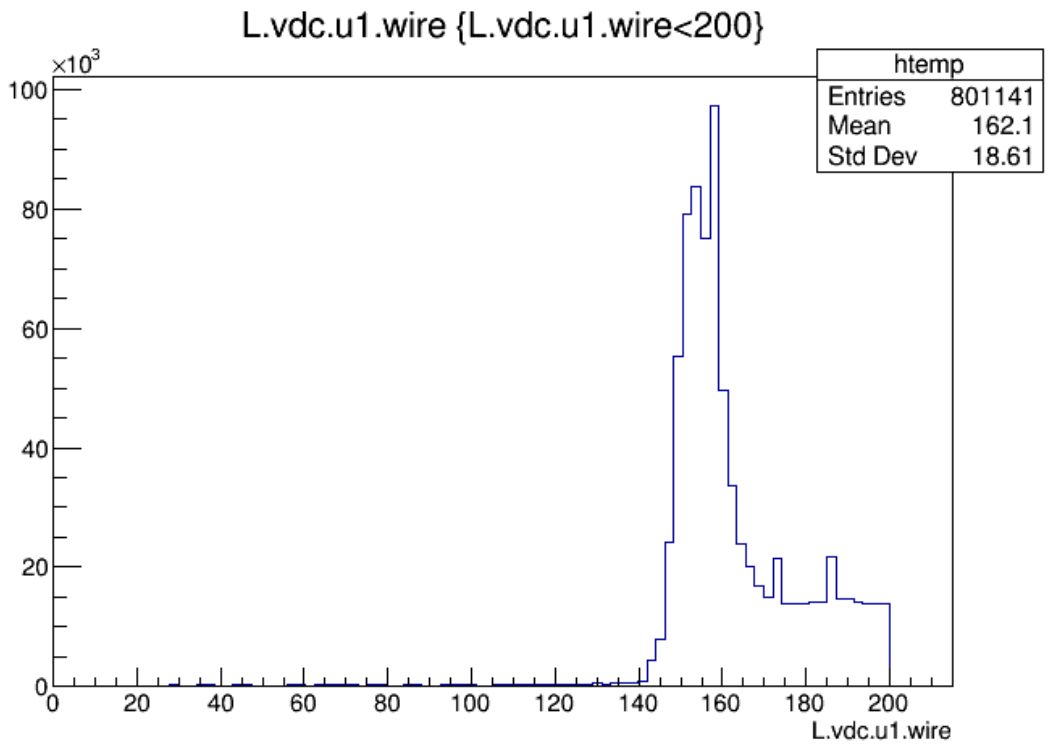
```
In [4]: T->Draw("L.vdc.u1.wire");
        c1.Draw();
```



As you can see, the spectrum is not flat. The reason is that these are data from elastic scattering from 12C, and the elastic peak is clearly visible.

To illustrate a cut, let's add a selection expression:

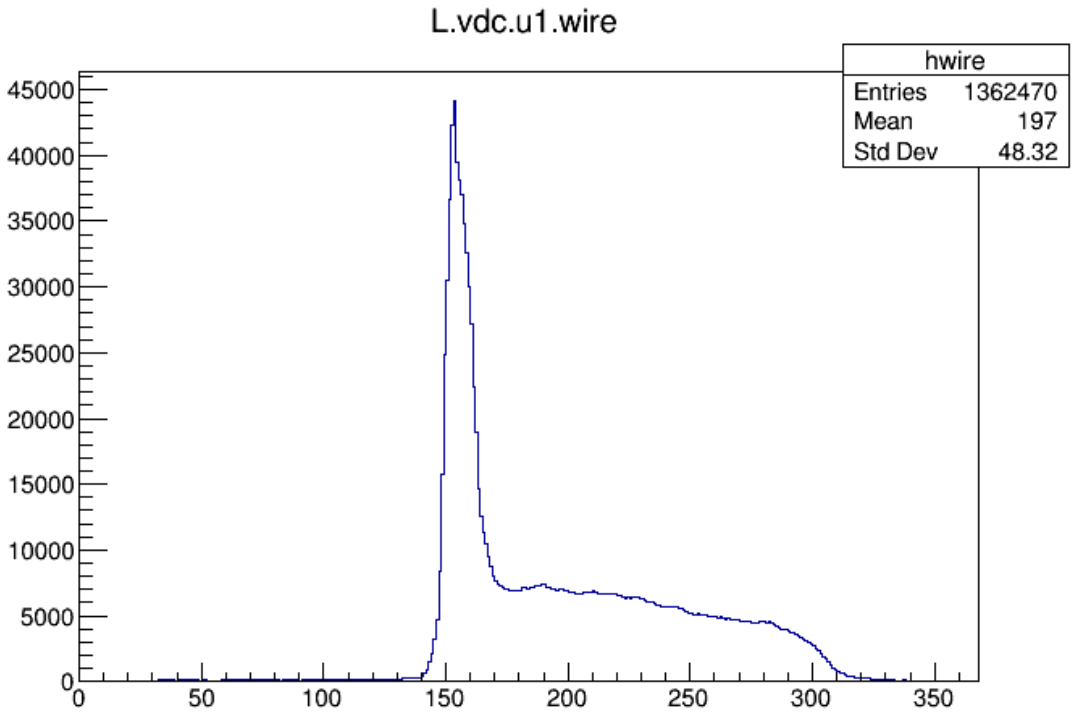
```
In [5]: T->Draw("L.vdc.u1.wire","L.vdc.u1.wire<200");
        c1.Draw();
```



Notice how the number of entries in the statistics box dropped—fewer events are now plotted.

Histograms are automatically binned by TTree::Draw(), which is often undesriable. The simplest solution is to tell ROOT your desired binning. The syntax for doing so is similar to that of the applicable 1-D or 2-D histogram constructor. Since we know that the VDC wire numbers run from 0-367 (368 total), the natural binning is one bin per wire number:

```
In [6]: T->Draw("L.vdc.u1.wire>>hwire(368,0,368)");
        c1.Draw();
```

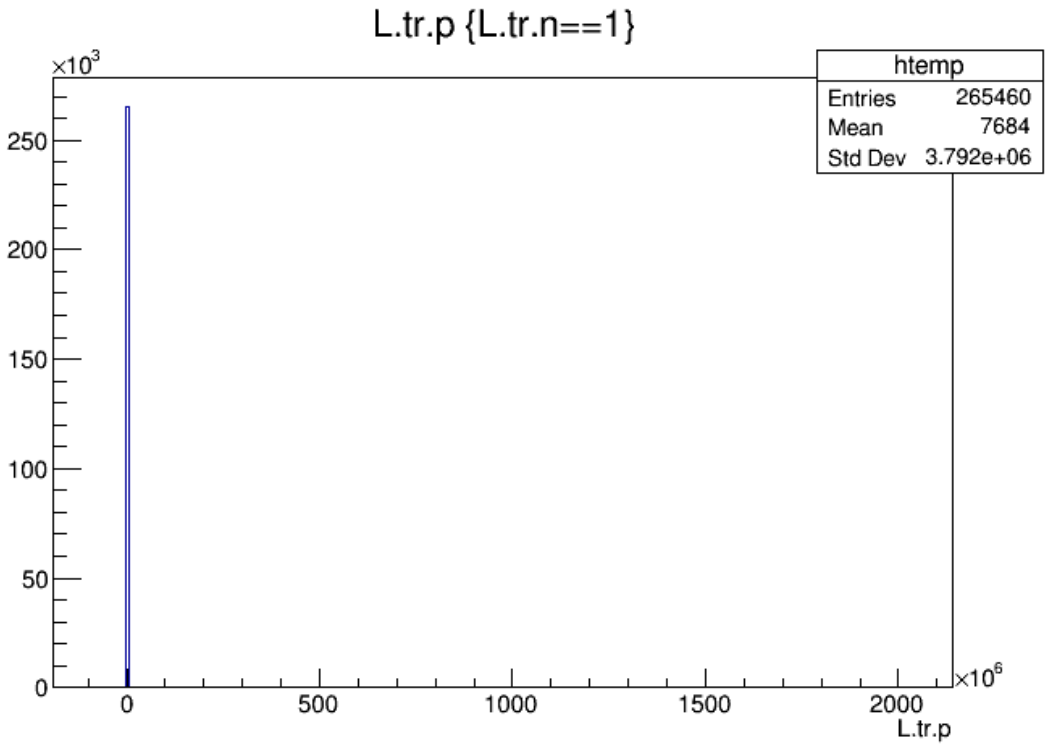


As a little exercise, repeat the above command, but choose a slightly smaller number of bins, *e.g.* 328.

Oops. Welcome to binning effects. Obviously the peaks you see are entirely artificial. Care must be take when creating histograms to avoid "beat" effects with any periodicity in the data. Rest assured, though, binning snafus happen even to experienced physicists.

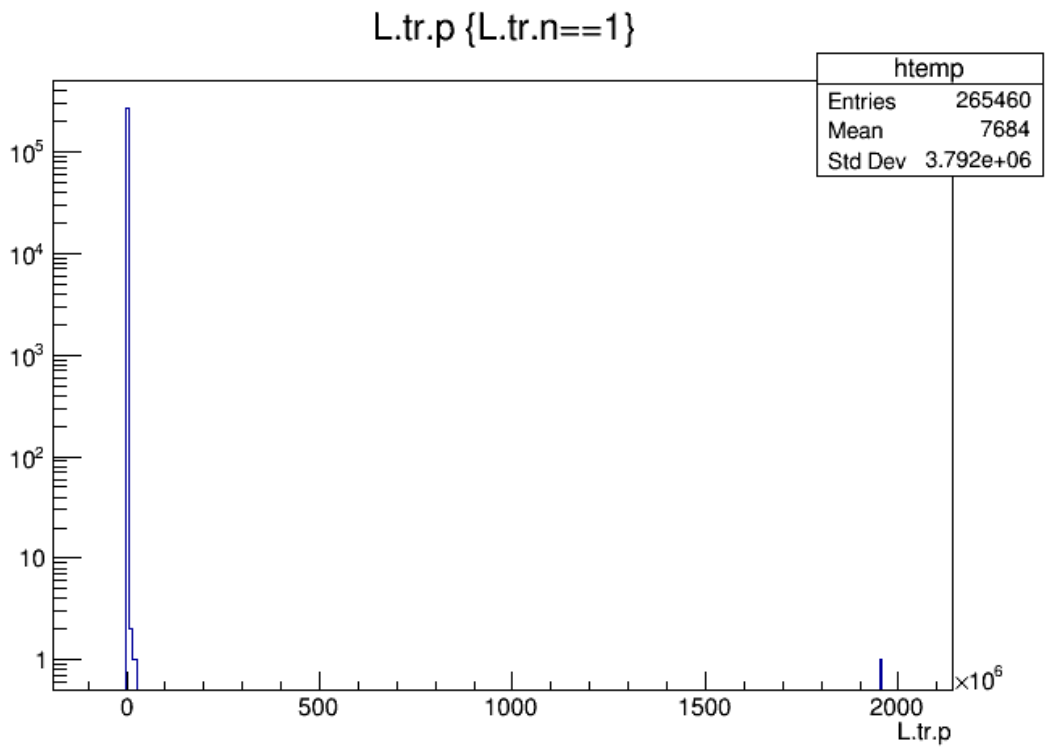
Let us look at a little bit of physics now: let's plot the distribution of the momentum of the scattered electron detected in the HRS. As we saw in Section 1, a small fraction of events contain multiple tracks, so we cut those out with a cut requiring that only a single track be reconstructed.

```
In [7]: T->Draw("L.tr.p","L.tr.n==1");
        c1.Draw();
```



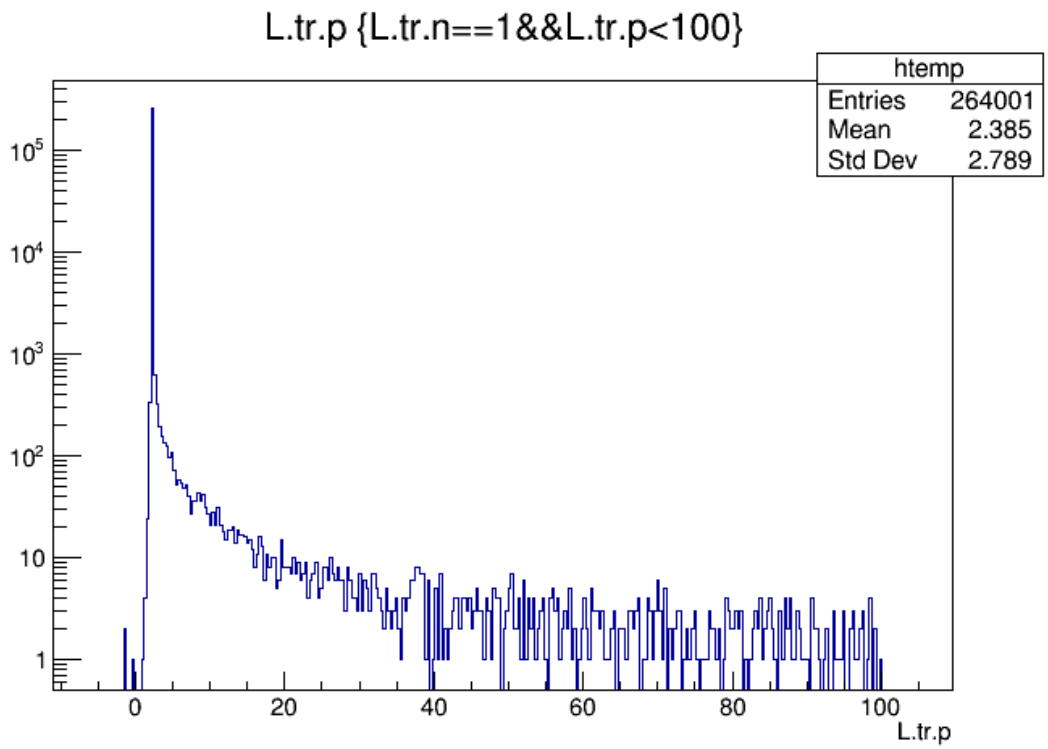
Whoops. Some tracks seems to be *REALLY* misreconstructed. To see better what's going on, let's set a logarithmic y-scale

```
In [8]: gPad->SetLogy();
        c1.Draw();
```



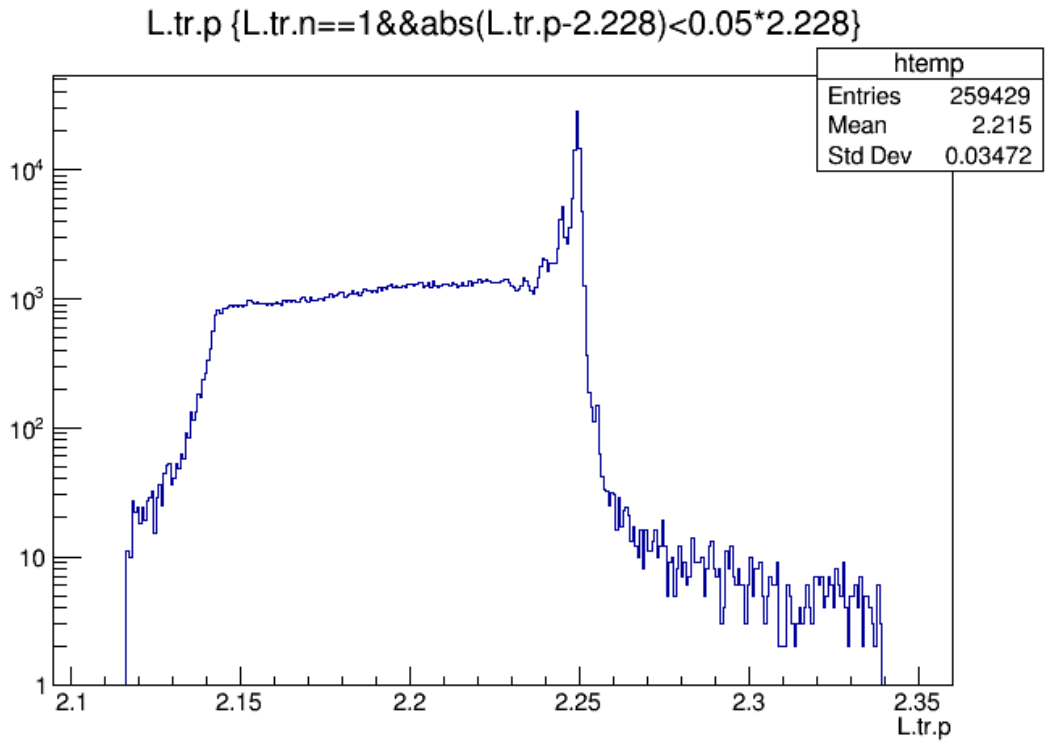
Aha. A single outlier with huge momentum. Let's cut it away:

```
In [9]: T->Draw("L.tr.p","L.tr.n==1&&L.tr.p<100");
        c1.Draw()
```



Oh my. Aside from one extreme outlier, there's lots of junk at unphysical momenta. Remember, the units are GeV, and I assure you that these are not cosmics whose momentum has been magically determined by our spectrometer. In fact, our track reconstruction code could be futher improved ... but I digress. Let's just select a range of sane momenta. This run was taken with a beam energy of 2.253 GeV/c^2 with the spectrometer central momentum set to 2.228 GeV/c. So let's select the central momentum +/-5%:

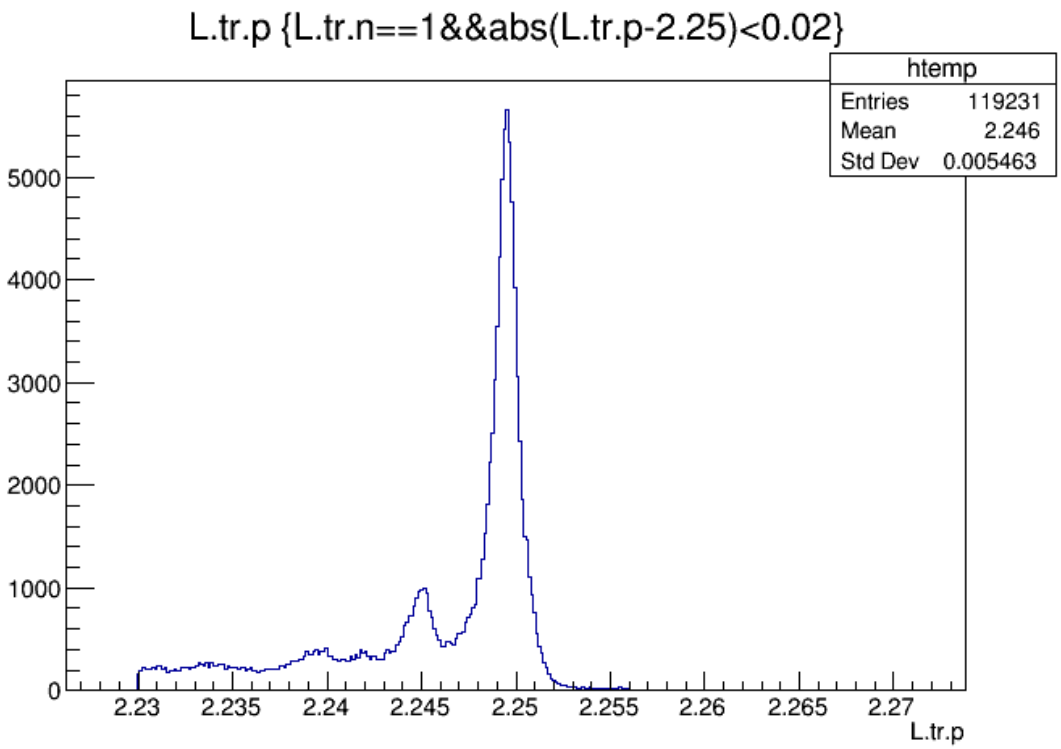
```
In [10]: T->Draw("L.tr.p","L.tr.n==1&&abs(L.tr.p-2.228)<0.05*2.228");
         c1.Draw();
```



That's more like it! Between about 2.15 and 2.26 GeV/c the signal-to-background ratio is well over 100. That's good data.

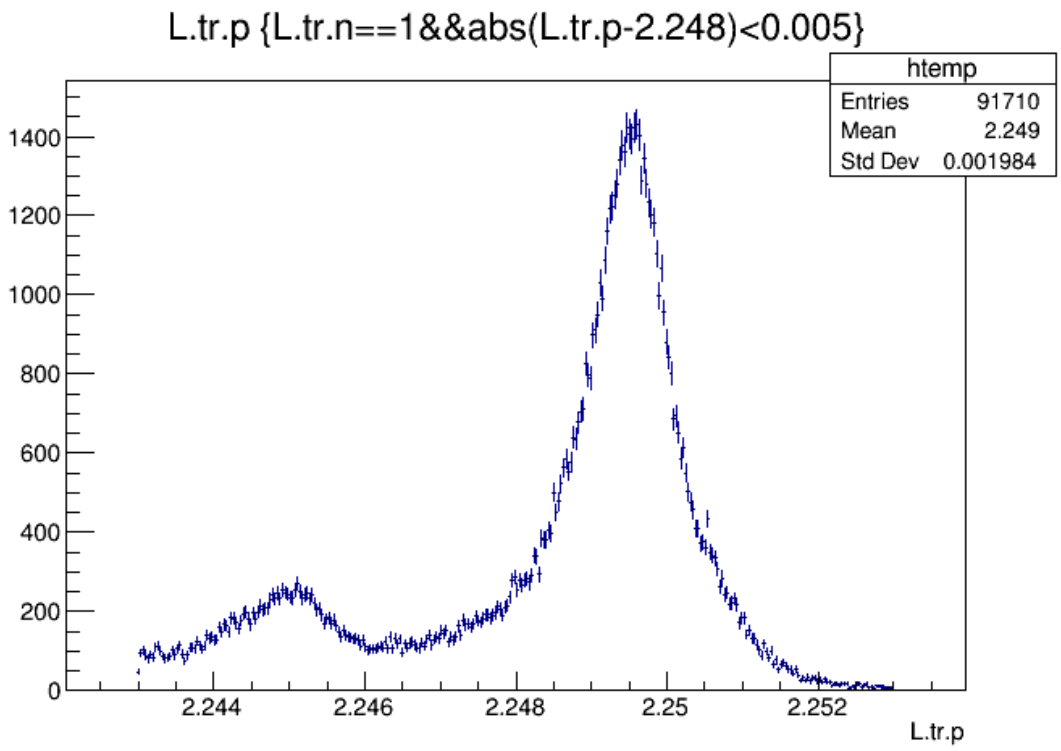
Let's look more closely at the elastic peak region. Also, let's get rid of the log scale:

```
In [11]: gPad->SetLogy(false);
         T->Draw("L.tr.p","L.tr.n==1&&abs(L.tr.p-2.25)<0.02");
         c1.Draw();
```



Now the elastic peak and first excited state of 12C are impossible to miss. Let's zoom in a little more and also turn on counting statistics error bars by using a drawing option:

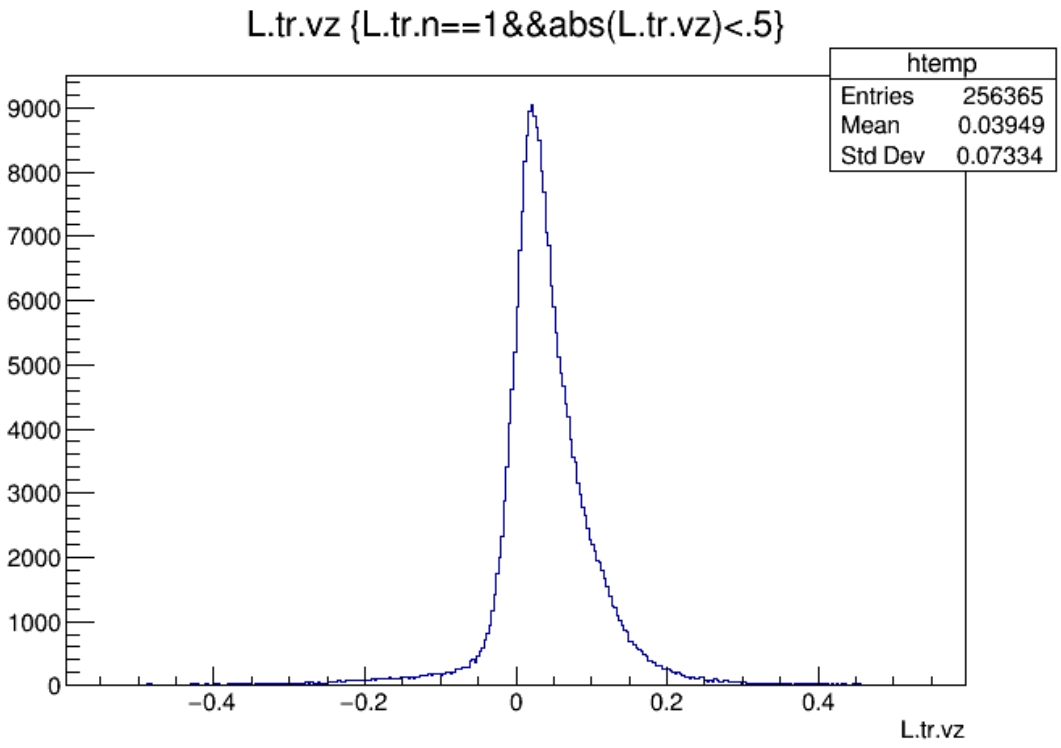
```
In [12]: T->Draw("L.tr.p","L.tr.n==1&&abs(L.tr.p-2.248)<0.005","E");
         c1.Draw();
```



In a real analysis, we would now have to apply a number of additional cuts to ensure the events in the plot are as clean as possible, before we determine number of counts in the peak, peak position and width, etc. Let's just determine one such cut: let's make sure only tracks that reconstruct to the target foil are plotted. This will eliminate some misreconstructed background.

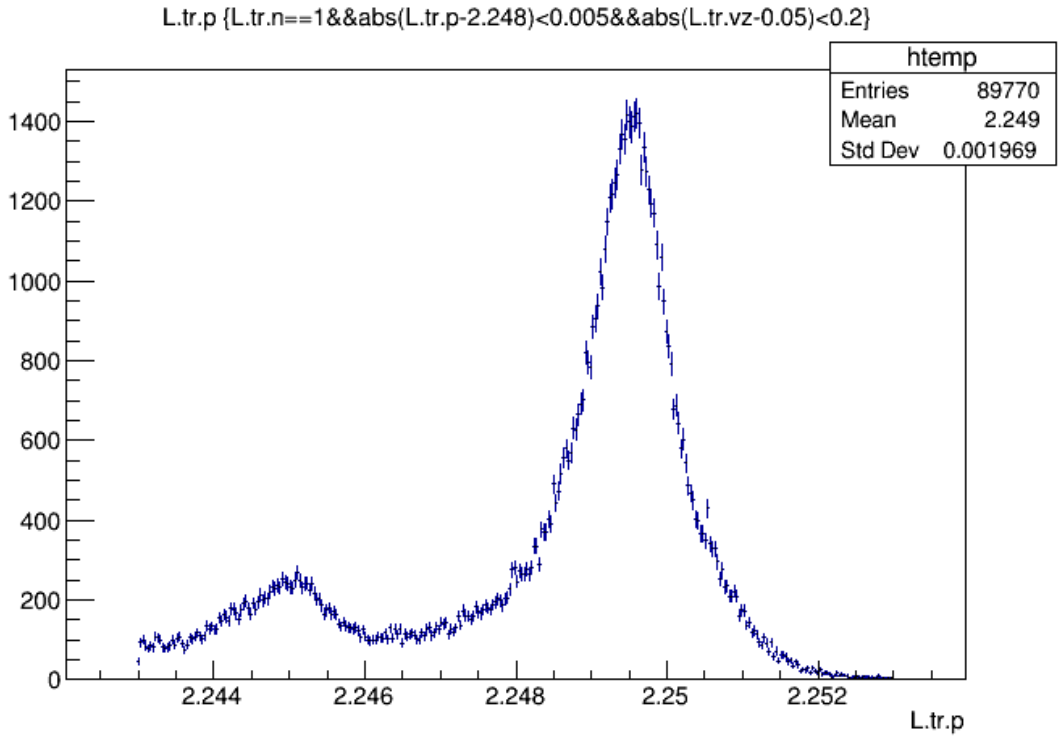
Let's look at the vertex z-position, again applying sanity cuts. We know, for example, that the target is no longer than a few 10 cm:

```
In [13]: T->Draw("L.tr.vz","L.tr.n==1&&abs(L.tr.vz)<.5");
         c1.Draw();
```



Doesn't look quite like a thin foil, does it? The main reason is that the spectrometer was placed at a very forward angle in this run: 5.69 degrees. This destroys the vertex z resolution because of the projection effect, 1/sin(theta), making this a less effective cut than it could otherwise be. But let's apply it anyway.

```
In [14]: T->Draw("L.tr.p","L.tr.n==1&&abs(L.tr.p-2.248)<0.005&&abs(L.tr.vz-0.05)<0.2","E");
         c1.Draw();
```

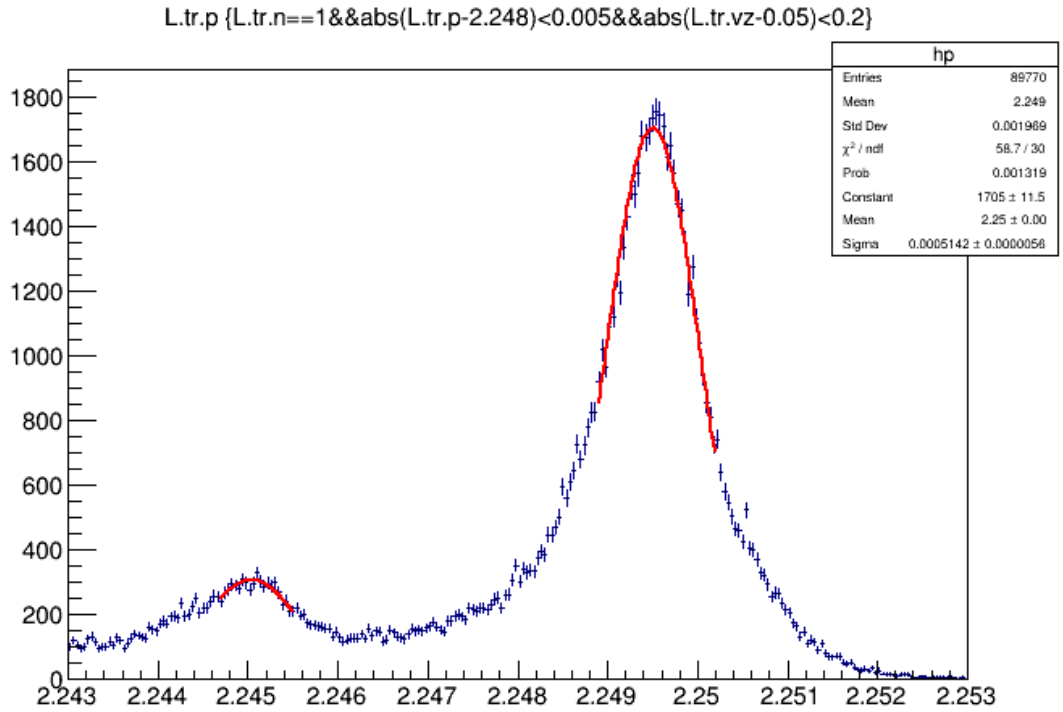


If you look at the number of entries before and after the vertex cut, you see that we cut about 2000 events (a little over 2%) with poorly reconstructed vertex position. Visually there's practically no difference.

Now let us fit the two visible peaks. To do so, we write the above plot to an explicitly-defined histogram on which we then apply ROOT fits. For simpicity, we just use a Gaussian fit function; in reality, we almost certainly need something better to match the line shape, which has a tail on the low-momentum side due to radiative effects, for example.

```
In [15]: T->Draw("L.tr.p>>hp(250,2.243,2.253)","L.tr.n==1&&abs(L.tr.p-2.248)<0.005&&abs(L.tr.vz-
```

```
In [16]: gStyle->SetOptFit(1111); // turn on display of fit results in statistics box
         hp->Fit("gaus","","",2.2489,2.2502);
         gaus2 = new TF1("gaus2","gaus");
         hp->Fit("gaus2","+","",2.2447,2.2455);
         c1.Draw();
```




```
 FCN=58.7011 FROM MIGRAD    STATUS=CONVERGED      75 CALLS          76 TOTAL
                     EDM=2.02426e-10    STRATEGY= 1  ERROR MATRIX UNCERTAINTY   2.5 per cent
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     1.70544e+03   1.14541e+01  -8.39341e-02   3.20924e-06
   2  Mean         2.24951e+00   3.74369e-06  -1.36073e-08   1.64116e+00
   3  Sigma        5.14203e-04   5.55341e-06   1.16170e-06   4.12273e-03
 FCN=13.029 FROM MIGRAD    STATUS=CONVERGED     112 CALLS         113 TOTAL
                     EDM=3.71366e-09    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     3.07944e+02   5.89998e+00   7.52589e-03  -1.52004e-05
   2  Mean         2.24504e+00   1.77328e-05   1.07052e-06  -2.38181e+00
   3  Sigma        5.25557e-04   4.12949e-05   5.29996e-05  -6.22284e-04
```

Move or shrink the statistics box by hand. I cannot find a way to position and resize it from within the notebook; it seems to ignore most gStyle commands.

Now we can get a rough estimate of the separation of the first excited state of 12C. Let's retrieve the fit results programmatically:

```
In [17]: f1 = hp->GetFunction("gaus");
         f2 = hp->GetFunction("gaus2");
```

```
        p_el = f1->GetParameter(1);
        e_el = f1->GetParError(1);
        p_ex1 = f2->GetParameter(1);
        e_ex1 = f2->GetParError(1);
        cout << "E_ex1 = " << 1e3*(p_el-p_ex1) << " +/- " << 1e3*TMath::Sqrt(e_el*e_el+e_ex1*e_
```

```
E_ex1 = 4.47143 +/- 0.0181236 MeV
```

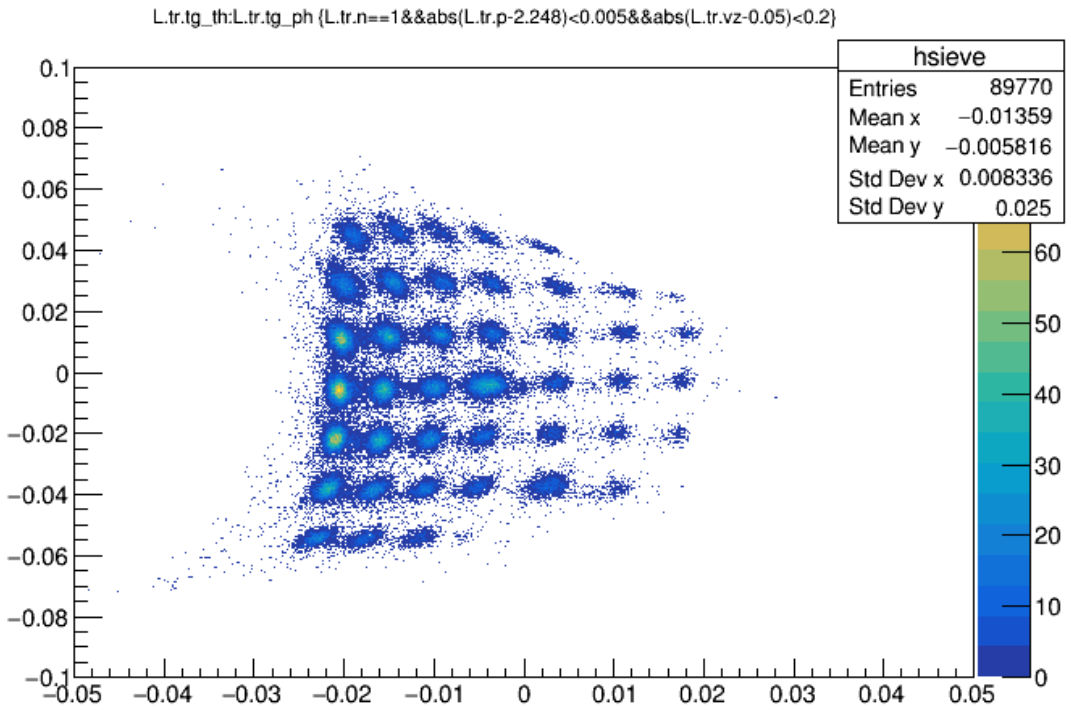
Maybe not call the PDG just yet, but not bad for a first shot.

0.2 Two-dimensional histograms

To conclude the section on `TTree::Draw`, let's take a brief look at two-dimensional histograms. We can only scratch the surface of the myriad of possibilities here.

Let's plot the in-plane vs. out-of-plane angles of the track as seen from the target. This will visualize the sieve slit pattern. Let's restrict ourselves to events near the elastic peak with good target z-position, as before. We use the "COLZ" drawing option to allow us to estimate the peak heights in the 2-D scatter plot.

```
In [18]: T->Draw("L.tr.tg_th:L.tr.tg_ph>>hsieve(500,-0.05,0.05,500,-0.1,0.1)","L.tr.n==1&&abs(L.
         c1.Draw();
```



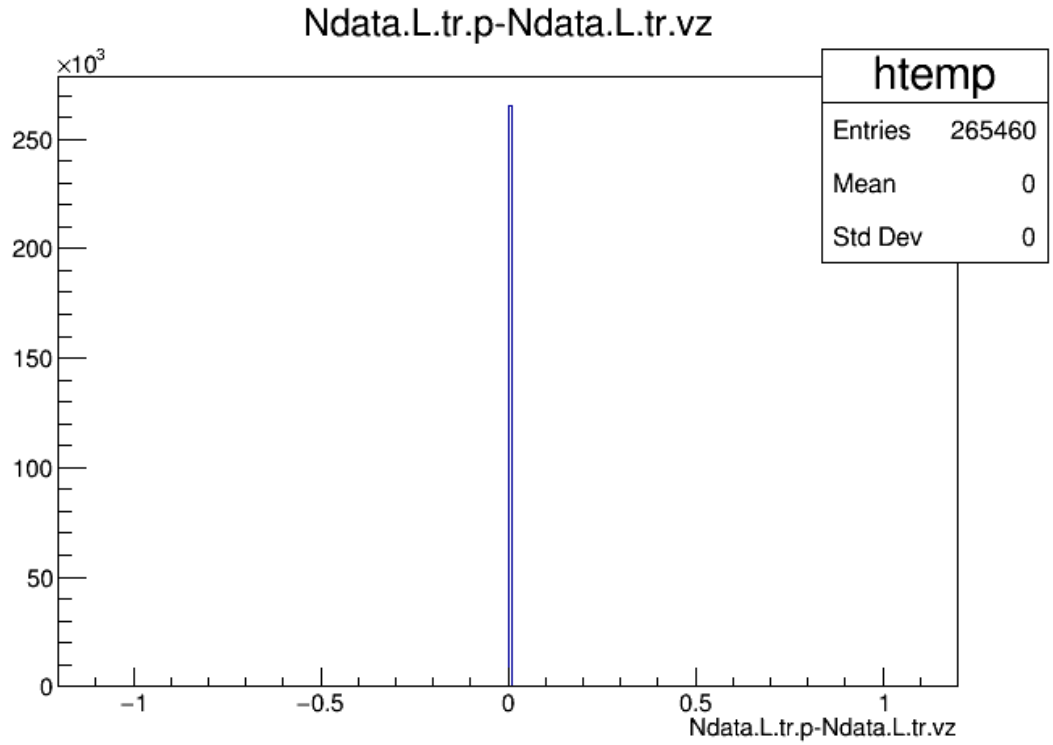
At this point I wanted to show 3-D features, but they seem to push the limits of what ROOT notebooks are capable of. Let us quickly rerun these plots in a non-notebook ROOT session.

0.3 Caveat: Processing arrays with TTree::Draw

One particular problem with `TTree::Draw` arises when arrays are involved. If more than one array appears in a `Tree::Draw` command, either in the variable expression or in the selection expression (the first and second argument), ROOT will iterate all arrays with one and the same index, up to the size of the smallest array. (This is equivalent to processing only the diagonal elements of a multidimensional matrix.) Although there are exceptions, this usually only makes sense if all arrays involved are parallel, *i.e.* their index has the same meaning (*e.g.* PMT number, track index) for all of them.

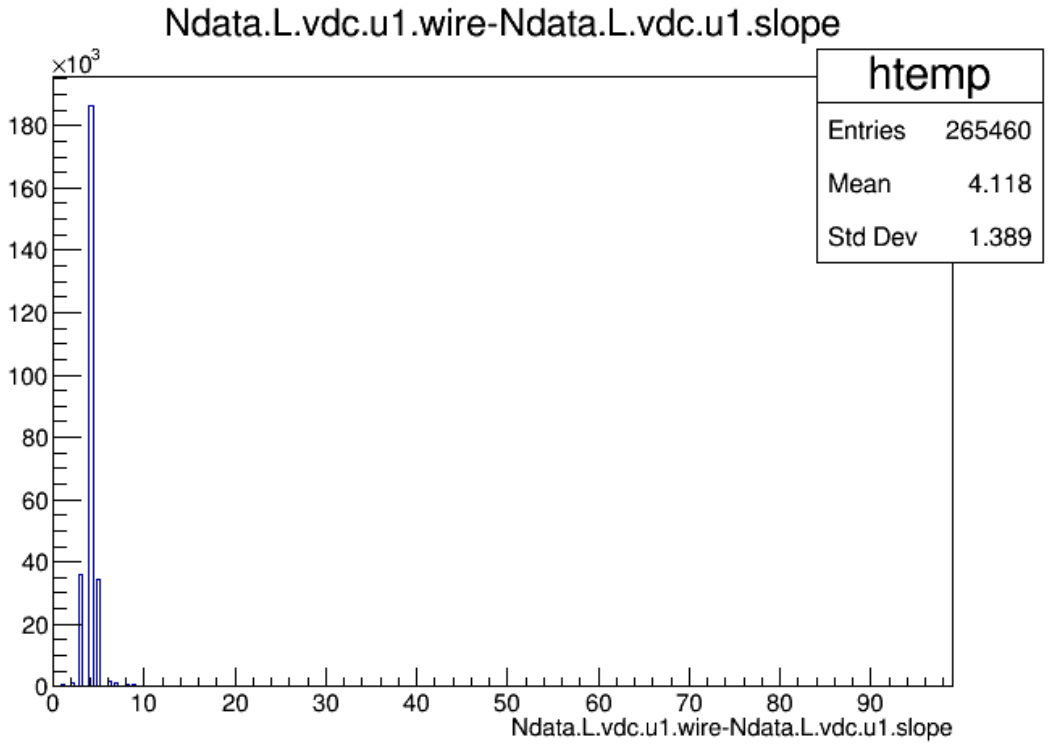
One can test empirically if two arrays are parallel by plotting the `Ndata` size counters of the arrays against each other, or by plotting their difference. For parallel arrays, the counters must always be identical and the difference therefore must be always zero. This is the case, for example, for all `L.tr.*` variables, whose index is the track index:

```
In [19]: T->Draw("Ndata.L.tr.p-Ndata.L.tr.vz");
         c1.Draw();
```



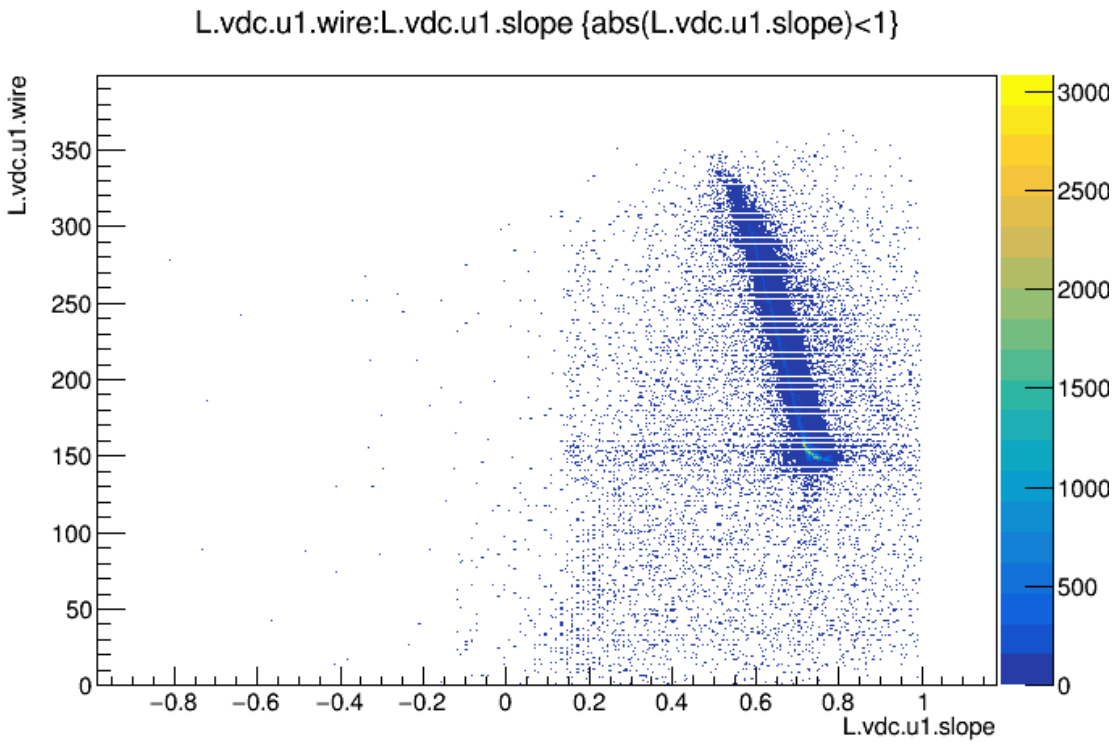
Here's a counter-example: The VDC wire number vs. the VDC cluster slope. The wire number array is indexed by the counter of wires with signals and runs up to the number of wires that have a signal in a given event, typically some 5-6. The cluster slope is indexed by the cluster number and runs to the number of clusters in the given plane for the given event, typically 1, occasionally two or more (giving rise to multi-track candidates). Both numbers are not completely independent (more clusters usually require more wires), but defnitely don't count in lockstep. The differece plot above shows this:

```
In [20]: T->Draw("Ndata.L.vdc.u1.wire-Ndata.L.vdc.u1.slope");
         c1.Draw();
```



Therefore, a plot like this contains at least some nonsense entries:

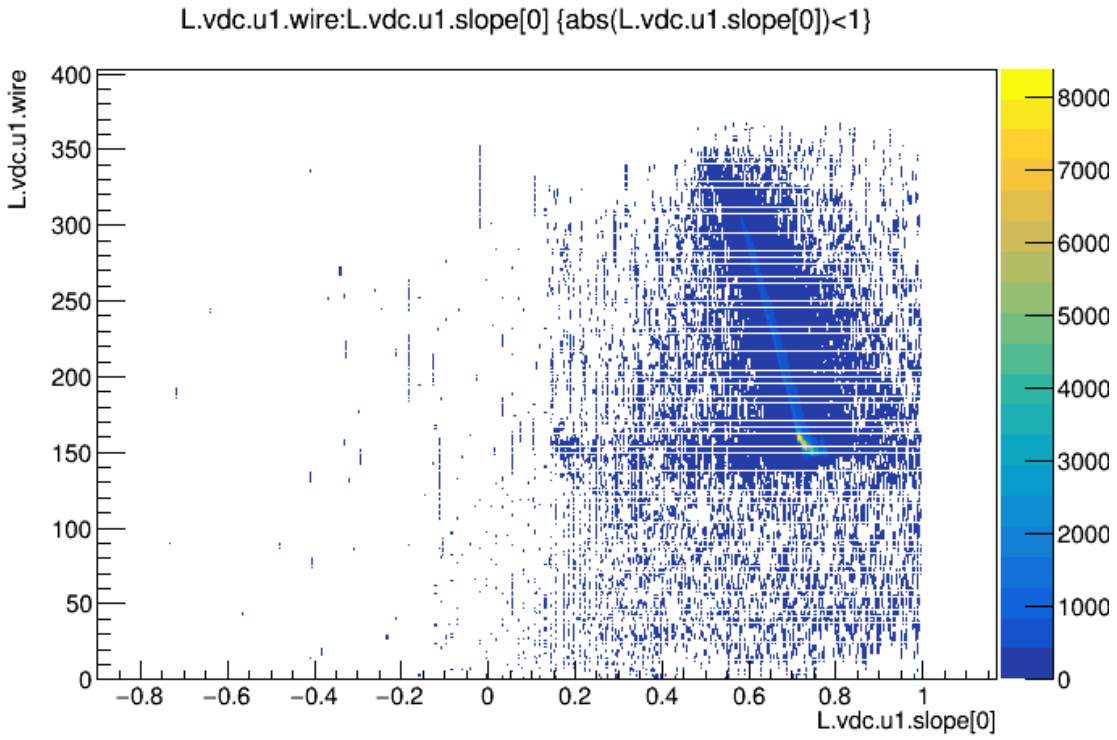
```
In [21]: T->Draw("L.vdc.u1.wire:L.vdc.u1.slope","abs(L.vdc.u1.slope)<1","COLZ");
         c1.Draw();
```



We're lucky, however, because the size of the slope array is usually just 1 AND the first wire number is usually close to the cluster centerposition, so the plot does approximately show the very real physical effect of the slope decreasing with increasing wire number (since the tracks get steeper closer to the front of the spectrometer).

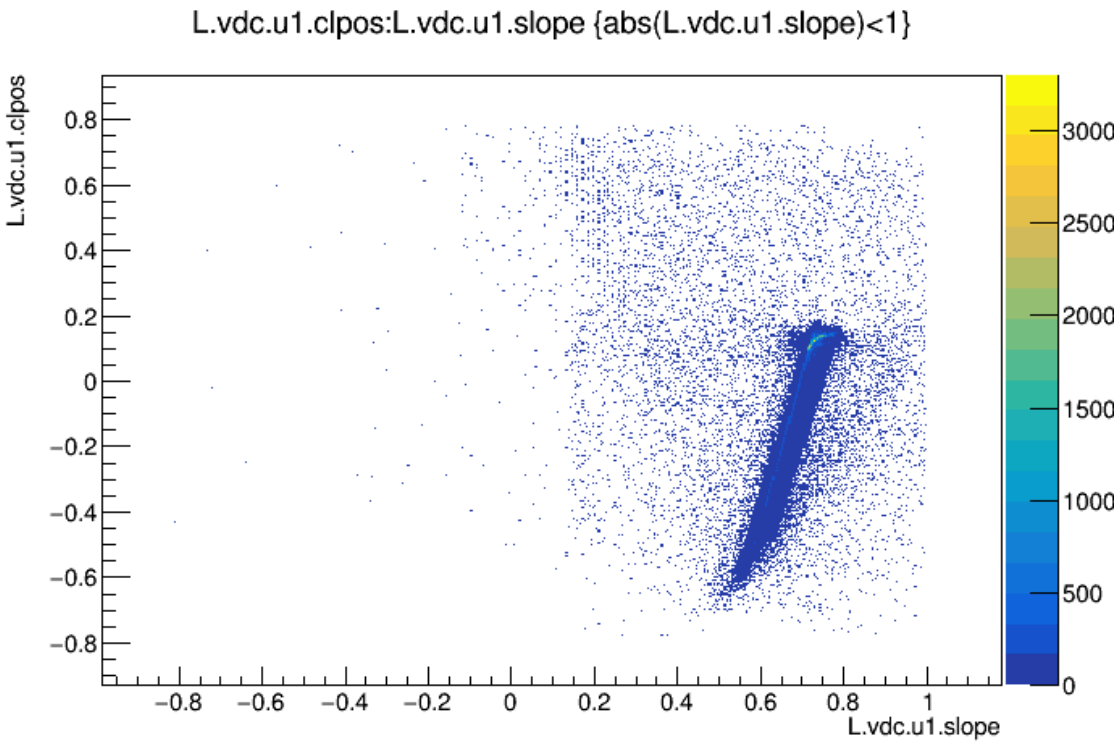
A better approach to this problem is to force one of the arrays to a scalar type by giving an explicit array index, usually `[0]`:

```
In [22]: T->Draw("L.vdc.u1.wire:L.vdc.u1.slope[0]","abs(L.vdc.u1.slope[0])<1","COLZ");
         c1.Draw();
```



Of course, this is still not a very meaningful plot. Why plot half a dozen presumably adjacent wire numbers? The correct way to show this dependence is to plot cluster slope vs. cluster position:

```
In [23]: T->Draw("L.vdc.u1.clpos:L.vdc.u1.slope","abs(L.vdc.u1.slope)<1","COLZ");
         c1.Draw();
```



Here, the correlation is correctly shown for *ALL* clusters, not just the first one. (The sign flip occurs because the wire number increases as the x-coordinate decreases. Negative cluster positions indicate the front of the spectrometer, closest to the target.)