Development of TRatioPlot in ROOT

Paul Gessinger
CERN Summer Student project

1 Status of ratio and similar plots in ROOT

The ROOT data analysis and visualization framework is a software package which is widely used in physics, especially in high energy physics. As the software became more and more sophisticated, numerous improvements were made, and regularly, simplifications for were achieved.

Examples for this are the spectrum fitter and the stack facility. A common visualization which has so far been lacking a direct implementation is the ratio plot, as well as a few similar types of plots. The common element is a splitting of the area into to sub areas, one of which contains the inputs, typically histograms, whereas the other one displays the result of a calculation, as for example the result of a division of one histogram by the other.

While it was perfectly possible to create a plot like this in ROOT before, achieving a good looking result could be tedious, which led to a large number of people writing automation on top of ROOT itself, that would take care of the process for them. The scope and goal of my summer student project at CERN was to implement a class in ROOT itself, that can take care of the most common types of calculations, and produces high quality visuals. A key factor in the design was to ensure interactive usability of the class, making it possible to explore calculation results more easily.

As can be seen Figure 1, directly using provided facilities in ROOT yields unsatisfactory visual results. Axis labels are sized inconsistently, as the font sizes are derived from the size of the pad within the parent. Showing axes titles would reveal that here sizing is also inconsistent, as are the offsets from the axis itself. A major problem for quality is the clipping of the 0 of the upper y axis. It results from the pads meeting, and the lower pad cutting of the overflowing the lowest label of the upper y axis.

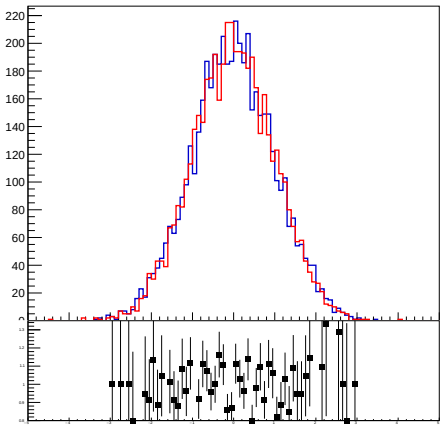


Figure 1: Naive implementation of a ratio plot using two `TPad` objects.

Not visible in the static image are the issues with interactivity this implementation has. When the user views a ROOT plot, it is possible to manipulate the visuals using the mouse. For example the user can zoom in by clicking and dragging on one of the axes. It is also possible to switch the display between a linear and a logarithmic scale. Clicking and dragging on the outer frame of the pads allows resizing the content frame. When performing any of these actions, the pads react completely independently, meaning changes made to one of the pads is not reflected on the other on. This can be problematic, specifically when attempting to use the zoom feature to explore the content, since replicating the zoomed range precisely can prove a challenge.

2 TRatioPlot

The aforementioned shortcomings are attempted to be fixed by implementing a class called `TRatioPlot` and focussed on three main goals:

- Produce high quality visuals with minimal setup
- Enable meaningful interactivity

- Implement most common calculations for these types of plots

2.1 Output quality and size normalization

Out of the box, when working with multiple pads, sizes are derived from the size of the pad. Therefore, if one attempts to have consistent sizing, conversion factors have to be derived, or absolute sizing has to be utilized. While the latter is easily achievable, it also removes the useful benefits of having adaptable sizing. Say you want to have a ratio type plot inside a positioned pad. In this case adaptable sizing will be useful to produce readable output.

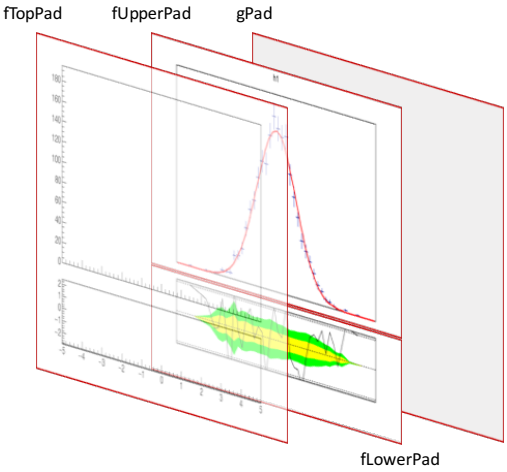


Figure 2: `TPad` structure of the new `TRatioPlot` class.

In order to preserve the sizing mechanism, the pad structure of the class was adjusted as can be seen in Figure 2. When creating an instance of the class, it will create three `TPad` objects inside the currently active pad. Two pads receive the input histograms and the output graphs respectively. Drawing of axes is disabled when drawing objects in these pads. The draw option "A" of `TH1` prevents visible axes, while retaining the ability to interact with them. The draw option "I" achieves the same when drawing `TGraph`. The pad marked by *fTopPad* receives all the visual axes. Since all of them are drawn in one pad spanning the full dimensions of the containing pad, label and title sizes and offsets are consistent between them. Properties for the visual axes are first imported from the corresponding axes of the drawn objects, and changed where necessary. For example, the class always hides the axes labels and title on the upper x axis. The margins of the pads are synchronized. Using setters on `TRatioPlot`, margins that should be identical can be set at the same time. Another setter can be used to determine the separation between the upper and the lower plot, while yet another one determines the point at which the two pads meet. Draw options given to `TRatioPlot` can enable and disable visual features, they are listed in Table 1.

Draw option	Description
grid / nogrid	enable (default) or disable drawing of dashed lines on lower plot
hideup	hides the first label of the upper axis if there is not enough space
fhideup	always hides the first label of the upper axis
hidelow (default)	hides the last label of the lower axis if there is not enough space
fhidelow	always hides the last label of the lower axis
nohide	does not hide a label if there is not enough space
noconfint	does not draw the confidence interval bands in the fit residual case
confint	draws the confidence interval bands in the fit residual case (default)

Table 1: Draw options for `TRatioPlot`.

The class will hide y axis labels once the pads are closer than a certain threshold, to prevent clipping and also overlapping of axis labels. The lower plot contains dashed lines, promoting points of interest, such as the value 1 in a ratio. This behaviour can be turned off via a draw options, however it is also possible to set the y positions of an arbitrary number of dashed lines manually using the `TRatioPlot::SetGridlines()` method, by passing an array of y values.

2.2 Interactive manipulation

In order to support the functionality of ROOT to view and manipulate most structures interactively, care has to be taken. Without the use of `TRatioPlot`, modifying one of the pads used for display

will not trigger a corresponding change in the other one. This is undesirable for obvious reasons and TRatioPlot aims to fix these problems. Using the Signal/Slot mechanism, it is possible to react to changes, that are triggered and processed elsewhere. The class connects to three signals from the contained pads, *RangeAxisChanged*, *Resized* and *UnZoomed*, only the first of which used to exist. It was called whenever the actual range of the axis changed, as the name indicates. TPad was modified to also emit this signal when on of the axes is switched between linear and logarithmic mode. The second signal is emitted after the pad has responded to interactive resizing by clicking and dragging, which means that it already contains the updated coordinates. The last signal was added and is emitted when selecting the unzoom entry in the context menu of an axis.

Using slots for these signals, TRatioPlot can react to changes made by the user. The class determines which element has changed, by comparing values to reference values stored inside the class after a previous change, or by determining the sender of the signal. It then updates internal references accordingly, and sets the corresponding properties on the relevant other objects. For instance, zooming updates the unchanged pad's axis to the one that was modified by clicking and dragging. A subsequent paint picks up these changes, and renders the updated histograms and graphs, while also updating the graphical axes that the class manages itself. Aside from axis ranges, this method is applied to changing pad margins, logarithmic or linear axis setting for the x axis, as well as the point where the pads meet.

2.3 Calculation modes

A large number of calculations can be performed, that lend itself to visualization in the manner that this class tries to provide. Effectively, any calculation whose output shares the x axis with this inputs benefits from display in this way. While implementing all calculations in such a class would be out of scope for a rather generic solution, it is still useful to have pre-build facilities for a few of the most used calculation modes.

Ratio of two histograms

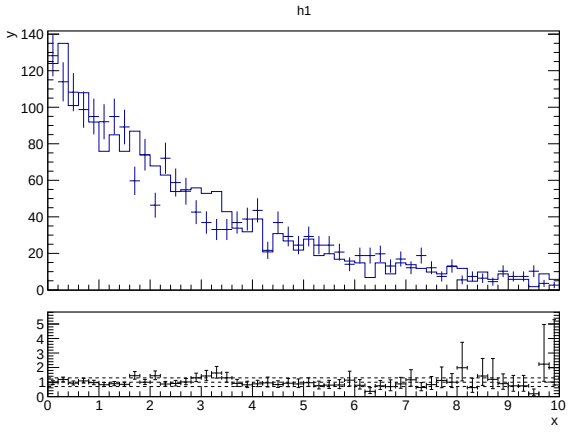
The simplest case is the division of one histogram by another one. This can be performed in multiple ways using TRatioPlot. The default, when instantiating the class simply with two histograms, is to delegate the calculation to TGraphAsymmErrors::Divide, while passing it the option *pois*, which makes use of the Poisson error for each of the bins, and yields asymmetric errors on the output. Another option is using the method TH1::Divide, which yields symmetric errors and can be invoked using the option *divsym* in the constructor.

Difference of two histogram

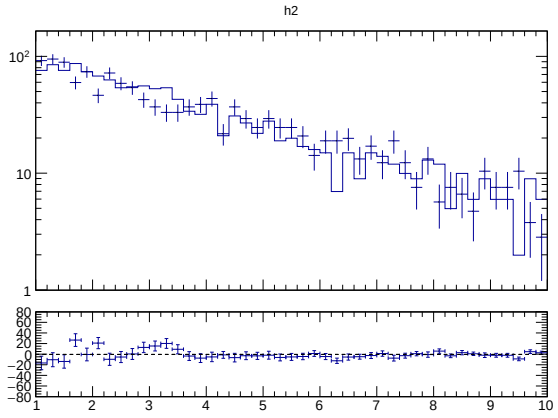
The second use case is two calculate the absolute difference between the two passed histograms. This can be achieved by giving the option *diff* when instantiating. Another possibility is to calculate the difference between the histograms, and then divide by the statistical uncertainty of the first histogram. This mode can be invoked by using the option *diffsig*. Two possibilities exist for the source of the uncertainty, which can be selected using an additional option. The default is to use the method TH1::GetBinError, which always yields symmetric uncertainties. By passing the option *errasym* one can have the calculation use the computed errors obtained through TH1:GetBinErrorUp and TH1::GetBinErrorLow, which can yield asymmetric errors. Depending on whether the first histogram's bin content is above the second histogram one's, the up or low error is used. To ensure correct calculation, TH1::SetBinErrrorOption must be called on the input histogram.

Fit residual

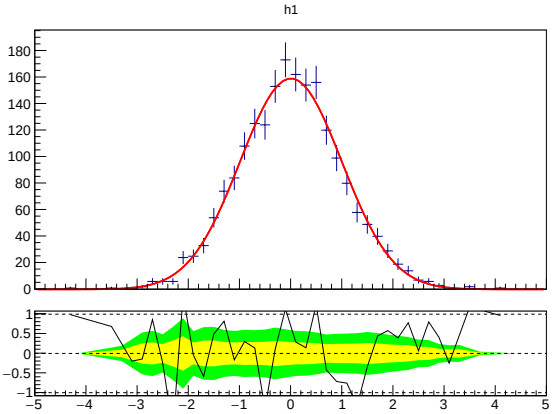
When fitting, it is often interesting to have a graphical representation of the quality and goodness of a fit. To provide this, TRatioPlot can calculate the fit residual, which is the difference between the actual bin content, and the corresponding fitted function value at the center of the bin, divided by the uncertainty. Different uncertainties can be used, firstly, both options described in the previous section are available, while a third option, *errfunc*, enables the use of the square root of the function value as the uncertainty. The fit residual calculation can be invoked by using the constructor of the class which only accepts a single histogram. It is expected that the given histogram contains one element in it's list of fitted functions. In addition to the fit residual itself, the class will also present the 1σ and



(a) Ratio of two histograms. (see Figure 4 for code)



(b) Difference between two histograms. (see Figure 5 for code)



(c) Residual of a fit to a histogram. (see Figure 6 for code)

2σ confidence intervals. Drawing of these bands can be configured by using the draw options found in Table 1.

3 Example output

This section shows three example outputs, that can be achieved using `TRatioPlot`. The corresponding code can be found in the appendix in an abbreviated form.

Figure 3a shows a simple ratio of two histograms, Figure 3b shows the difference between two histograms and Figure 3c shows the case of a fit residual to a histogram.

4 Summary

During the CERN Summer Student project, a class called `TRatioPlot` was integrated into the ROOT framework, than can greatly simplify common plotting tasks, and potentially save a lot of time spend by a lot of people on reimplementing the same functionality. The class can produce high quality visualization of ratio and similar plots. It also enables interactivity for these kinds of plots for the first time in ROOT. Additionally, the class implements a number of commonly performed calculations typically shown in the relevant manner. This means that producing these plots will require less code in most cases. Customization of the output is also improved, since the class provides new facilities to operate on coupled display pads at the same time.

A Appendix

```
void ratioplot1() {
   gStyle->SetOptStat(0);
   auto c1 = new TCanvas("c1", "A ratio example");
   auto h1 = new TH1D("h1", "h1", 50, 0, 10);
   auto h2 = new TH1D("h2", "h2", 50, 0, 10);
   auto f1 = new TF1("f1", "exp(- x/[0] )");
   f1->SetParameter(0, 3);
   h1->FillRandom("f1", 1900);
   h2->FillRandom("f1", 2000);
   h1->Sumw2();
   h2->Scale(1.9 / 2.);
   h1->GetXaxis()->SetTitle("x");
   h1->GetYaxis()->SetTitle("y");
   auto rp = new TRatioPlot(h1, h2);
   c1->SetTicks(0, 1);
   rp->Draw();
   c1->Update();
}
```

Figure 4: Simple example for a ratio.

```
{
   gStyle->SetOptStat(0);

   auto c1 = new TCanvas("c1", "logxy diff");
   c1->cd();
   gPad->SetFrameFillStyle(0);
   c1->SetLogy();
   auto h2 = new TH1D("h2", "h2", 50, 0, 10);
   auto h3 = new TH1D("h3", "h3", 50, 0, 10);
   auto f1 = new TF1("f1", "exp(- x/[0] )");
   f1->SetParameter(0, 3);
   h2->FillRandom("f1", 1900);
   h3->FillRandom("f1", 2000);
   h3->Sumw2();
   h3->Scale(1.9 / 2.);

   h2->GetXaxis()->SetRangeUser(1, 10);
   auto rp5 = new TRatioPlot((TH1*)h2->Clone(), (TH1*)h3->Clone(), "diff grid");
   rp5->Draw();
   rp5->GetLowerRefGraph()->SetMinimum(-80);
   rp5->GetLowerRefGraph()->SetMaximum(80);

   c1->Update();
}
```

Figure 5: Simple example for a difference of two histograms.

```
{
   gStyle->SetOptStat(0);

   auto c1 = new TCanvas("c1", "fit residual simple");
   auto h1 = new TH1D("h1", "h1", 50, -5, 5);
   h1->FillRandom("gaus", 2000);
   h1->Fit("gaus");
   h1->Sumw2();

   c1->Clear();

   auto rp1 = new TRatioPlot(h1);
   rp1->Draw();
   c1->Update();
}
```

Figure 6: Simple example for a fit residual.