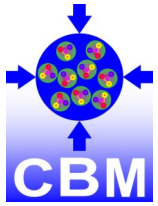


Good plot in ROOT

(a personal perspective)

Krzysztof Piasecki
University of Warsaw, Poland

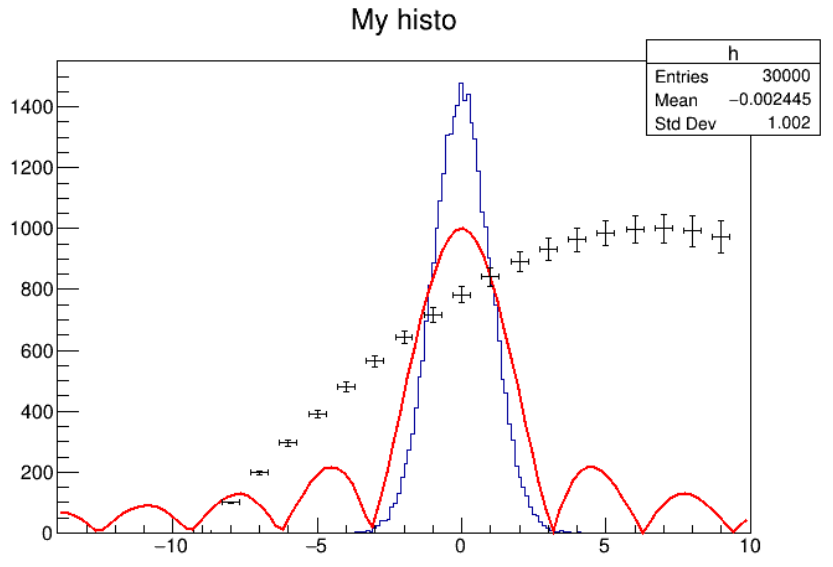
Assume we want to plot our data, with intention to show it in a talk or publish in a report or paper.

- (All our content is available on GitHub:
  `git clone https://github.com/kpias/GoodPlotInRoot`)

Let's start with [this macro] :

```
int firstPlot ()
{
  /*** Stage 1: generating some dummy data ***/

  TH1F* h = new TH1F ("h", "My histo", 200, -14, 10);
  h->FillRandom ("gaus", 30000);

  TF1* f = new TF1 ("f", "1000*abs(sin(x)/x)", -14, 14);

  Int_t i = 0;
  Double_t x[50], y[50], ex[50], ey[50];
  for (Double_t xval = -9; xval <= 9; xval++, i++) {
    x[i]  = xval;
    y[i]  = 1000 * sin ( (xval + 9)/10 ) ;
    ex[i] = 0.3;
    ey[i] = (xval+9) * 3 ;
  }

  TGraphErrors* g = new TGraphErrors (i, x, y, ex, ey);

  /*** Stage 2: Data visualization ***/

  h->Draw ();
  f->Draw ("same");
  g->Draw ("P");

  return 0;
}
```

- `$ root -l firstPlot.C`



Let's have a closer look:

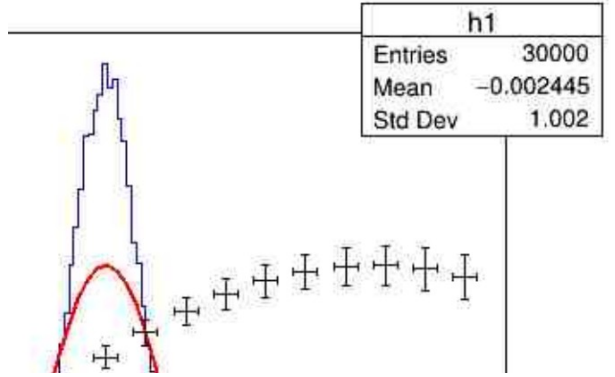
- No description of axes
- Title describes ... which one of 3 objects?
  - ↳ no legend
- Statistics box often unnecessary
- Y axis: structure of divisions a bit too dense

Embedding this figure into a text document:

- font sizes are too small
- adding caption makes upper title redundant

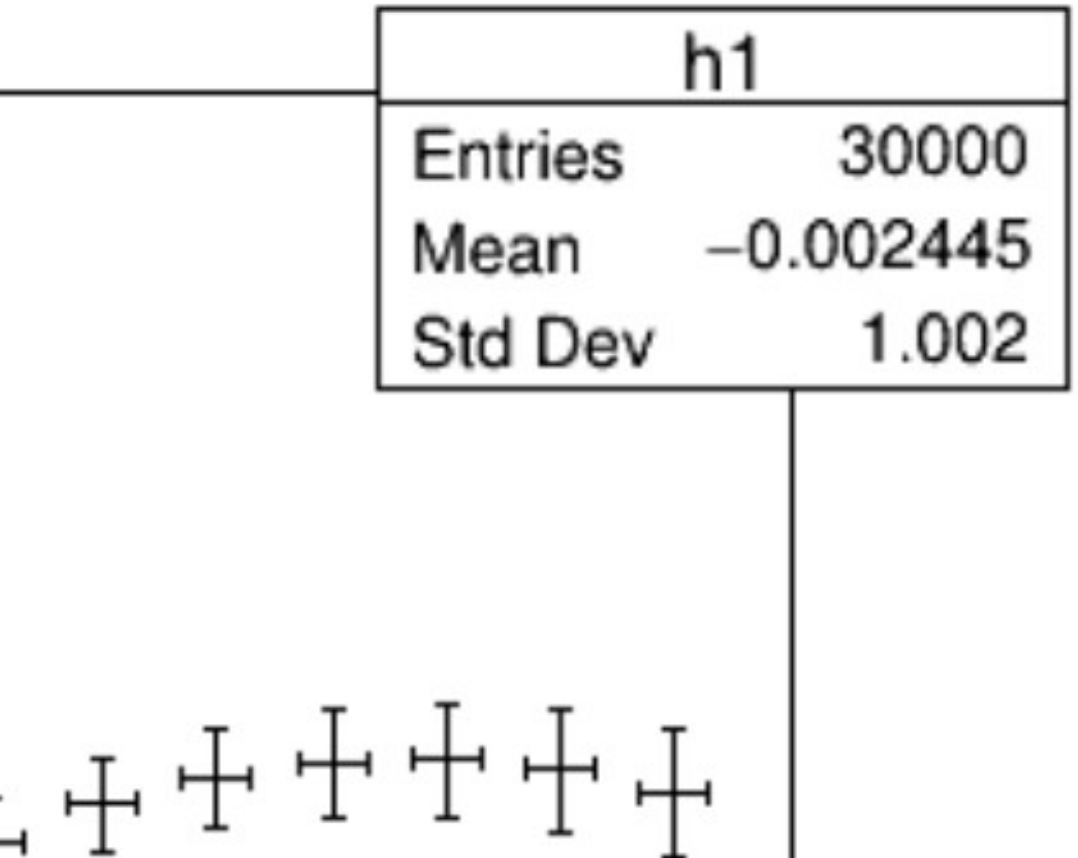
Let's store our `TCanvas` in a graphical file.
But… which format to choose?

- **jpg** is lossy. For technical figures, forget it.
- **png**, **gif** are lossless but pixelwise. If zoomed, see below what happens
- **svg, eps, pdf** are lossless, vector-type. Best to embed in document, but:
  - **svg**: if document is LaTeX, make sure you have the package
  - **eps**: not always openable in your favorite graphics program
  - **pdf**: make sure it doesn't contain extra space below

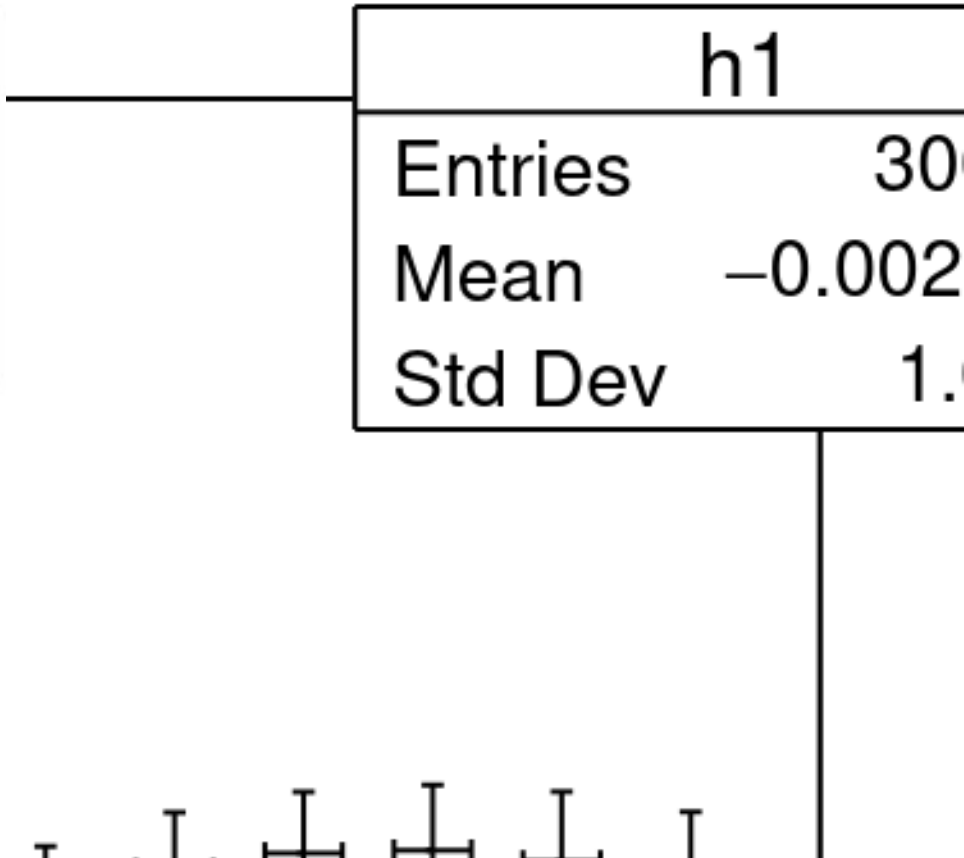


jpg

(png, gif)



(svg, eps, pdf)



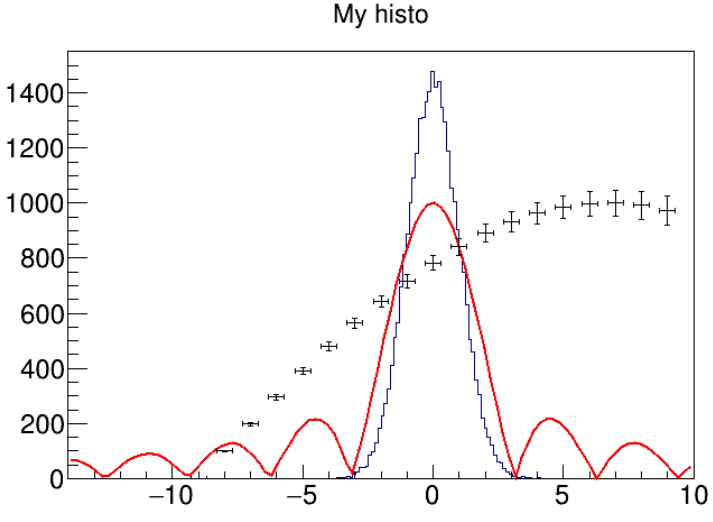
gStyle:

- a container with parameters of current graphical style
- always present during interactive ROOT sessions and macros in interactive mode.
- Technically, pointer to object of **TStyle** class
- by `gStyle→Set ...` and `Get…` methods we can set these parameters and retrieve the current values

For example,

```
root[0] gStyle→SetLabelSize (0.055, "XY" );
root[1] gStyle→SetTextFont  (42);
root[2] gStyle→SetOptStat (0);
root[3] .x firstPlot.C
```

will increase sizes of values at X and Y axes
will change font to Helvetica at moderate precision
will remove the statistics box



some job done, but clearly work ahead:

- No. of divisions on Y axis is too dense
- "*My histo*" is not informative: 2 out of 3 objects are not histos. A legend instead would be better.
- Axis labels missing

Btw. a histogram during its creation inherits the style properties. But if something in `gStyle` is changed later, it will not propagate to that histogram. You need to invoke `hist→UseCurrentStyle()` and redraw your histo.

- Let's now add the **axis titles** (like physics quantities with units).
  However, currently in the bottom and left areas there's no room for inscriptions. We need to:
  - take control over `TCanvas` – by creating it ourselves and placing it under the pointer.

```
TCanvas* c1 = new TCanvas ("c1", "", 800, 600) ;
```

  - reposition the histogram box by steering the margin sizes:

```
c1→SetTopMargin    (0.05);
c1→SetBottomMargin (0.15);
c1→SetRightMargin  (0.04);
c1→SetLeftMargin   (0.16);
```

  - set the axis titles:

```
h→GetXaxis()→SetTitle ( "Position (#mum)" );
h→GetYaxis()→SetTitle ( "Rate (arb. units)" );
```

  - They're now attached to the ends of axes. You may like to center them:

```
h→GetXaxis()→CenterTitle (true);
h→GetYaxis()→CenterTitle (true);
```

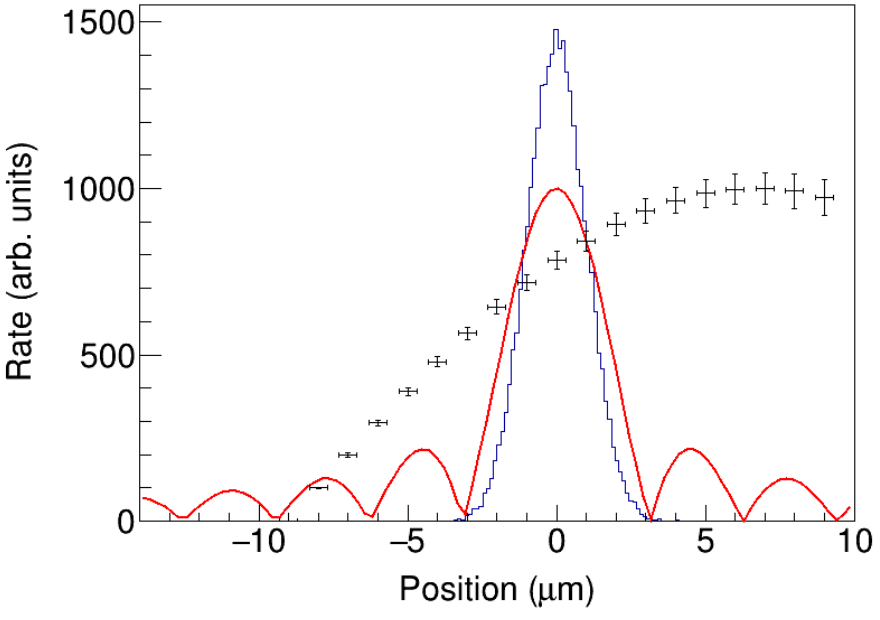
- Let's now enlarge the **font size**, and take care for a nice **space between axis line and title** :

```
h→GetXaxis()→SetTitleSize (0.055);
h→GetYaxis()→SetTitleSize (0.055);
h→GetXaxis()→SetTitleOffset (1.5);
h→GetYaxis()→SetTitleOffset (1.5);
```

- Let us also focus on the **net of divisions** on the Y axis: it seems to be too dense.
  This setting can be changed either per histogram, or globally, by adressing `gStyle`.
  Let's propose some standard setting for both axes:

```
gStyle→SetNdivisions (505, "XY" );
```

So, let's combine all together into the [new macro] and check the effect of our work:



seems better…

but often it's beneficial to describe the presented objects in the legend.

```
int plot_v2 () {
  gStyle→SetOptStat (0);
  gStyle→SetLabelSize  (0.055, "XY");
  gStyle→SetNdivisions ( 505 , "XY");
  gStyle→SetTextFont (42);

  /*** Stage 1: generating some dummy data ***/

  TH1F* h = new TH1F ("h", "", 200, -14, 10);
  h→FillRandom ("gaus", 30000);
  h→GetXaxis()→SetTitle ("Position (#mum)");
  h→GetXaxis()→CenterTitle (true);
  h→GetXaxis()→SetTitleOffset (1.25);
  h→GetXaxis()→SetTitleSize(0.055);
  h→GetYaxis()→SetTitle ("Rate (arb. units)");
  h→GetYaxis()→CenterTitle (true);
  h→GetYaxis()→SetTitleOffset (1.5);
  h→GetYaxis()→SetTitleSize(0.055);

  TF1* f = new TF1 ("f", "1000*abs(sin(x)/x)", -14, 14);

  Int_t i = 0;
  Double_t x[50], y[50], ex[50], ey[50];
  for (Double_t xval = -9; xval ≤ 9; xval++, i++) {
    x[i]  = xval;
    y[i]  = 1000 * sin ( (xval + 9)/10 ) ;
    ex[i] = 0.3;
    ey[i] = (xval+9) * 3 ;
  }

  TGraphErrors* g = new TGraphErrors (i, x, y, ex, ey);

  /*** Stage 2: Data visualization ***/

  TCanvas* c1 = new TCanvas ("c1", "", 800, 600);
  c1→SetTopMargin    (0.05);
  c1→SetBottomMargin (0.15);
  c1→SetRightMargin  (0.04);
  c1→SetLeftMargin   (0.16);

  h→Draw ();
  f→Draw ("same");
  g→Draw ("P");
  return 0;
}
```

To design a legend, we will create the `TLegend` object. It acts as a legend manager.

```
TLegend* leg = new TLegend (0.21, 0.67, 0.48, 0.90, "", "NDC");
```

↳ First 4 arguments are: (X,Y) position of bottom-left and top-right edges.
Option `"NDC"` : positions above are given relative to `TCanvas` coordinates (0,0) → (1,1)

▷ We now have to inform the legend manager about our objects:

```
leg→AddEntry (  h   , "   Experiment" , "f"  );
leg→AddEntry ( "f" , "   Model"      , "l"  );
leg→AddEntry ( "g" , "   Efficiency" , "lep");
```

1st argument: here you place the pointer to the graphical object or give its name.
2nd argument: here you pass the signature of this object
3rd argument: here you inform, how your object should be symbolized:

↳ `"f"` filled box
`"l"` line
`"p"` marker
`"e"` error bar

Btw. if you want to add (inside the legend) just a line of text without a symbol, do this:

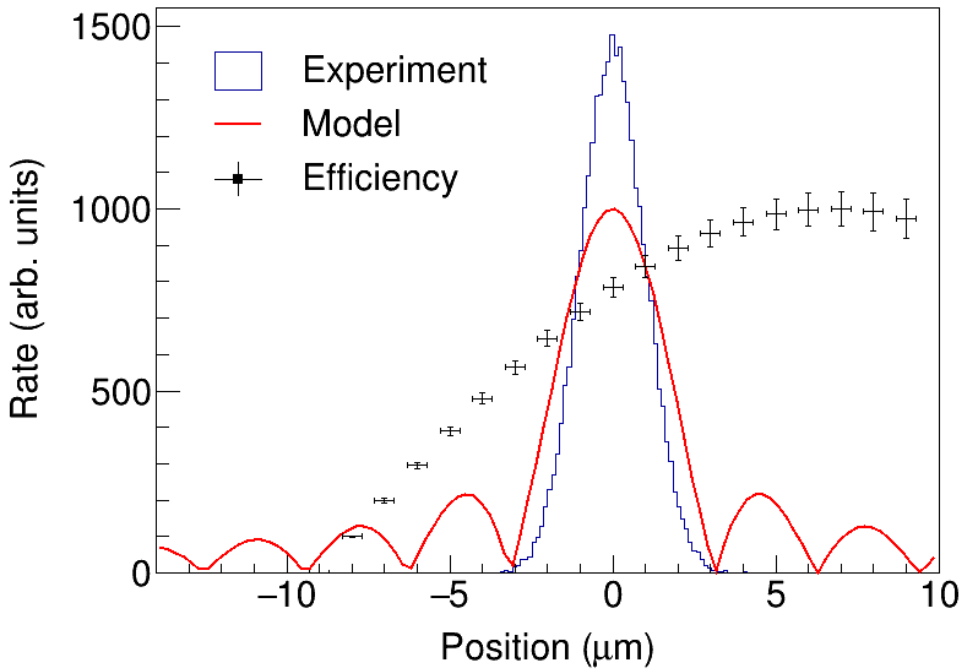
```
leg→AddEntry ( (TObject*) 0 , "Just some text" , "" );
```

▷ After designing the legend, we want to draw it (on the existing figure) :

```
leg→Draw ();
```

▷ And fine-tuning:
We should adjust the font size to that of other texts: `gStyle→SetLegendTextSize (0.055);`
Probably we don't want our legend to be inside a box: `gStyle→SetLegendBorderSize (0);`

Let's implement it in the [macro] and check the outcome:



starts looking not so bad :)

```
int plot_v3 () {
  gStyle→SetOptStat (0);
  gStyle→SetLegendBorderSize (0);
  gStyle→SetLegendTextSize   (0.055);
  gStyle→SetLabelSize  (0.055, "XY");
  gStyle→SetNdivisions ( 505 , "XY");
  gStyle→SetTextFont (42);

  /*** Stage 1: generating some dummy data ***/

  TH1F* h = new TH1F ("h", "", 200, -14, 10);
  h→FillRandom ("gaus", 30000);
  h→GetXaxis()→SetTitle ("Position (#mum)");
  h→GetXaxis()→CenterTitle (true);
  h→GetXaxis()→SetTitleOffset (1.25);
  h→GetXaxis()→SetTitleSize(0.055);
  h→GetYaxis()→SetTitle ("Rate (arb. units)");
  h→GetYaxis()→CenterTitle (true);
  h→GetYaxis()→SetTitleOffset (1.5);
  h→GetYaxis()→SetTitleSize(0.055);

  TF1* f = new TF1 ("f", "1000*abs(sin(x)/x)", -14, 14);

  Int_t i = 0;
  Double_t x[50], y[50], ex[50], ey[50];
  for (Double_t xval = -9; xval ≤ 9; xval++, i++) {
    x[i]  = xval;
    y[i]  = 1000 * sin ( (xval + 9)/10 ) ;
    ex[i] = 0.3;
    ey[i] = (xval+9) * 3 ;
  }

  TGraphErrors* g = new TGraphErrors (i, x, y, ex, ey);

  /*** Stage 2: Data visualization ***/

  TCanvas* c1 = new TCanvas ("c1", "", 800, 600);
  c1→SetTopMargin    (0.05);
  c1→SetBottomMargin (0.15);
  c1→SetRightMargin  (0.04);
  c1→SetLeftMargin   (0.16);

  h→Draw ();
  f→Draw ("same");
  g→Draw ("P");
  return 0;
}
```

- Sometimes we need to write some text inside.
  Let's pretend that this figure is one of series of panels and we want to place "(A)" somewhere.
  We can use either **TText** or the **TLatex** object (manager of Latex-ish inscriptions). Let's take the latter one:

```
TLatex l;
l.SetNDC (1);
```

  We should align the text font and style with those of the others:

```
l.SetTextSize (0.055);
l.SetTextFont (42);
```

  Now let's draw the panel symbol:

```
l.DrawLatex (0.82, 0.845, "(A)" );
```

- We can also amplify the visibility of objects by playing with line color, style, marker style etc.
  (within good proportions) . Let's add it here and there, e.g.:

  - to the histogram:

```
h→SetLineWidth (2);
h→SetLineColor (kBlack);
h→SetFillColor (19);
```

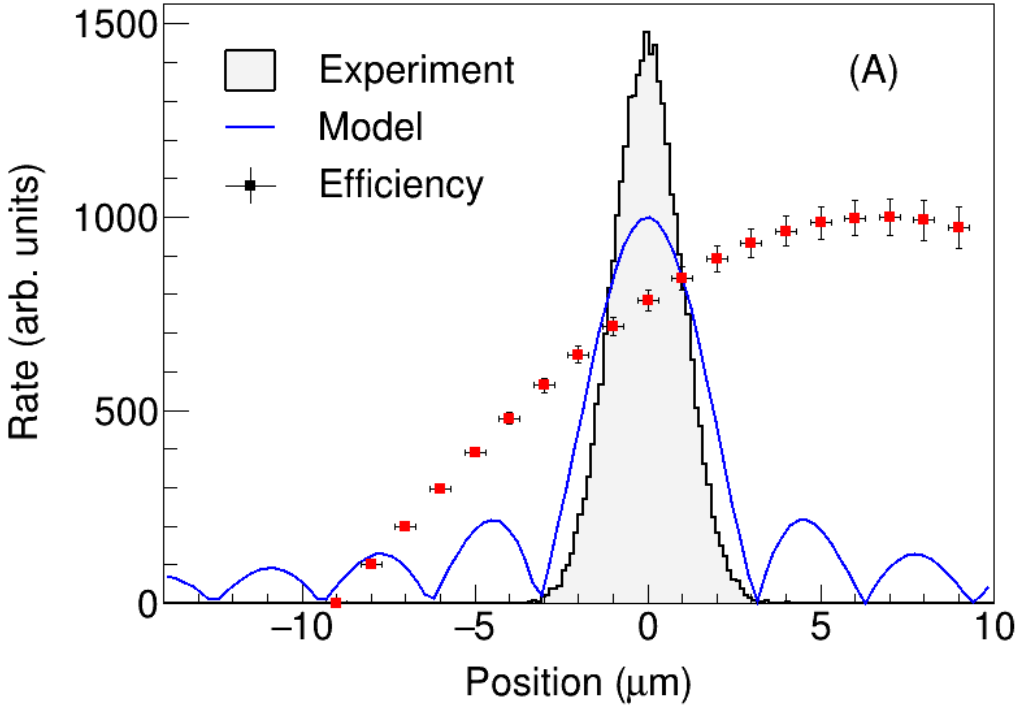
  - to the function:

```
f→SetLineColor (kBlue);
```

  - to the graph:

```
g→SetMarkerStyle (21);
g→SetMarkerColor (kRed);
```

Finally, that's how [our macro] looks and the figure we've now produced:



isn't it more readable and informative?

Ugh… but how long it takes to write such a code*!*

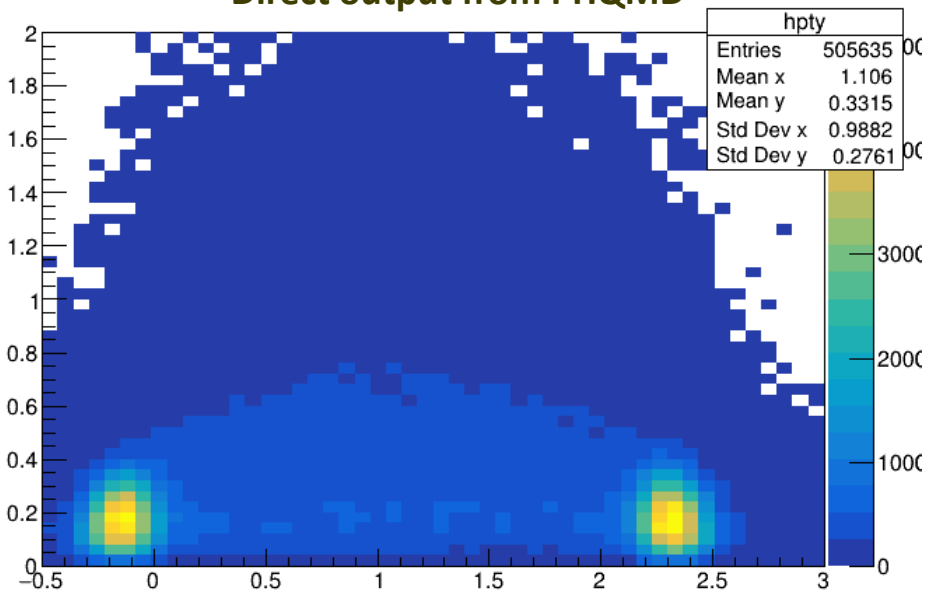
↳ No, just keep this macro home,
and copy-paste its parts for good starting points 😊

```
int plot_v4 () {
  gStyle→SetOptStat (0);
  gStyle→SetLegendBorderSize (0);
  gStyle→SetLegendTextSize   (0.055);
  gStyle→SetLabelSize  (0.055, "XY");
  gStyle→SetNdivisions ( 505 , "XY");
  gStyle→SetTextFont (42);

  /*** Stage 1: generating some dummy data ***/

  TH1F* h = new TH1F ("h", "", 200, -14, 10);
  h→FillRandom ("gaus", 30000);
  h→SetLineWidth (2);
  h→SetLineColor (kBlack);
  h→SetFillColor (19);
  h→GetXaxis()→SetTitle ("Position (#mum)");
  h→GetXaxis()→CenterTitle (true);
  h→GetXaxis()→SetTitleOffset (1.25);
  h→GetXaxis()→SetTitleSize(0.055);
  h→GetYaxis()→SetTitle ("Rate (arb. units)");
  h→GetYaxis()→CenterTitle (true);
  h→GetYaxis()→SetTitleOffset (1.5);
  h→GetYaxis()→SetTitleSize(0.055);

  TF1* f = new TF1 ("f", "1000*abs(sin(x)/x)", -14, 14);
  f→SetLineColor (kBlue);

  Int_t i = 0;
  Double_t x[50], y[50], ex[50], ey[50];
  for (Double_t xval = -9; xval ≤ 9; xval++, i++) {
    x[i]  = xval;
    y[i]  = 1000 * sin ( (xval + 9)/10 ) ;
    ex[i] = 0.3;
    ey[i] = (xval+9) * 3 ;
  }

  TGraphErrors* g = new TGraphErrors (i, x, y, ex, ey);
  g→SetMarkerStyle (21);
  g→SetMarkerColor (kRed);

  /*** Stage 2: Data visualization ***/

  TCanvas* c1 = new TCanvas ("c1", "", 800, 600);
  c1→SetGrid (0, 0);
  c1→SetTopMargin    (0.05);
  c1→SetBottomMargin (0.15);
  c1→SetRightMargin  (0.04);
  c1→SetLeftMargin   (0.16);

  h→Draw ();
  f→Draw ("same");
  g→Draw ("P");

  TLegend* leg = new TLegend (0.21, 0.67, 0.48, 0.90, "", "nbNDC");
  leg→AddEntry ( h , "   Experiment" , "f"  );
  leg→AddEntry ("f", "   Model"      , "l"  );
  leg→AddEntry ("g", "   Efficiency" , "lep");
  leg→Draw ();

  TLatex l;
  l.SetNDC (1);
  l.SetTextSize (0.055);
  l.SetTextFont (42);
  l.DrawLatex (0.82, 0.845, "(A)" );
  return 0;
}
```

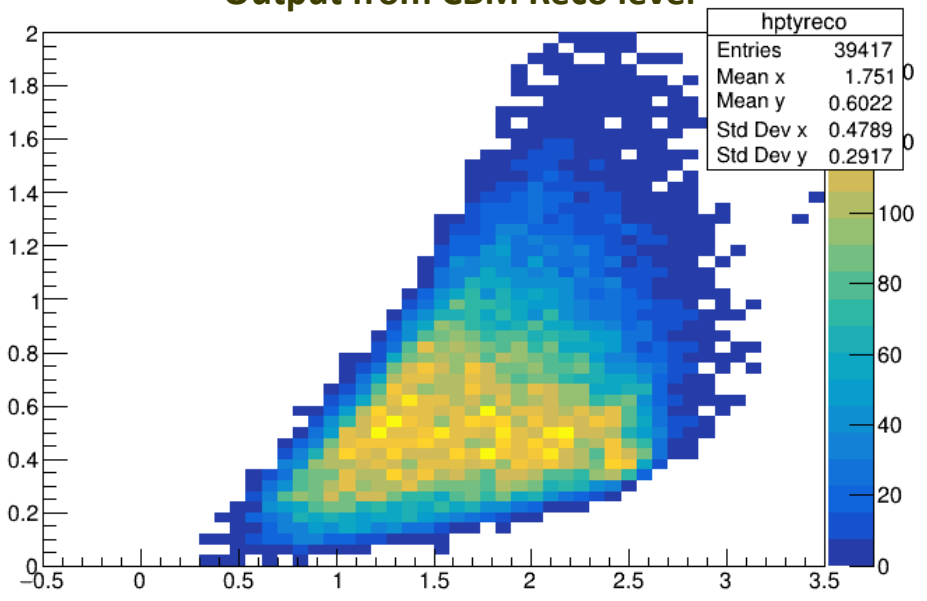
Presenting the phase space plots (e.g. pt-y plots)

Example: PHQMD simulation of Au+Au @ T_{Beam} = 3.25A GeV , b = [0, 16] fm, pt-y distribution of protons.

Direct output from PHQMD



Output from CBM Reco level



Let's put esthetics aside. Missing things: **curves of constant ϑ and p in Lab frame** and **midrapidity line**.
It's like a **map without a cartographic grid**. These features allow to have a first-order understanding of the plot.

$$\begin{cases} y = atanh\,\beta \\ \beta = \dfrac{p_z}{E} \\ \tan\theta = \dfrac{p_T}{p_z} \end{cases}$$

$$\begin{cases} p_T\left(y;\ \theta = const\right) = \dfrac{m \cdot \sinh y\ \tan\theta}{\sqrt{1 - \left(\sinh y\ \tan\theta\right)^2}} \\ p_T\left(y;\ p = const\right) = \dfrac{\sqrt{p^2 - \left(m \cdot \sinh y\right)^2}}{\cosh y} \end{cases}$$

- To apply it, we first need to embed the relevant function in the code.
  Here, functions (with some fuses).

```
Double_t pt_y_ptot (Double_t* xarg, Double_t *par) {
  Double_t y   = xarg[0] ,
         ptot = par[0] ,
         mass = par[1] ;

  Double_t u = pow(ptot, 2.) - pow(mass*sinh(y) , 2.);

  return (u > 0.) ? sqrt(u) / cosh(y) : 0. ;
}
```

↪ Input parameters:

par[0] = momentum (*const*)
par[1] = mass (*const*)

```
Double_t pt_y_theta (Double_t* xarg, Double_t* par) {
  Double_t y = xarg[0] ,
           theta = par[0] ,
           mass  = par[1] ;

  Double_t u = sinh(y) * tan ( theta * M_PI/180. ) ,
           pt_value =
           (u < 1.) ? mass * u / sqrt(1. - u*u) : 0.;

  return (pt_value > 0. ) ? pt_value : 10. ;
}
```

↪ Input parameters:

par[0] = theta (*const*)
par[1] = mass (*const*)

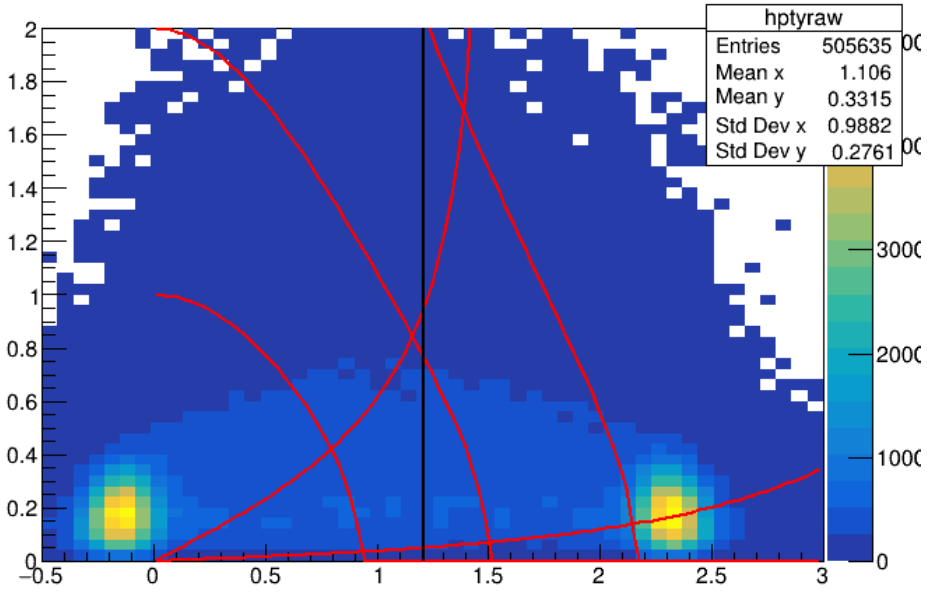
▷ Next, in the `main` function we need to create 1 × `TF1` object per curve, set up parameters and draw. E.g.

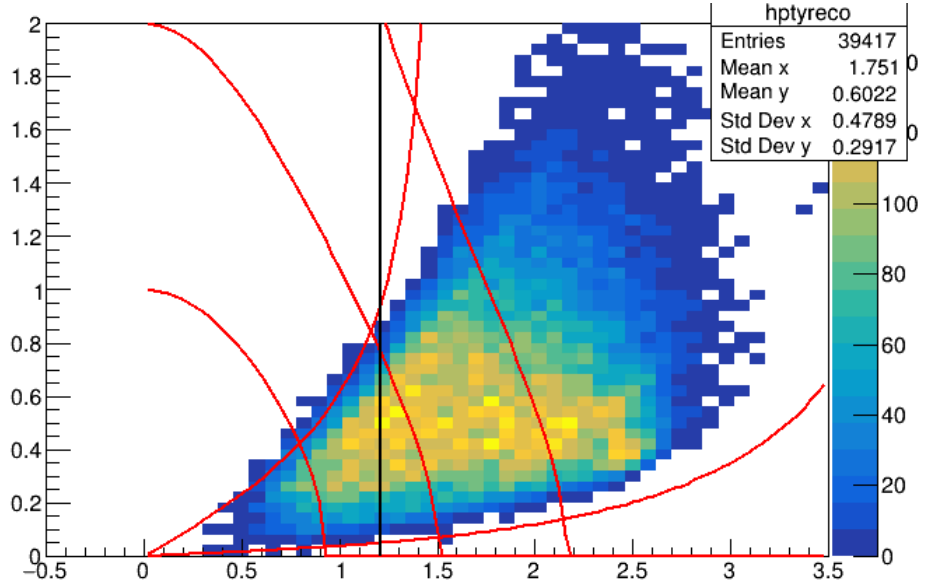
```
TF1* fpty_th1 = new TF1 ("fptyth1", pt_y_theta, 0, 10, 2);
fpty_th1->SetParameters (25. , 0.939);
fpty_th1->Draw ("same");
```

▷ Plus, we should draw the midrapidity line. Assume T_{Beam} = 3.25 GeV. Put Y_{NN} calculation into your code:

```
Float_t Tbeam = 3.25 , MNuc = 0.939, Ebeam = Tbeam + MNuc,
        pbeam = sqrt (Ebeam * Ebeam + MNuc * MNuc) ,
        Bbeam = pbeam / (Ebeam + MNuc),
        Ynn   = atanh (Bbeam);
```

Let’s put it into [the code]. We’ll use data from [raw] and [reco] stages.



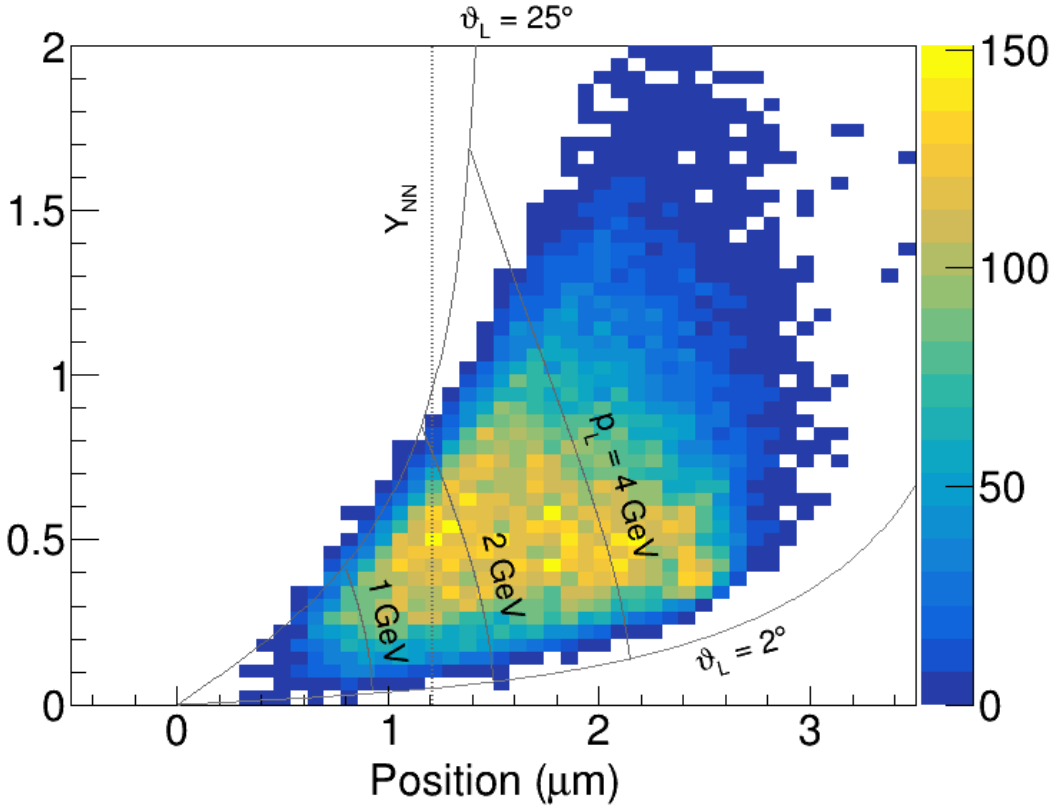



```
Double_t pt_y_theta (Double_t* xarg, Double_t* par)  {
  Double_t y = xarg[0] ,
           theta = par[0] ,
           mass  = par[1] ;
  Double_t u = sinh(y) * tan ( theta * M_PI/180. ) ,
           pt_value =
           (u < 1.) ? mass * u / sqrt(1. - u*u) : 0. ;

  return (pt_value > 0. ) ? pt_value : 10. ;
}

Double_t pt_y_ptot (Double_t* xarg, Double_t *par) {
  Double_t y  = xarg[0] ,
         ptot = par[0] ,
         mass = par[1] ;
  Double_t u = pow(ptot, 2.) - pow(mass*sinh(y) , 2.);
  return (u > 0.) ? sqrt(u) / cosh(y) : 0. ;
}

int pty_plot (int optInput = 1, int optDraw = 1) {
  Float_t Tb = 3.25 , mnuc = 0.939, Ebeam = Tb + mnuc,
          pbeam = sqrt (Ebeam*Ebeam + mnuc*mnuc) ,
          Bbeam = pbeam/(Ebeam+mnuc), Ynn = atanh (Bbeam);
  TFile* f;
  TH2F * h;

  if (optInput == 1) {
    f = new TFile ("phqmd_auau3.25GeV_u2t_prot.root");
    h = new TH2F ("hptyraw", "", 50, -0.5, 3, 50, 0, 2);
    TTree* t = (TTree*) f->Get ("t");
    t->Project ("hptyraw", "pt:ylab") ;
  }
  else if (optInput == 2) {
    f = new TFile ("phqmd_auau3.25GeV_cbmreco_prot.root");
    h = (TH2F*) f->Get ("prot_pty");
    h->SetName ("hptyreco");
  }
  h->Draw ("colz");

  if (optDraw == 0)
    return 0;
  else {
    TF1* fpty_th1 = new TF1 ("fptyth1", pt_y_theta, 0, 10, 2);
    fpty_th1->SetParameters (2. , 0.939);
    fpty_th1->Draw ("same");

    TF1* fpty_th2 = new TF1 ("fptyth2", pt_y_theta, 0, 10, 2);
    fpty_th2->SetParameters (25. , 0.939);
    fpty_th2->Draw ("same");

    TF1* fpty_p0 = new TF1 ("fptyp0", pt_y_ptot , 0, 10, 2);
    fpty_p0->SetParameters (1.0 , 0.939);
    fpty_p0->Draw ("same");

    TF1* fpty_p1 = new TF1 ("fptyp1", pt_y_ptot , 0, 10, 2);
    fpty_p1->SetParameters (2.0 , 0.939);
    fpty_p1->Draw ("same");

    TF1* fpty_p2 = new TF1 ("fptyp2", pt_y_ptot , 0, 10, 2);
    fpty_p2->SetParameters (4.0 , 0.939);
    fpty_p2->Draw ("same");
  }
  TLine* lYnn = new TLine (Ynn, 0., Ynn, 2.);
  lYnn->SetLineWidth (2);
  lYnn->Draw ("same");
  return 0;
}
```

- Let's finalize our work:
  - curves should not intersect
  - they should be labelled
  - they style, color and width should harmonize with the rest

- The [final code] is a bit too long to show, but you can look it up online.

Here you can see the result.



Hope you like it. Use the features from this code or invent yours.

Let's remember: ***readability and esthetics are important for effective dissemination of our work*** 😊