ROOT Basic Tools: histograms and graphs

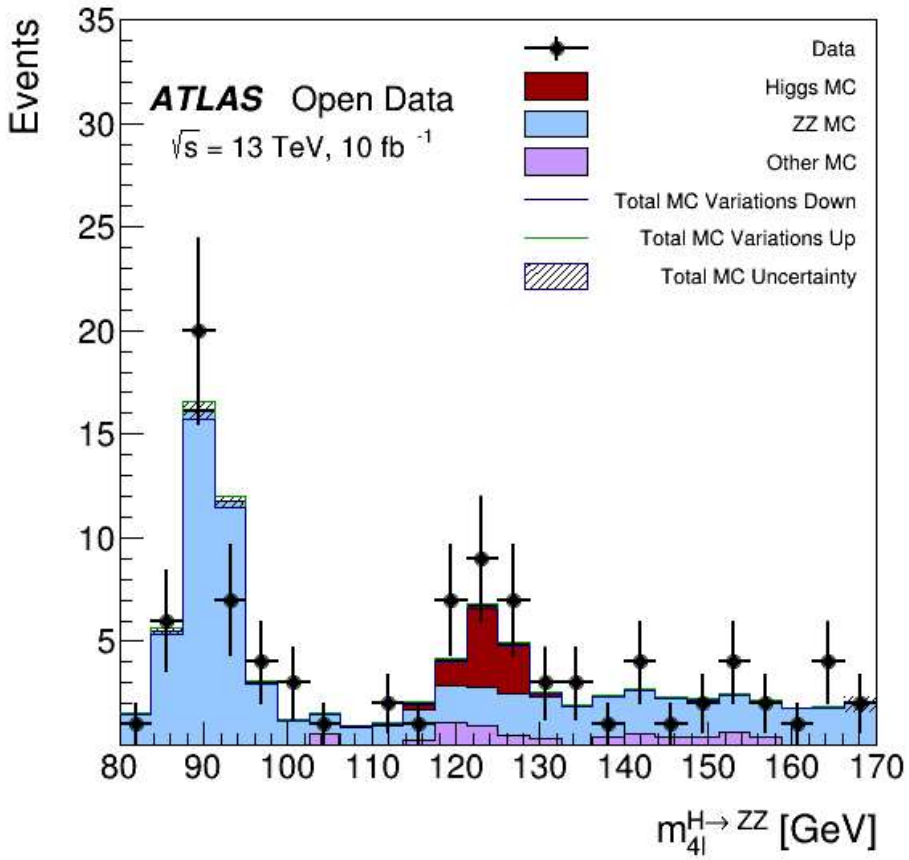
ROOT histograms

Histogram class documentation

ROOT has powerful histogram objects that, among other features, let you produce complex plots and perform fits of arbitrary functions.

Below is an example histogram that can be obtained using one of our tutorials: Higgs to Four Leptons.

`TH1D` is a 1D histogram with floating point double precision y-axis, `TH2I` is a 2D histogram with Integer y-axis, etc.



To have something to play with, let's quickly fill a histogram with 5000 normally distributed values:

In [1]:
```
import ROOT
h = ROOT.TH1D(name="h", title="My histo", nbinsx=100, xlow=-5, xup=5)


h.FillRandom("gaus", ntimes=5000)
```

```
Welcome to JupyROOT 6.30/04
```

To check the full documentation you can always refer to https://root.cern/doc/master (and then switch to the documentation for your particular ROOT version with the drop-down menu at the top of the page).

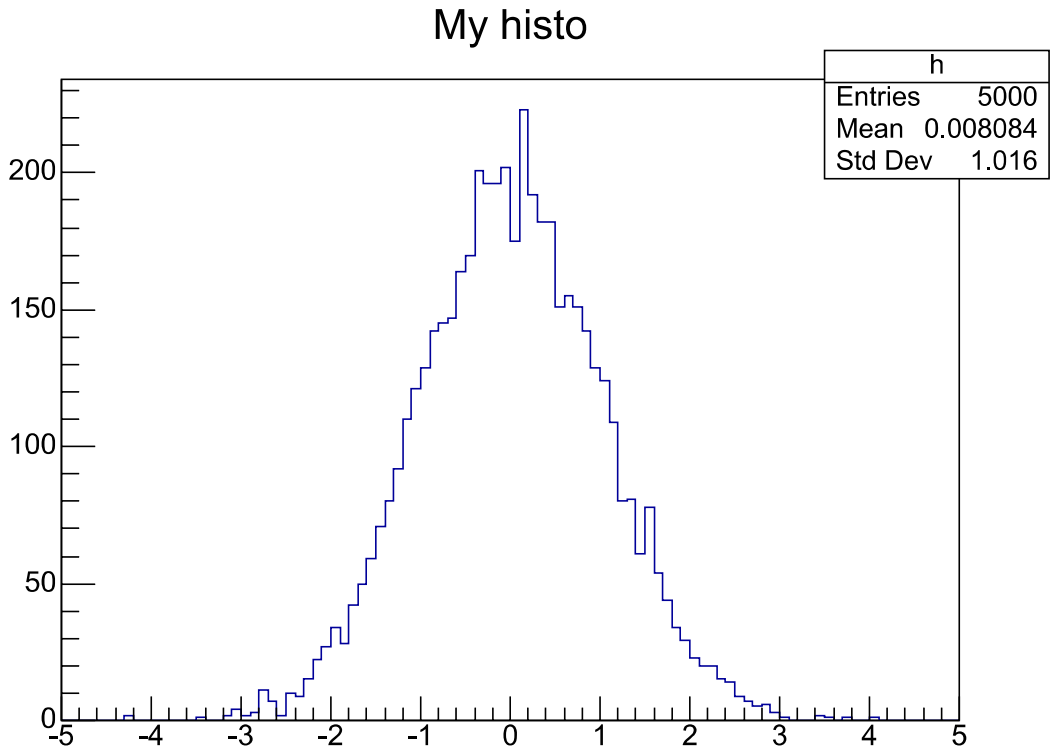
Drawing a histogram

Drawing options documentation

The link above contains the documentation for the histogram drawing options.

In a notebook, we want to use the `%jsroot on` magic and explicitly draw a `TCanvas`.

In [2]:
```
%jsroot on
c = ROOT.TCanvas()
#h.SetLineColor(ROOT.kBlue)
#h.SetFillColor(ROOT.kBlue)
#h.GetXaxis().SetTitle("value")
#h.GetYaxis().SetTitle("count")
#h.SetTitle("My histo with latex: p_{t}, #eta, #phi")
h.Draw() # draw the histogram on the canvas
c.Draw() # draw the canvas on the screen
```



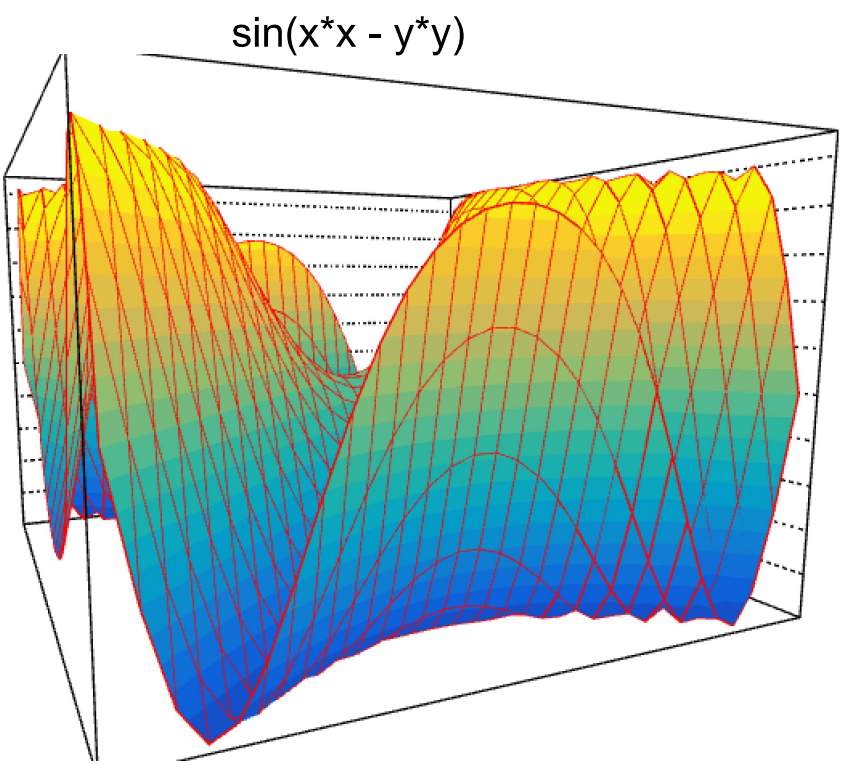
ROOT functions

The type that represents an arbitrary one-dimensional mathematical function in ROOT is TF1.
Similarly, TF2 and TF3 represent 2-dimensional and 3-dimensional functions.

As an example, let's define and plot a simple surface:

In [3]:
```
f2 = ROOT.TF2("f2", "sin(x*x - y*y)", xmin=-2, xmax=2, ymin=-2, ymax=2)
```

In [4]:
```
c = ROOT.TCanvas()
f2.Draw("surf1") # to get a surface instead of the default contour plot
c.Draw()
```



Fitting a histogram

Let's see how to perform simple histogram fits of arbitrary functions. We will need a `TF1` that represents the function we want to use for the fit.

This time we define our `TF1` as a C++ function (note the usage of the `%%cpp` magic to define some C++ inline). Here we define a simple gaussian with scale and mean parameters (`par[0]` and `par[1]` respectively):

In [5]:
```
%%cpp

double gaussian(double *x, double *par) {
    return par[0]*TMath::Exp(-TMath::Power(x[0] - par[1], 2.) / 2.)
           / TMath::Sqrt(2 * TMath::Pi());
}
```

The function signature, that takes an array of coordinates and an array of parameters as inputs, is the generic signature of functions that can be used to construct a `TF1` object:

In [6]:
```
fitFunc = ROOT.TF1("fitFunc", ROOT.gaussian, xmin=-5, xmax=5, npar=2)
```

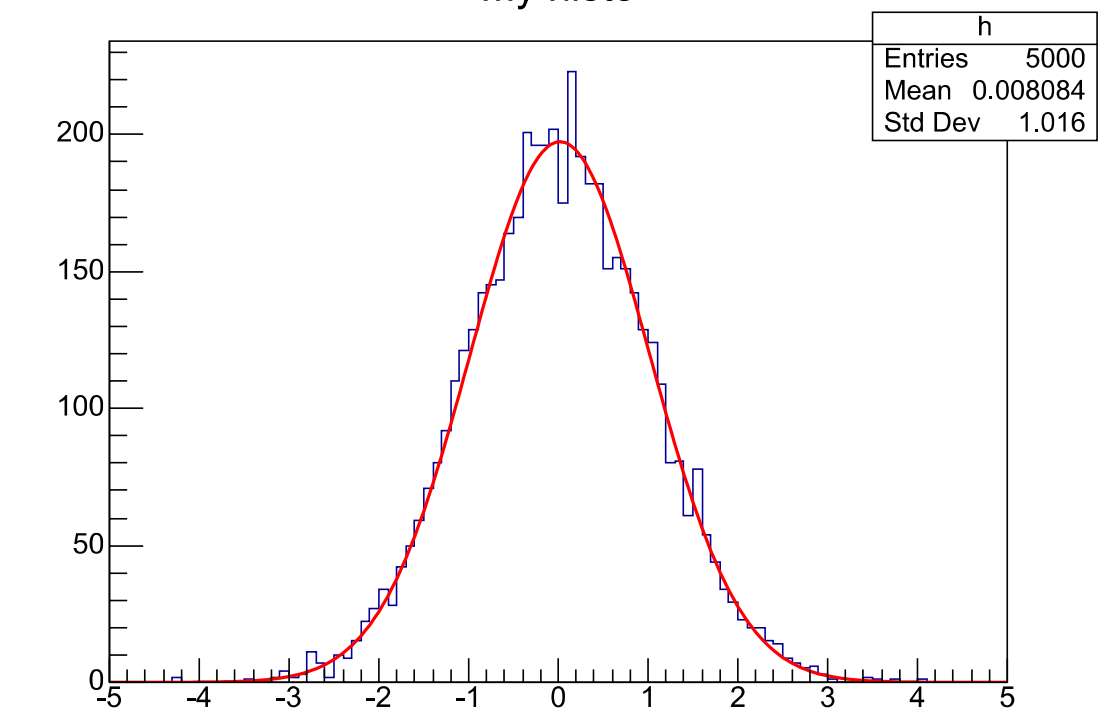
Now we fit our `h` histogram with `fitFunc` :

In [7]:
```
res = h.Fit(fitFunc, "S") # the "S" option makes the function return a fit result object
```

```
****************************************
Minimizer is Minuit2 / Migrad
Chi2                      =      57.2302
NDf                       =           67
Edm                       =  1.34911e-09
NCalls                    =           44
p0                        =      494.933   +/-   7.03982
p1                        =    0.0173689   +/-   0.0142894
```

Drawing the histogram now automatically also shows the fitted function:

In [8]:
```
c2 = ROOT.TCanvas()
h.Draw()
c2.Draw()
```

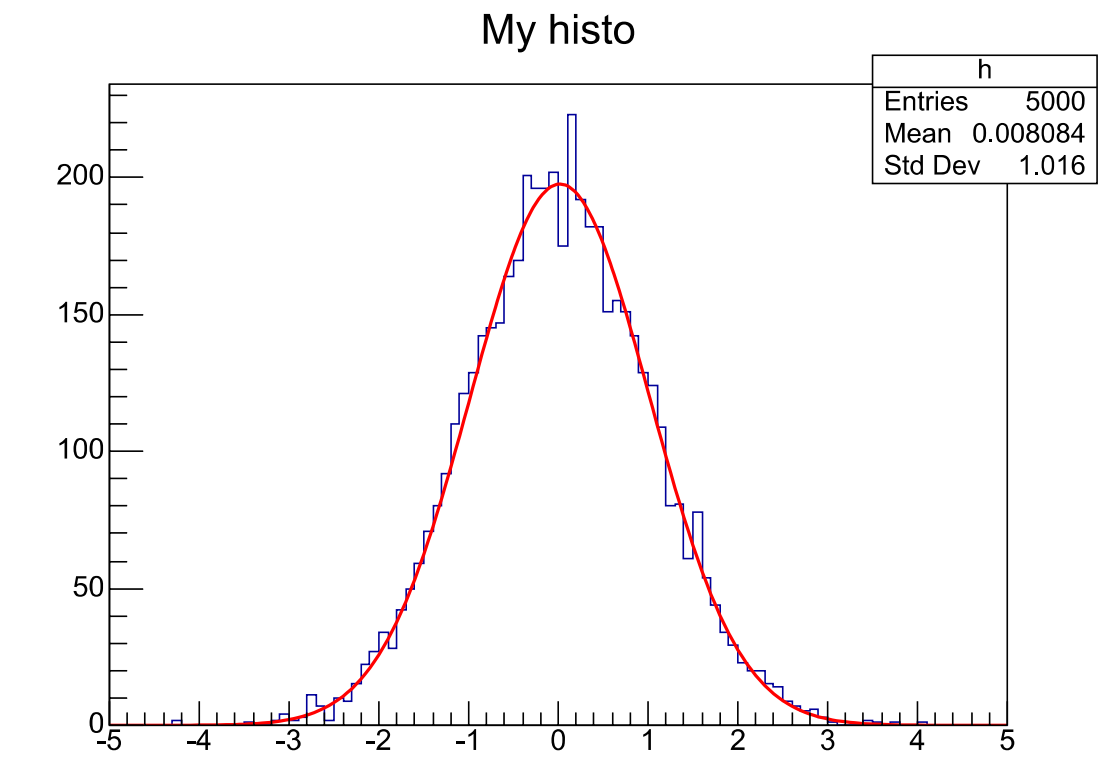


For the particular case of a gaussian fit, we could also have used the built-in `"gaus"` function, as we did when we called `FillRandom` (for the full list of supported expressions see here):

In [9]:
```
res = h.Fit("gaus", "S")
```

```
****************************************
Minimizer is Minuit2 / Migrad
Chi2                      =      57.2152
NDf                       =           66
Edm                       =  9.31601e-06
NCalls                    =           55
Constant                  =      197.704   +/-   3.46748
Mean                      =    0.0172556   +/-   0.014353
Sigma                     =      0.99873   +/-   0.0103909      (limited)
```

In [10]:
```
c3 = ROOT.TCanvas()
h.Draw()
c3.Draw()
```



For more complex binned and unbinned likelihood fits, check out RooFit, a powerful data modelling framework integrated in ROOT.

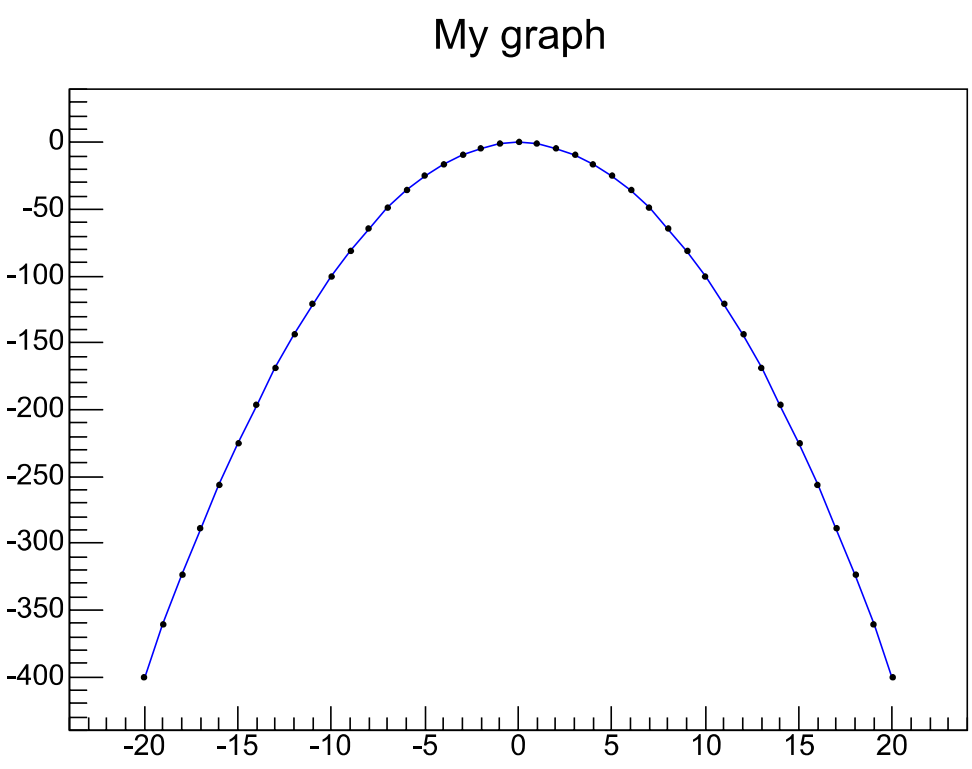
ROOT graphs

TGraph is a type useful for scatter plots.

Their drawing options are documented here.

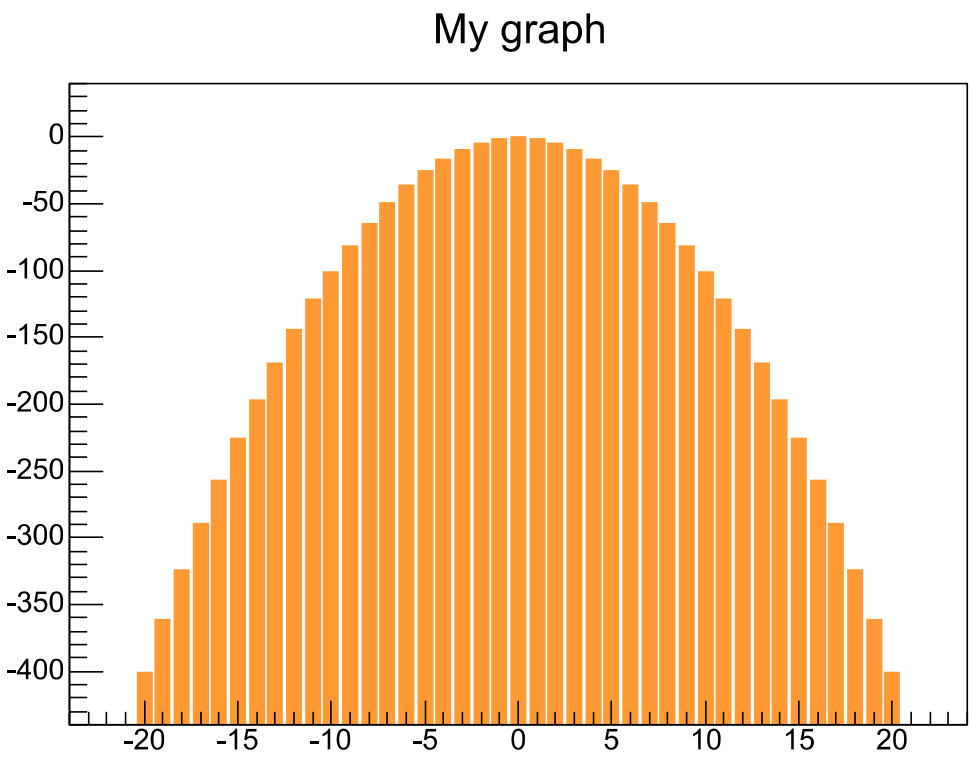
Like for histograms, the aspect of `TGraph`s can be greatly customized, they can be fitted with custom functions, etc.

In [11]:
```
g = ROOT.TGraph()

for x in range(-20, 21):
    y = -x*x
    g.AddPoint(x, y)

c4 = ROOT.TCanvas()
g.SetMarkerStyle(7)
g.SetLineColor(ROOT.kBlue)
g.SetTitle("My graph")
g.Draw()
c4.Draw()
```



The same graph can be displayed as a bar plot:

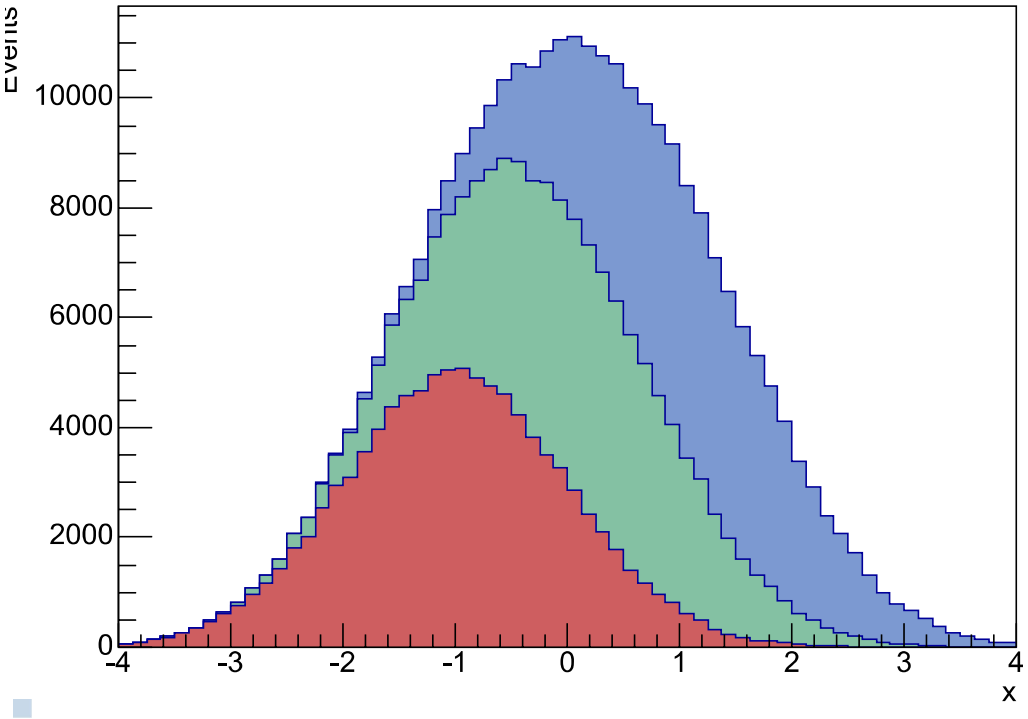
In [12]:
```
c5 = ROOT.TCanvas()
g.SetTitle("My graph")
g.SetFillColor(ROOT.kOrange + 1) # base colors can be tweaked by adding/subtracting values to them
g.Draw("AB1")
c5.Draw()
```



Plot example: histogram stack

In HEP, we often plot stacked histograms, for example to show the contributions of different processes. This can be done with THStack.

In [13]:
```
f1 = ROOT.TF1("f1", "gaus", -4.0, 4.0)

histos = [ROOT.TH1D(f"h{i}", "x", 64, -4.0, 4.0) for i in range(3)]

hs = ROOT.THStack("hs","")
hs.SetTitle(";x;Events")

colors = [46, 30, 38]

for i in range(len(histos)):
    h = histos[i]
    f1.SetParameters(1.0, i - 1, 1.0)
    h.FillRandom("f1", 100000)
    h.SetFillColor(colors[i])
    hs.Add(h)

c6 = ROOT.TCanvas()
hs.Draw()
c6.Draw()
```



Plot example: efficiency curves

Another common workflow is to draw efficiency curves with TEfficiency, which also gives uncertainties.

In [16]:
```
h_pass = ROOT.TH1D("h_pass", "My histogram", 50, 0, 100.0)
h_total = ROOT.TH1D("h_total", "My histogram", 50, 0, 100.0)

f_gaus = ROOT.TF1("f_gaus", "gaus", 0, 100.0)

f_gaus.SetParameters(1.0, 56.0, 20.0)
h_pass.FillRandom("f_gaus", 40000)
h_pass.SetLineColor(ROOT.kRed)
f_gaus.SetParameters(1.0, 50.0, 20.0)
h_total.FillRandom("f_gaus", 100000)
```

```
Warning in <TROOT::Append>: Replacing existing TH1: h_pass (Potential memory leak).
Warning in <TROOT::Append>: Replacing existing TH1: h_total (Potential memory leak).
```

In [17]:
```
teff = ROOT.TEfficiency(h_pass,h_total)

c7 = ROOT.TCanvas("rf101_basics", "rf101_basics", 800, 400)
c7.Divide(2)
c7.cd(1)
```

```
h_total.Draw()
h_pass.Draw("SAME")
c7.cd(2)
teff.Draw()
c7.Draw()
```