ROOT Tutorial

Simulation and fitting in ROOT

Attilio Andreazza

Università di Milano and INFN

Caterina Doglioni

Université de Genève

Outline

- Today is my personal opinion of what should be in the toolbox of every physicist!

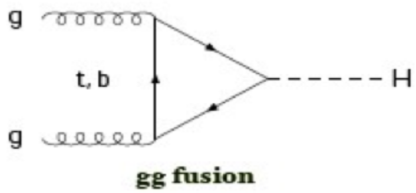
- Making simulations:
  - Random number generation in RooT
  - Build the TTree we will use in the hands-on session
- Fitting a model to data
- ...and how to do all of that in a simpler way using RooFit
  - Or more complicated way, it is a matter or taste...

- Macros for today exercises in `RootTutorial2.zip` in the indico page of the lecture.

What we want to learn

- **How the Higgs boson decay in two photons looks like in a LHC detector?**
- Produced gluon-gluon fusion with gluons, carring fractions x_a and x_b of protons 4-momentum

$$p_a = x_a\left(p_p, 0, 0, p_p\right)$$
$$p_b = x_b\left(p_p, 0, 0, -p_p\right)$$
$$p_H = p_a + p_b = \left((x_a + x_b)p_p, 0, 0, (x_a - x_b)p_p\right)$$



gg fusion

  - It is a narrow resonance: its mass is well defined $m_H^2 = p_H^2 = x_a x_b (4p_p^2) = x_a x_b s$
  - It has rapidity $y = \frac{1}{2}\ln\frac{E+p_z}{E-p_z} = \frac{1}{2}\ln\frac{x_a}{x_b}$

- It decays into two γ's isotropically distributed $\frac{1}{N}\frac{dN}{d\Omega^*} = \frac{1}{4\pi} \Rightarrow \frac{1}{N}dN = \frac{d\phi^*}{2\pi}\frac{d\cos\theta^*}{2}$
- Each photon is observed if:
  - With a finite energy resolution
  - If it is within the geometrical acceptance (|η|<2.4 for ATLAS)
  - If it passes acceptance cuts (p_T>40 GeV for leading, p_T>30 GeV for sub-leading)

TRandom

- Class **TRandom** is the basic random number generator in ROOT
  http://root.cern.ch/root/html/TRandom.html
  - More refined generators **TRandom2**, **TRandom3** inherit from **TRandom** ➔ same interface

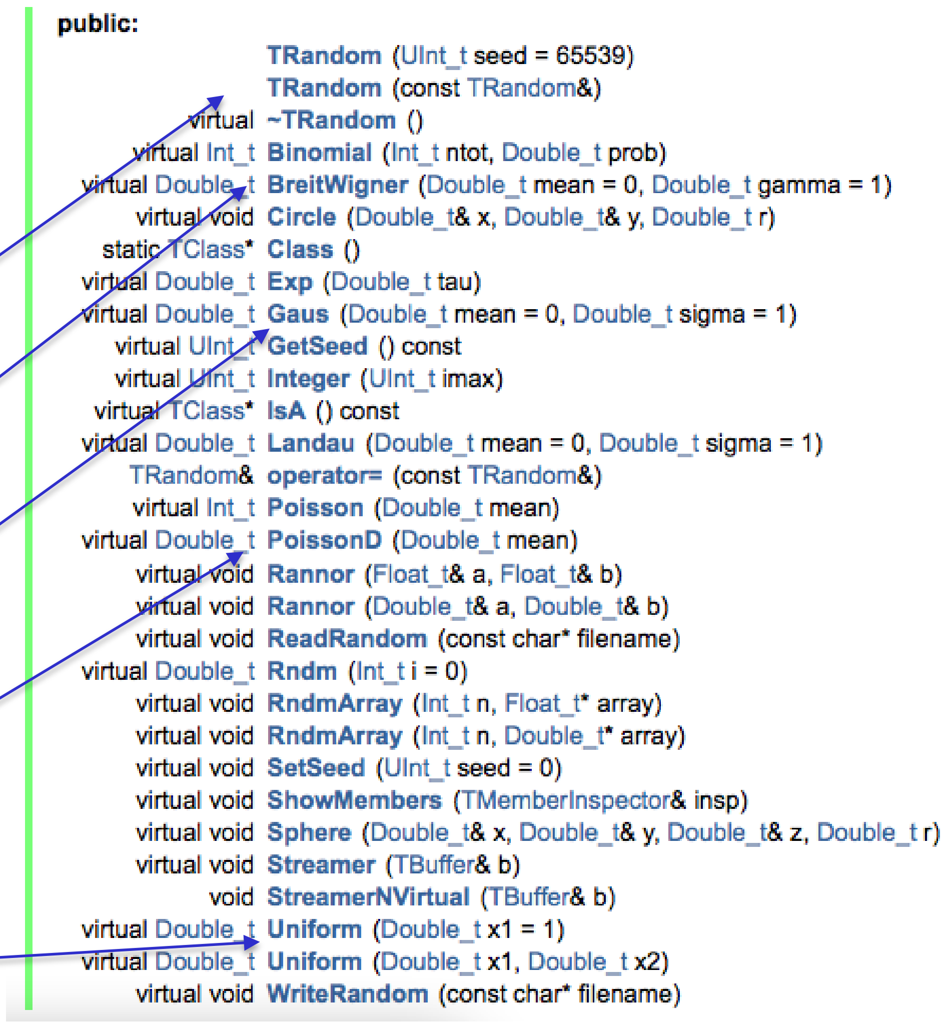
Constructor: argument is seed. Seed=0: use system clock [*s*]

BreitWigner (resonance) distribution

Gaussian distribution

Poisson distribution

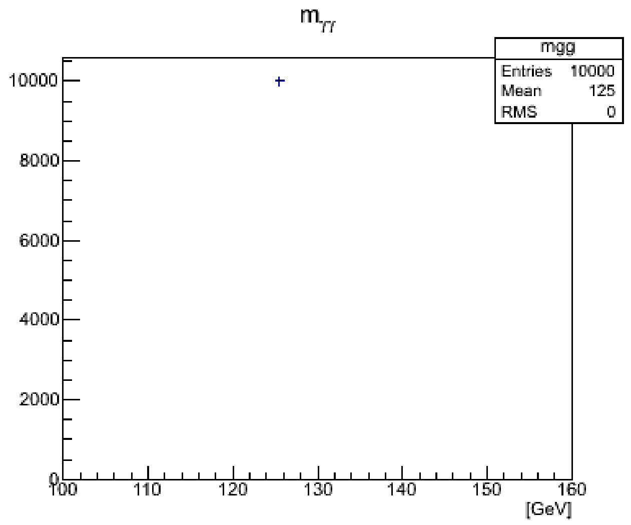
Uniformt distribution

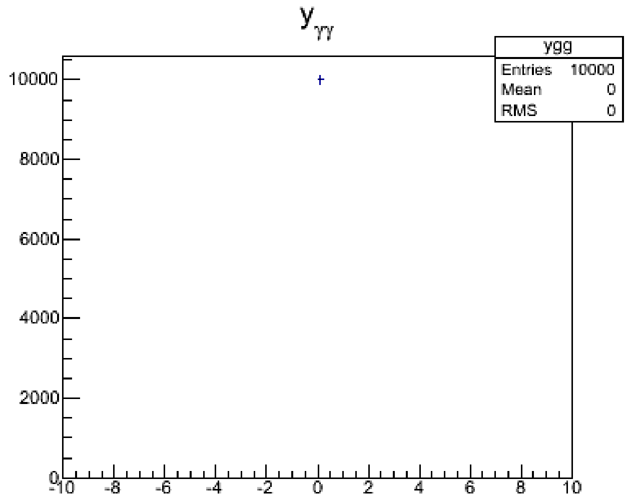


```
public:
                          TRandom (UInt_t seed = 65539)
                          TRandom (const TRandom&)
                 virtual  ~TRandom ()
           virtual Int_t  Binomial (Int_t ntot, Double_t prob)
        virtual Double_t  BreitWigner (Double_t mean = 0, Double_t gamma = 1)
            virtual void  Circle (Double_t& x, Double_t& y, Double_t r)
          static TClass*  Class ()
        virtual Double_t  Exp (Double_t tau)
        virtual Double_t  Gaus (Double_t mean = 0, Double_t sigma = 1)
          virtual UInt_t  GetSeed () const
          virtual UInt_t  Integer (UInt_t imax)
         virtual TClass*  IsA () const
        virtual Double_t  Landau (Double_t mean = 0, Double_t sigma = 1)
                TRandom&  operator= (const TRandom&)
           virtual Int_t  Poisson (Double_t mean)
        virtual Double_t  PoissonD (Double_t mean)
            virtual void  Rannor (Float_t& a, Float_t& b)
            virtual void  Rannor (Double_t& a, Double_t& b)
            virtual void  ReadRandom (const char* filename)
        virtual Double_t  Rndm (Int_t i = 0)
            virtual void  RndmArray (Int_t n, Float_t* array)
            virtual void  RndmArray (Int_t n, Double_t* array)
            virtual void  SetSeed (UInt_t seed = 0)
            virtual void  ShowMembers (TMemberInspector& insp)
            virtual void  Sphere (Double_t& x, Double_t& y, Double_t& z, Double_t r)
            virtual void  Streamer (TBuffer& b)
                    void  StreamerNVirtual (TBuffer& b)
        virtual Double_t  Uniform (Double_t x1 = 1)
        virtual Double_t  Uniform (Double_t x1, Double_t x2)
            virtual void  WriteRandom (const char* filename)
```

Higgs decay

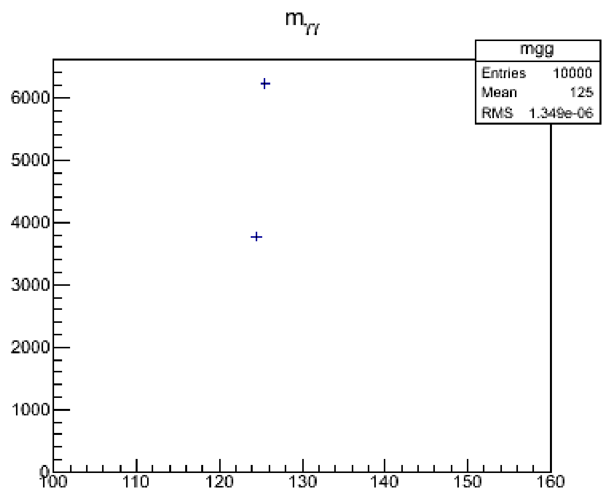
```
{Double_t mH = 125.;
TRandom3 gen(0);
TH1F* mgg = new TH1F("mgg" ,"m_{#gamma#gamma}”,60,100.,160.);
TH1F* ygg = new TH1F("ygg" ,"y_{#gamma#gamma}",100,-10.,10.);
for (Int_t i=0; i<events; i++) {
  Double_t phis = gen.Uniform(0.,TMath::TwoPi());
  Double_t coss = gen.Uniform(-1.,1.);
  Double_t sins = sqrt(1-coss*coss);
  TLorentzVector g1(cos(phis)*sins,sin(phis)*sins,coss,1.);
  g1*=(mH/2.);
  TLorentzVector g2(-cos(phis)*sins,-sin(phis)*sins,-coss,1.);
  g2*=(mH/2.);
  TLorentzVector H=g1+g2;
  mgg->Fill( H.M() );
  ygg->Fill( H.Rapidity() );
}
TCanvas myCanvas("myCanvas","Higgs properties",400,700);
myCanvas.Divide(1,2);
myCanvas.cd(1); mgg->Draw("e");
myCanvas.cd(2); ygg->Draw("e"); }
```

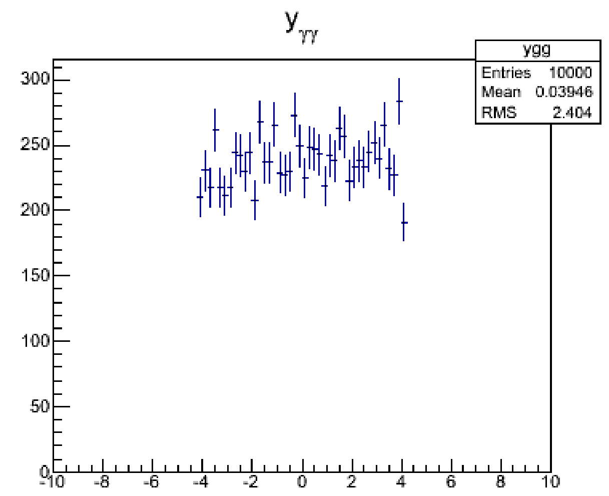




Higgs production

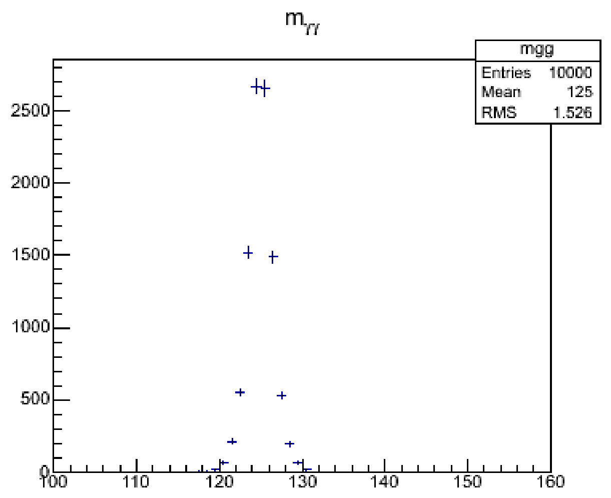
```
{
  ...
  Double_t sqrts = 8000.; // center of mass energy
  Double_t ymax = log     (sqrts/mH);
  for (Int_t i=0; i<events; i++) {
    ...
    Double_t y = gen.Uniform(-ymax,ymax);
    TLorentzVector beta(0.,0.,tanh(y),1.);
    g1.Boost(beta.BoostVector());
    g2.Boost(beta.BoostVector());
    ...
  }
  ...
}
```

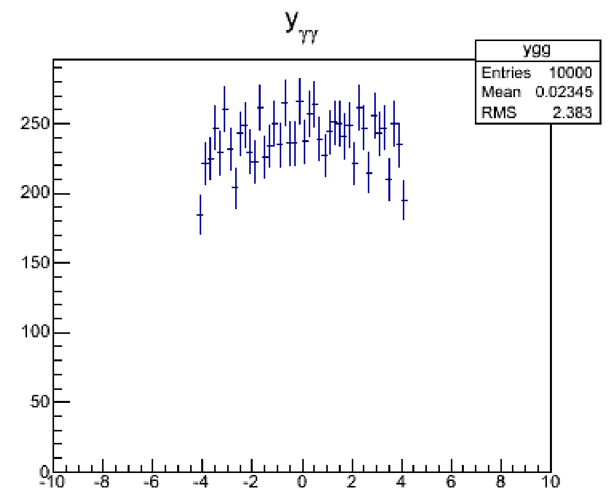




Energy resolution

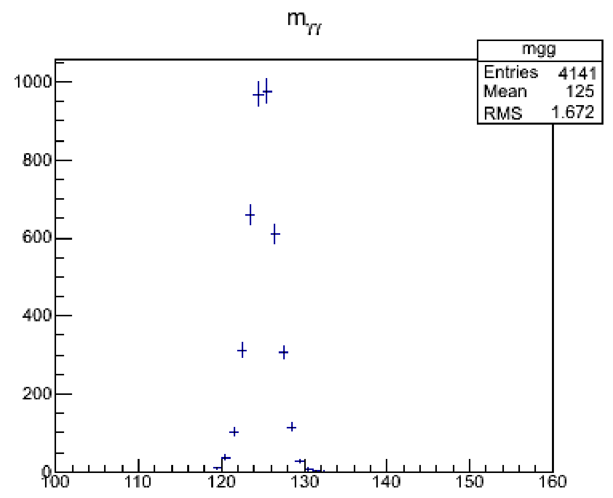
```
{
  ...
  Double_t A = 0.01;
  Double_t B = 0.15;
  for (Int_t i=0; i<events; i++) {
    ...
    g1*=(gen.Gaus(1.,sqrt(A*A+B*B/g1.E())));
    g2*=(gen.Gaus(1.,sqrt(A*A+B*B/g2.E())));
    ...
  }
  ...
}
```

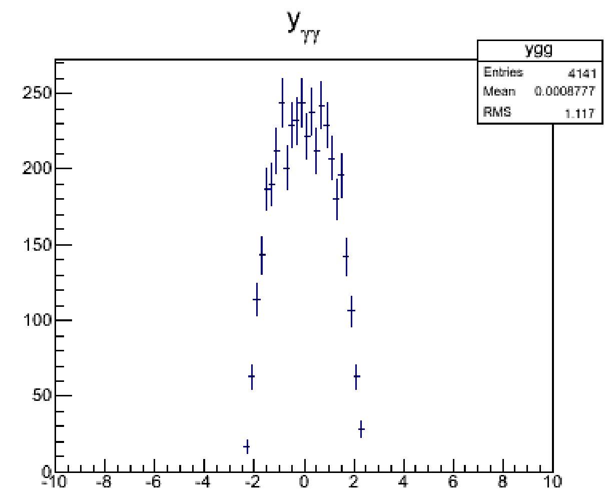




Detector and selection acceptance

```
{
  ...
  for (Int_t i=0; i<events; i++) {
    ...
    if ( fabs(g1.Eta())>2.4 || fabs(g2.Eta())>2.4 ) continue;
    if ( g1.ET()<40. && g2.ET()<40. ) continue;
    if ( g1.ET()<30. || g2.ET()<30. ) continue;
    ...
  }
  ...
}
```





Putting everything into a TTree

```
{
  ...
  TFile* myFile = TFile::Open(“myHiggs.root","RECREATE");
  TTree *tree = new TTree("tree","Di-photons");
  Double_t Et1, eta1, phi1, Et2, eta2, phi2;
  tree->Branch("Et1" ,&Et1 ,"Et1/D" );
  tree->Branch("eta1",&eta1,"eta1/D");
  tree->Branch("phi1",&phi1,"phi1/D");
  tree->Branch("Et2" ,&Et2 ,"Et1/D" );
  tree->Branch("eta2",&eta2,"eta2/D");
  tree->Branch("phi2",&phi2,"phi2/D");
  for (...) {
    ...
  }
  ...
  tree->Write();
  mgg->Write();
  ygg->Write();
  myFile->Close()
}
```

```
for (...) {
  ...
  Et1 =g1.Et();
  eta1=g1.Eta();
  phi1=g1.Phi();
  Et2 =g2.Et();
  eta2=g2.Eta();
  phi2=g2.Phi();
  tree->Fill();
  ...
}
```

Move to a compiled macro

```
#include "TRandom3.h"
#include "TH1F.h"
#include "TMath.h"
#include "TTree.h"
#include "TFile.h"
#include "TLorentzVector.h"
#include "TCanvas.h"

#include <cmath>

void generator_simple(Int_t events, char* filename) {
  ...
}
```

All used classes need their corresponding *header* file

Define interface to interactively change number of events and output file

Interpreted macro:

```
root -l
.L generator_simple.C
generator_simple(100000,"file.root")
```

Compiled macro:

```
root -l
.L generator_simple.C+
generator_simple(100000,"file.root")
```

Check the execution time difference

For other distributions

- The **TF1** and the **TH*** (including multi-dimensional histograms) classes provide a **GetRandom** method to generate random numbers according to the given distributions.
- **Example:**
  use for the photons a forward-peaked distribution, instead of an uniform one:

$$dN \propto \frac{d\cos\theta^*}{\sin\theta^*}$$

```
TF1* myPDF = new TF1("myPDF","1./sqrt(1-x*x)",-0.99,0.99);
for (Int_t i=0; i<events; i++) {
  ...
  Double_t coss = myPDF->GetRandom();
  ...
```

Fitting

- Determine a set of parameters of a model that better describe the measurements.
- But also:
  - Does the model described the data well enough?
  - Which are the uncertainties on the best set of parameters?
  - Correlations among the parameters
- Different techniques and methods:
  - Binned vs. unbinned data
  - □ χ^2 vs. log-Likelihood

Example: χ^2 fit

- ***Minimize deviations normalized by their uncertainties.***
- **Binned data**
  - Compare bin content of an histogram with expectation from model.
  - **Example:** data normally distributed $N_i^{\mathrm{exp}} = Ae^{-\frac{1}{2}\frac{(x_i-\bar{x})^2}{\sigma^2}}$

$$\chi^2 = \sum_{i=\mathrm{bins}} \frac{\left(N_i^{\mathrm{obs}} - N_i^{\mathrm{exp}}\right)^2}{N_i^{\mathrm{obs}}} = \sum_{i=\mathrm{bins}} \frac{\left(N_i^{\mathrm{obs}} - A\exp\left(-\frac{1}{2}\frac{(x_i-\bar{x})^2}{\sigma^2}\right)\right)^2}{N_i^{\mathrm{obs}}}$$

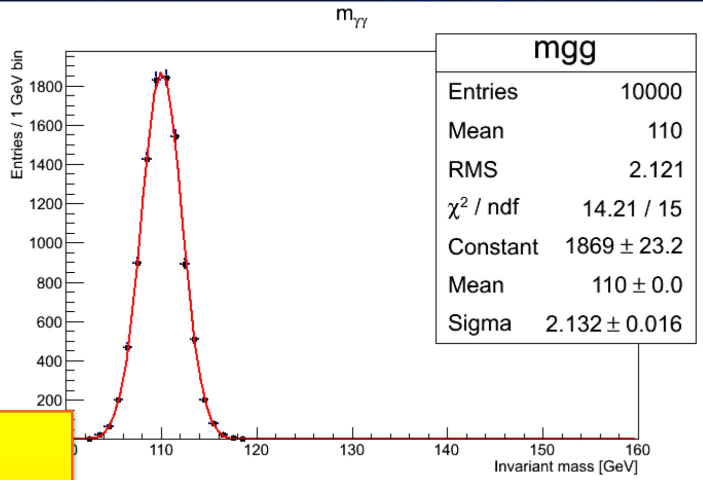
Poisson statistics

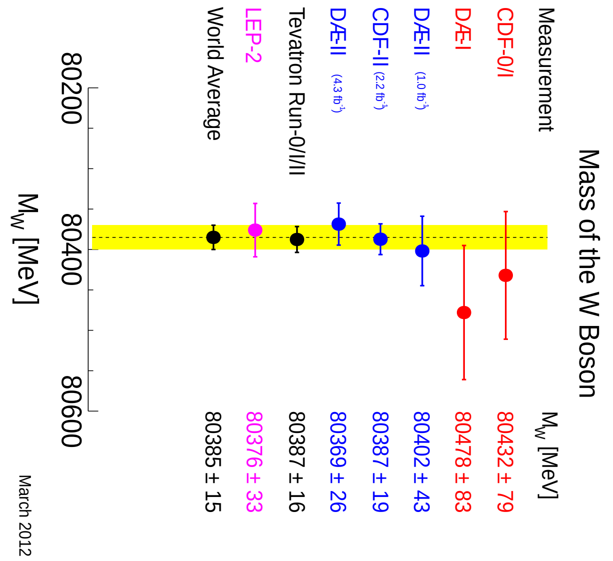
Exercise: figure out the connection between A and number of events in Gaussian

- **Unbinned data**
  - Compare each individual measurement with model
  - **Example:** different measuremens of the same physics quantity α

$$\chi^2 = \sum_{i=\mathrm{measurements}} \frac{(x_i - \alpha)^2}{\sigma_i^2}$$

  - Gets back the well known weighted mean: $\alpha = \dfrac{\sum_i x_i/\sigma_i^2}{\sum_i 1/\sigma_i^2}$
- **1σ uncertainty**: increase of χ^2 by 1





Example: Likelihood fit

- ***Use the pdf of observables and maximize the product of likelihoods.***
  - In practice: minimize the -2*logarithm of the likelihoods
- **Binned data**
  - Use Poisson pdf for bin content
  - **Example:** data normally distributed

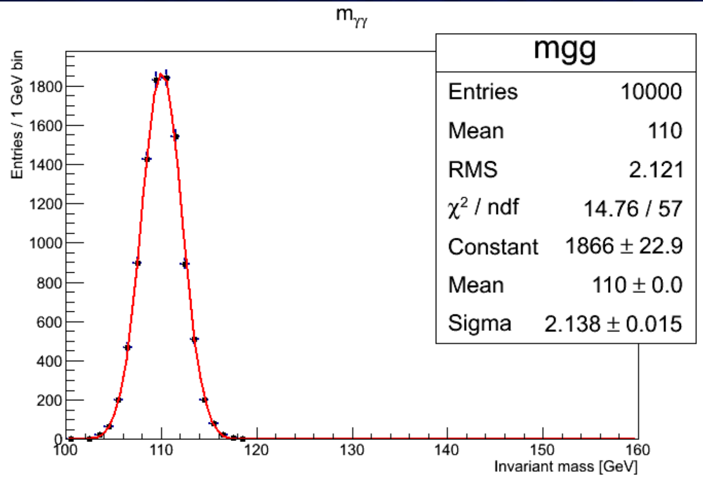
$$\mathcal{L} = \prod_{i=\text{bins}} \text{Poisson}\left[N_i^{\text{obs}}, A\exp\left(-\frac{1}{2}\frac{(x_i-\bar{x})^2}{\sigma^2}\right)\right]$$

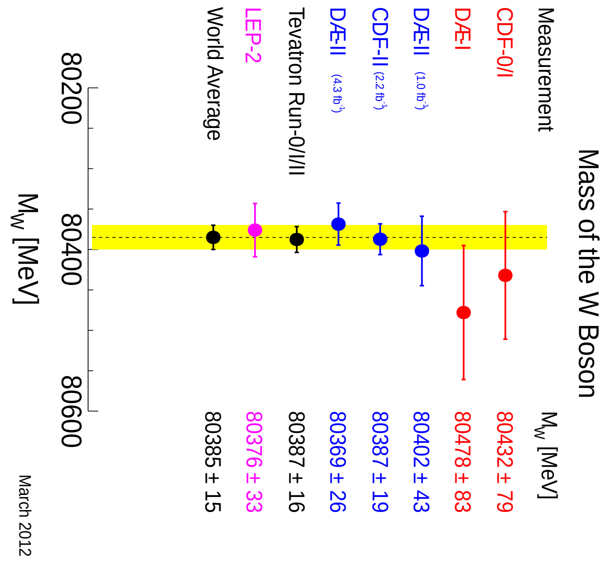
- **Unbinned data**
  - **Example:** normally distributed points about α

$$\mathcal{L} = \prod_{i=\text{measurements}} \frac{1}{\sqrt{2\pi}\sigma_i}\exp\left(-\frac{1}{2}\frac{(x_i-\alpha)^2}{\sigma_i^2}\right)$$

$$-2\ln\mathcal{L} = \sum_{i=\text{measurements}} \left[\frac{(x_i-\alpha)^2}{\sigma_i^2} + 2\ln\left(\sqrt{2\pi}\sigma_i\right)\right]$$

  - It looks familiar, doesn't it?
- **1σ uncertainty**: increase of $-2\ln\mathcal{L}$ by 1





Fitting in the GUI

Sample histrogram in **myHisto.root:**

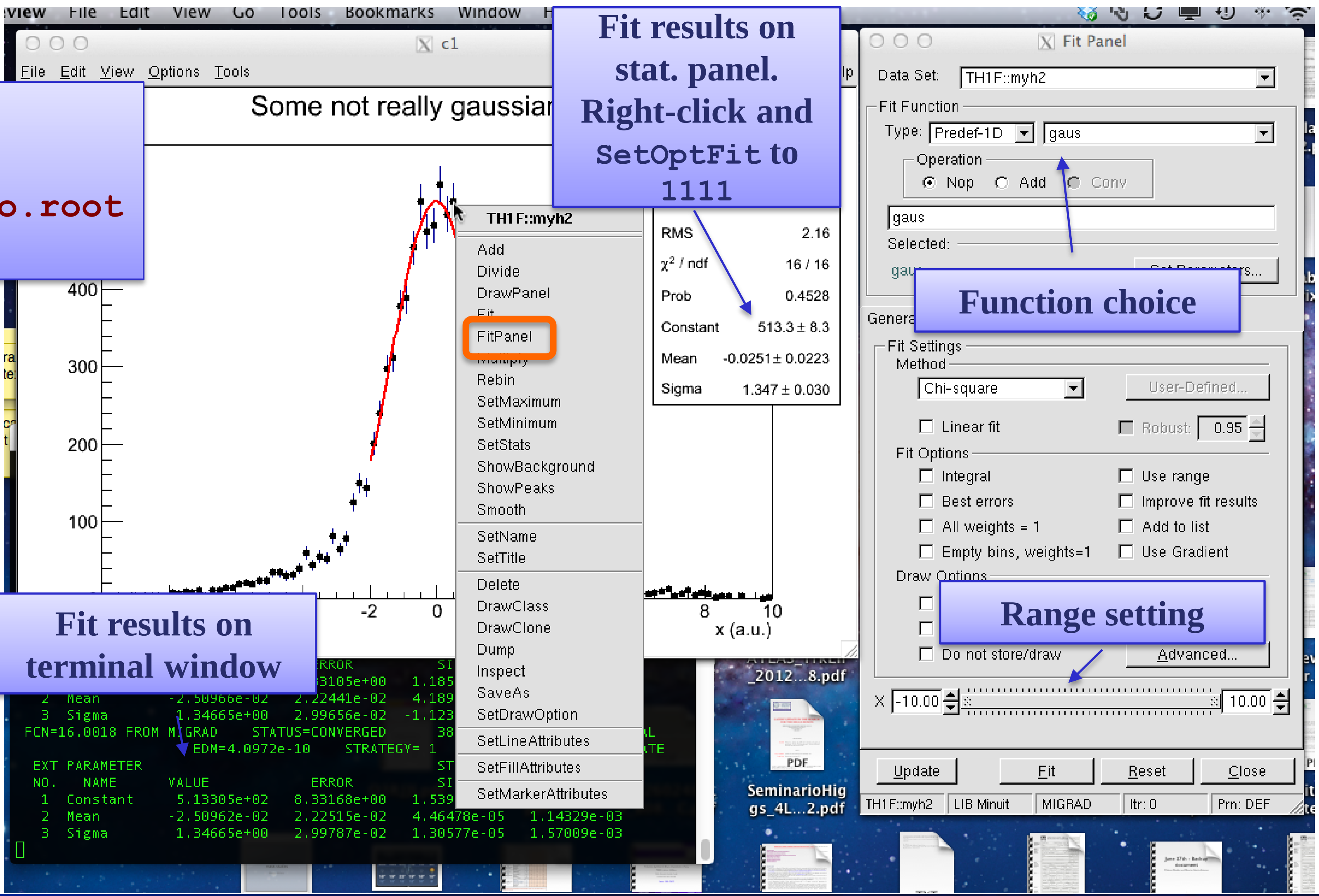
```
> root -l myHisto.root
> myh2->Draw();
```

Fit results on stat. panel. Right-click and `SetOptFit` to `1111`

Function choice

Range setting

Fit results on terminal window



RMS	2.16
χ^2 / ndf	16 / 16
Prob	0.4528
Constant	513.3 ± 8.3
Mean	-0.0251± 0.0223
Sigma	1.347 ± 0.030

Data Set: TH1F::myh2
Fit Function — Type: Predef-1D, gaus
Operation: [✓ Nop] Add Conv
gaus
Selected: gaus
Fit Settings — Method: Chi-square
Linear fit; Robust: 0.95
Fit Options: Integral, Use range, Best errors, Improve fit results, All weights = 1, Add to list, Empty bins, weights=1, Use Gradient
Draw Options: Do not store/draw; Advanced...
X -10.00 — 10.00
TH1F::myh2 | LIB Minuit | MIGRAD | Itr: 0 | Prn: DEF

```
                                   ERROR          SI
                                 3105e+00    1.185
  2  Mean        -2.50966e-02   2.22441e-02   4.189
  3  Sigma        1.34665e+00   2.99656e-02  -1.123
 FCN=16.0018 FROM MIGRAD    STATUS=CONVERGED      38
                     EDM=4.0972e-10    STRATEGY= 1
  EXT PARAMETER                                   ST
  NO.   NAME      VALUE            ERROR          SI
   1  Constant     5.13305e+02   8.33168e+00   1.539
   2  Mean        -2.50962e-02   2.22515e-02   4.46478e-05   1.14329e-03
   3  Sigma        1.34665e+00   2.99787e-02   1.30577e-05   1.57009e-03
```

Simple fits

```
myh2->Fit("gaus","","",-2.,2.);
```

Function name

Fit options

Plotting options

Fit range

```
myh2->GetFunction(”gaus”)->Draw();
```

Retrieve a pointer to the fitted TF1

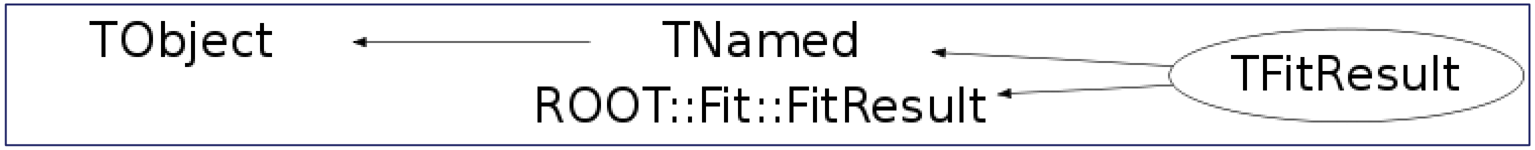
The ponter can be used to manipulate the function: plotting, changing style, inspecting parameters...

```
TFitResultPtr fitres = myh2->Fit(”gaus”,”S”);

fitres->Parameter(2);
fitres->ParError(2);
fitres->Chi2();
fitres->Ndf();
```

- Some predefined fit functions available:
  - Exponential
  - Gaussian
  - Landau
  - Polynomial (up to 9th degree)
- Most common options:
  - `“L”` likelihood fit
  - `“WL”` likelihood fit with weighed points
  - `“Q”` quiet mode: do not print info on screen
  - `“S”` save the results in a TFitResultPtr opbect.

TFitResult

- The **`TFitResultPtr`** is a class that behaves like a pointer to a **`TFitResult`** object.
  - The latter inherits from a **`ROOT::Fit::FitResult`**

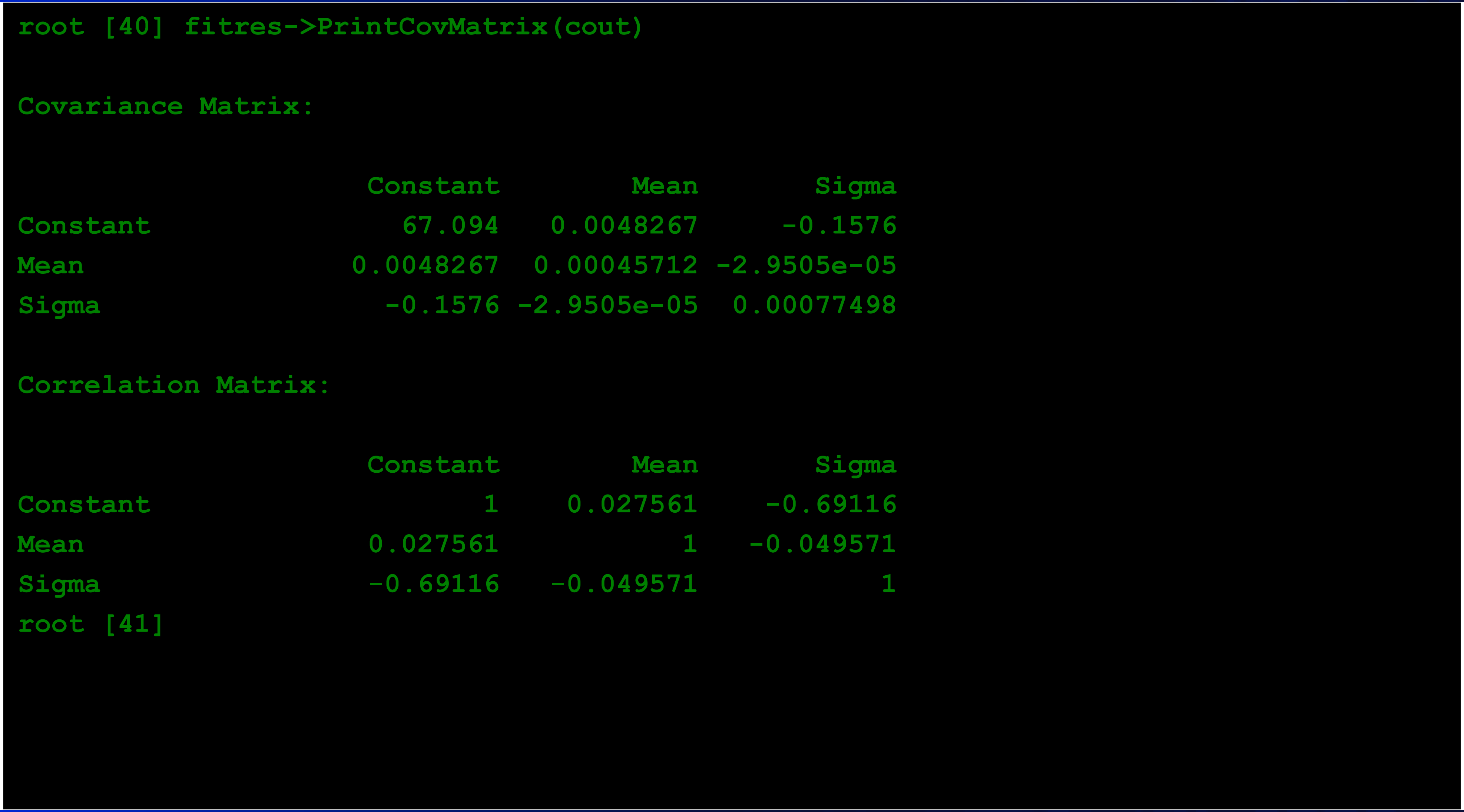


- Many accessor methods, full documentation at `http://root.cern.ch/root/html/ROOT__Fit__FitResult.html`

TFitResult

```
root [38] TFitResultPtr fitres = myh2->Fit("gaus","S","",-2.,2.)
 FCN=16.5644 FROM MIGRAD    STATUS=CONVERGED      71 CALLS          72 TOTAL
                     EDM=6.18403e-09    STRATEGY= 1       ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     5.12179e+02   8.19111e+00   1.21137e-02   2.10983e-06
   2  Mean        -1.99953e-02   2.13804e-02   4.36950e-05  -5.06752e-03
   3  Sigma        1.35596e+00   2.78382e-02   1.20495e-05  -1.87226e-03
root [39] fitres->Print(cout)

****************************************
Minimizer is Minuit / Migrad
Chi2                      =      16.5644
NDf                       =           17
Edm                       =  6.18403e-09
NCalls                    =           72
Constant                  =      512.179   +/-   8.19111
Mean                      =   -0.0199953   +/-   0.0213804
Sigma                     =      1.35596   +/-   0.0278382            (limited)
```

TFitResult



```
root [40] fitres->PrintCovMatrix(cout)

Covariance Matrix:

                      Constant        Mean       Sigma
Constant                67.094   0.0048267     -0.1576
Mean                 0.0048267  0.00045712 -2.9505e-05
Sigma                  -0.1576 -2.9505e-05  0.00077498

Correlation Matrix:

                      Constant        Mean       Sigma
Constant                     1    0.027561    -0.69116
Mean                  0.027561           1   -0.049571
Sigma                 -0.69116   -0.049571           1
root [41]
```

Inlined functions

- **Not always you can find the function**
  ...or the parameterization that you want to use.
- In this case we want to try to fit our distribution as the *sum of two Gaussians*.
- So you can define it yourself providing its formula:

```
TF1* gauss2 = new TF1("gauss2",
  "([0]/[2])*exp(-0.5*(x-[1])**2/[2]**2)+([3]/[4])*exp(-0.5*(x-[1])**2/[4]**2)",
  -10.,10.);
```

- And use in fitting:

```
TFitResultPtr fitres = histo->Fit("gauss2","S");
```

- Just remember in between:

```
gauss2->SetParameter(0,100.);
gauss2->SetParameter(1,0.);
gauss2->SetParameter(2,1.);
gauss2->SetParameter(3,100.);
gauss2->SetParameter(4,2.);
```

C-functions

- **Sometimes you cannot put everything on a line...**
- ...use a normal C-function:

```
Double_t myTwoGauss(Double_t *x, Double_t *p) {
  Double_t norm = 1./sqrt(TMath::TwoPi());
  Double_t g1 = exp(-0.5*pow((x[0]-p[1])/p[2],2));
  g1*=(norm*p[0]/p[2]);
  Double_t g2 = exp(-0.5*pow((x[0]-p[1])/p[4],2));
  g2*=(norm*p[3]/p[4]);
  return g1+g2;
}
```

- So you can define it yourself providing its formula:

```
TF1* gauss3 = new TF1("gauss3", myTwoGauss, -10., 10., 5);
```

- And use in fitting:

```
TFitResultPtr fitres2 = histo->Fit("gauss3","S");
```

- Just remember to set the parameters!

Let's run the `FitTwoGaussian.C` macro

...and when things starts to become really complicated

ROOFIT

Introduction to RooFit

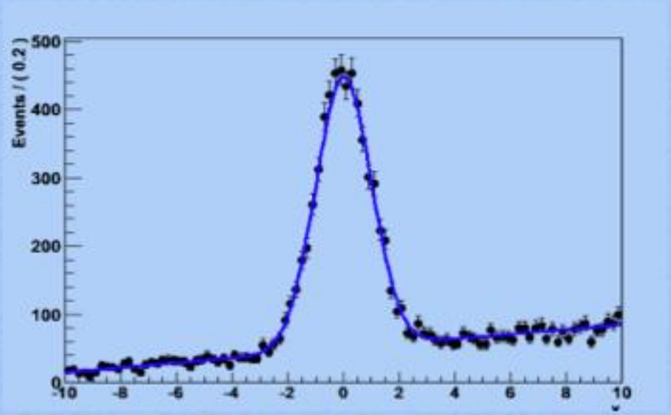
- **Disclaimer:**
  I am not a RooFit expert: I learnt it last week because of **you**!
  Only few of the features will be presented here!

- So you can get much better material from the official sources:
  - RooFit web page: **`http://roofit.sourceforge.net/`**
  - RooFit tutorials:
    `http://root.cern.ch/root/html/tutorials/roofit/index.html`
  - RooFit entry in the ROOT reference guide:
    `http://root.cern.ch/root/html/ROOFIT_Index.html`

- In particular I appreciated a lot the tutorial at the *Desy 2012 School of Statistics* by L. Moneta and S. Kreiss:
  `https://twiki.cern.ch/twiki/bin/view/RooStats/WebHome#Resources`
  many slides here by them and W. Werverke

RooFit

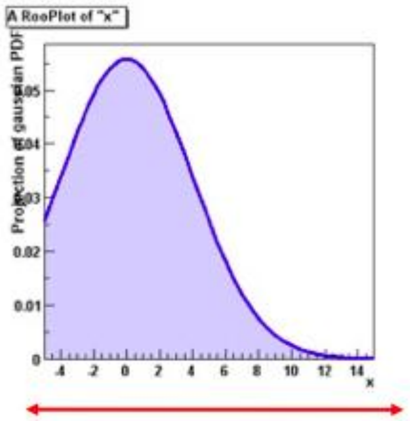
- Toolkit for data modeling
  - developed by *W. Verkerke and D. Kirkby*
- model distribution of observable ***x*** in terms of parameters ***p***
  - probability density function (pdf): $\mathcal{P}$`(x;p)`
- pdf are normalized over allowed range of observables ***x*** with respect to the parameters ***p***



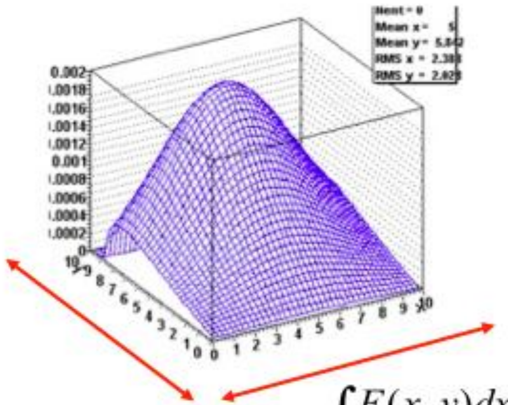
Mathematic – Probability density functions

- Probability Density Functions describe probabilities, thus
  - All values most be >0
  - The total probability must be 1 *for each* *p*, i.e.
  - Can have any number of dimensions

$$\int_{\vec{x}_{\min}}^{\vec{x}_{\max}} g(\vec{x},\vec{p})\,d\vec{x} \equiv 1$$

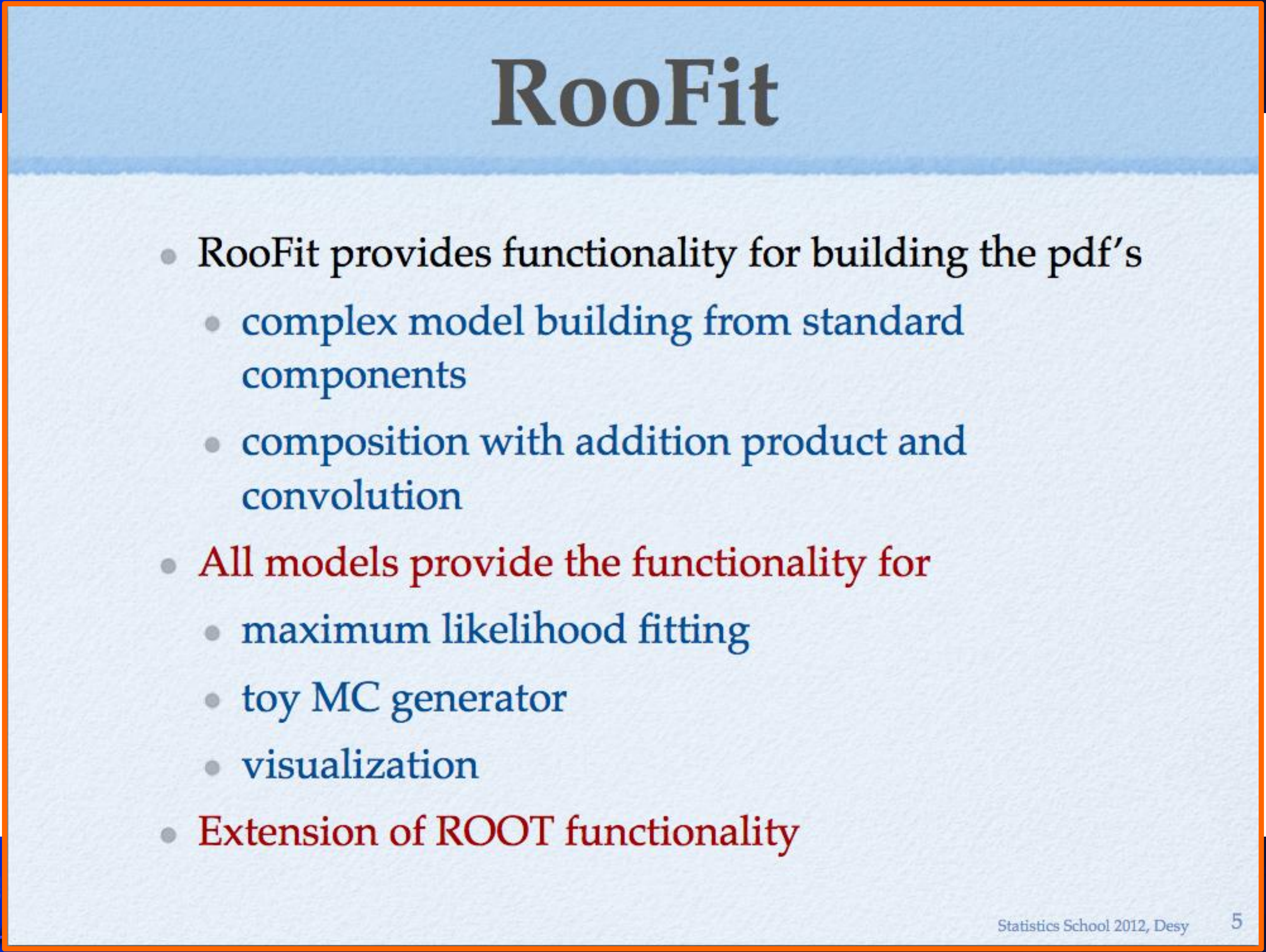


$$\int F(x)dx \equiv 1$$



$$\int F(x,y)dxdy \equiv 1$$

- Note distinction in role between *parameters* (p) and *observables* (x)
  - Observables are measured quantities
  - Parameters are degrees of freedom in your model

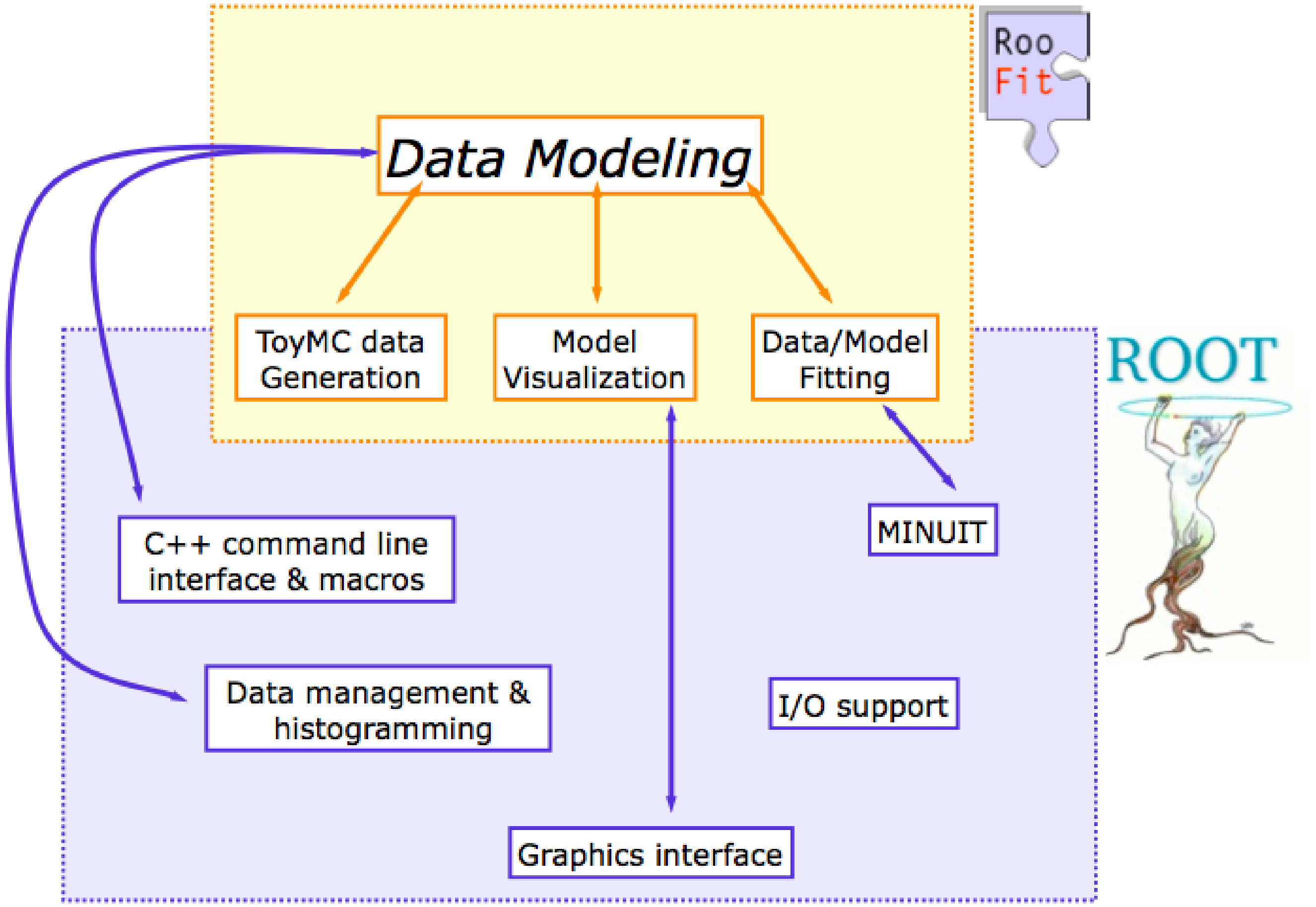


RooFit

- RooFit provides functionality for building the pdf's
  - complex model building from standard components
  - composition with addition product and convolution
- All models provide the functionality for
  - maximum likelihood fitting
  - toy MC generator
  - visualization
- Extension of ROOT functionality

Introduction – Relation to ROOT

Extension to ROOT – (Almost) no overlap with existing functionality

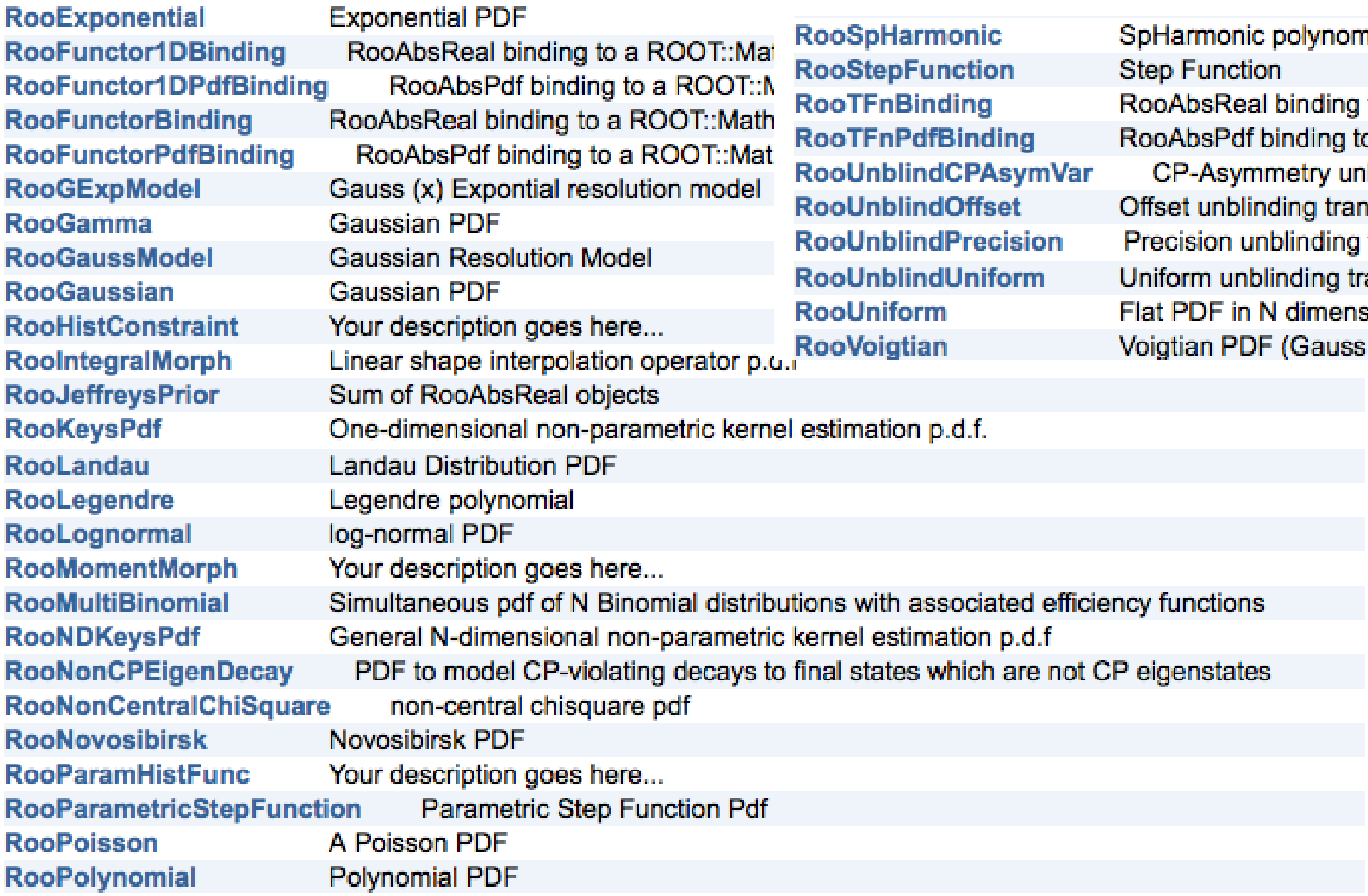


RooFit core design philosophy

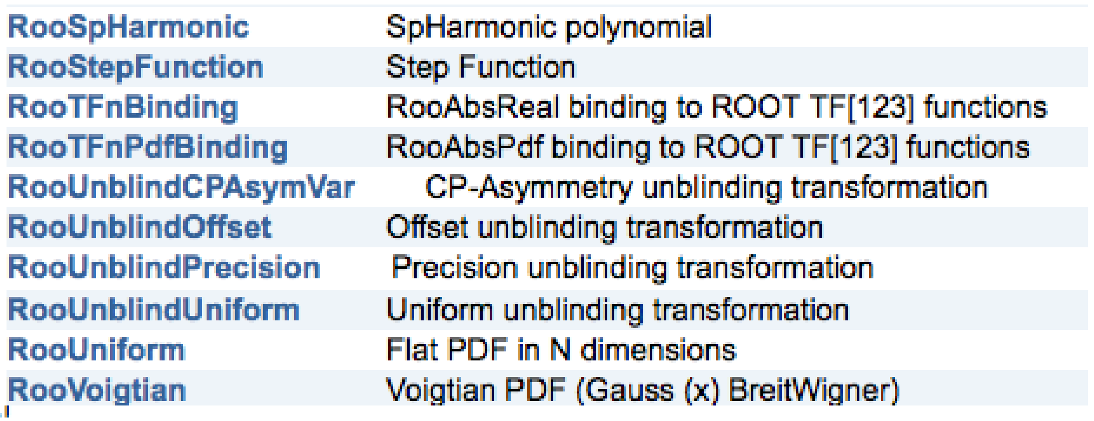
- Mathematical objects are represented as C++ objects

Mathematical concept		RooFit class
variable	x	`RooRealVar`
function	$f(x)$	`RooAbsReal`
PDF	$f(x)$	`RooAbsPdf`
space point	$\vec{x}$	`RooArgSet`
integral	$\int_{x_{\min}}^{x_{\max}} f(x)dx$	`RooRealIntegral`
list of space points		`RooAbsData`

Some of the available pdf



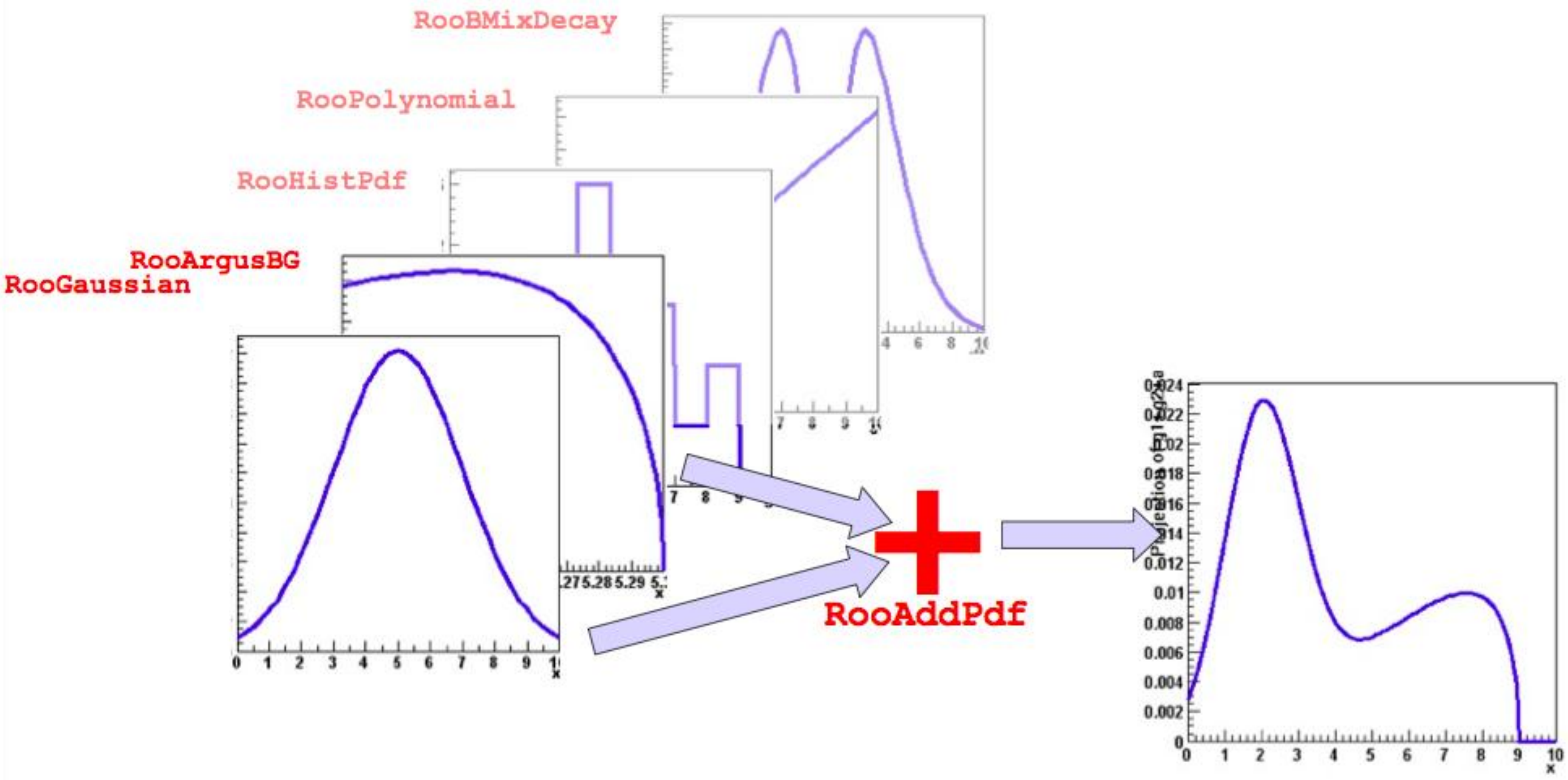
RooExponential	Exponential PDF
RooFunctor1DBinding	RooAbsReal binding to a ROOT::Mat
RooFunctor1DPdfBinding	RooAbsPdf binding to a ROOT::M
RooFunctorBinding	RooAbsReal binding to a ROOT::Math
RooFunctorPdfBinding	RooAbsPdf binding to a ROOT::Mat
RooGExpModel	Gauss (x) Expontial resolution model
RooGamma	Gaussian PDF
RooGaussModel	Gaussian Resolution Model
RooGaussian	Gaussian PDF
RooHistConstraint	Your description goes here...
RooIntegralMorph	Linear shape interpolation operator p.d.
RooJeffreysPrior	Sum of RooAbsReal objects
RooKeysPdf	One-dimensional non-parametric kernel estimation p.d.f.
RooLandau	Landau Distribution PDF
RooLegendre	Legendre polynomial
RooLognormal	log-normal PDF
RooMomentMorph	Your description goes here...
RooMultiBinomial	Simultaneous pdf of N Binomial distributions with associated efficiency functions
RooNDKeysPdf	General N-dimensional non-parametric kernel estimation p.d.f
RooNonCPEigenDecay	PDF to model CP-violating decays to final states which are not CP eigenstates
RooNonCentralChiSquare	non-central chisquare pdf
RooNovosibirsk	Novosibirsk PDF
RooParamHistFunc	Your description goes here...
RooParametricStepFunction	Parametric Step Function Pdf
RooPoisson	A Poisson PDF
RooPolynomial	Polynomial PDF



RooSpHarmonic	SpHarmonic polynomial
RooStepFunction	Step Function
RooTFnBinding	RooAbsReal binding to ROOT TF[123] functions
RooTFnPdfBinding	RooAbsPdf binding to ROOT TF[123] functions
RooUnblindCPAsymVar	CP-Asymmetry unblinding transformation
RooUnblindOffset	Offset unblinding transformation
RooUnblindPrecision	Precision unblinding transformation
RooUnblindUniform	Uniform unblinding transformation
RooUniform	Flat PDF in N dimensions
RooVoigtian	Voigtian PDF (Gauss (x) BreitWigner)

Model building – (Re)using standard components

- Most realistic models are constructed as the sum of one or more p.d.f.s (e.g. signal and background)
- Facilitated through operator p.d.f **`RooAddPdf`**



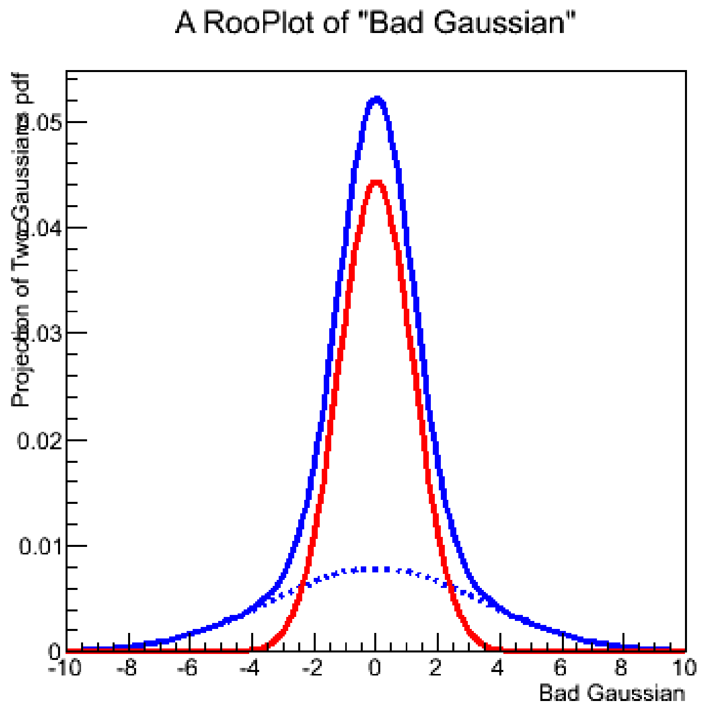
Two Gaussians in RooFit

- This is the story behind the histogram used for the fit study:

```
RooRealVar x("x","Bad Gaussian",-10.,10.);
RooRealVar mean("mean","mean of Gaussian",0.,-10.,10.) ;
RooRealVar sigma1("sigma1","width of narrow Gaussian",1.2,0.1,10.) ;
RooRealVar sigma2("sigma2","width of wide Gaussian",3.4,2.,10.) ;
RooRealVar fraction("fraction","fraction of narrow Gaussian",2./3.,0.,1.);

RooGaussian gauss1("gauss1","Narrow Gaussian",x,mean,sigma1);
RooGaussian gauss2("gauss2","Wide Gaussian",x,mean,sigma2);

RooAddPdf twogauss("twogauss","Two Gaussians pdf",
                    RooArgList(gauss1,gauss2),fraction);
```

Now `twogaussian` is pdf: what can we do with it?

Drawing a pdf



- Simply plotting a pdf is a bit more elaborate:
  - Create a frame to show a certain variable

```
RooPlot* xframe = x.frame();
```

  - Plot the pdf in that frame

```
twogauss.plotOn(xframe)
```

  - But also can plot individual components, setting the style on the fly:

```
twogauss.plotOn(xframe,Components("gauss2"),LineStyle(kDashed));
twogauss.plotOn(xframe,Components("gauss1"),LineColor(kRed));
```

  - Finally draw the frame

```
xframe.Draw();
```

Generate events according to a pdf

- From a pdf it is possible to create datasets:
  - Create a dataset of 10000 sampling of variable *x*

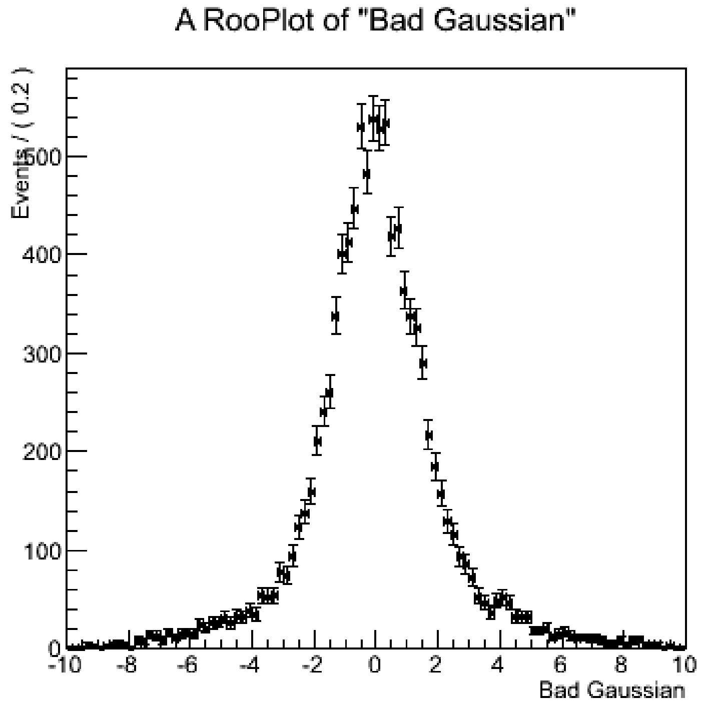
```
RooDataSet* mydata = twogauss.generate(x,10000);
```

  - Plot the dataset in a frame

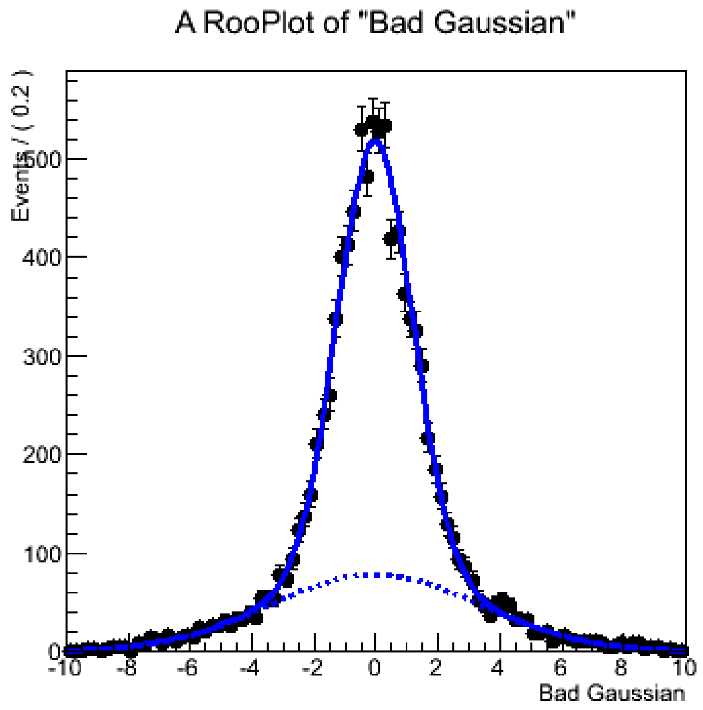
```
RooPlot* xframe = x.frame();
mydata->plotOn(xframe)
```

  - Finally draw the frame

```
xframe.Draw();
```



Fitting a dataset



- A pdf can be fitted to a dataset:
  - This is very simple:

```
twogauss.fitTo(*mydata);
```

  - Plot the resulting dataset and pdf in at frame

```
mydata->plotOn(xframe);
twogauss.plotOn(xframe);
twogauss.plotOn(xframe,Components("gauss2"),LineStyle(kDashed));
```

  - Like for normal fits, it is possible to store results in a **`RooFitResult`**:

```
RooFitResult* fitres = twogauss.fitTo(*mydata,Save());
fitres->Print();
fitres->correlationMatrix()->Print();
```

To fix a parameter:

```
mean=0.;
mean.setConstant(true);
```

Datasets <-> TTree, TH1

- It is possible to convert a TTree or a TH1 into a RooDataSet for fitting

```
    RooDataSet mytreedata(“mytreedata”,”imported
data”,x,Import(*myTree));

    RooDataHist myhistdata(“myhistdata”,”imported
data”,x,Import(*myTH1));
```

- And also the other way around (this is how our test histogram was created)

```
    TH1F* myh = x.createHistogram("myh","Entries");
    TH1* myh2 = mydata->fillHistogram(myh,x);

    TTree* myTree =  mydata->tree();
```

Another way of summing

- In many application we are interested to know how many event are in the narrow Gaussian (**`N1`**) or in the wide one (**`N2`**):

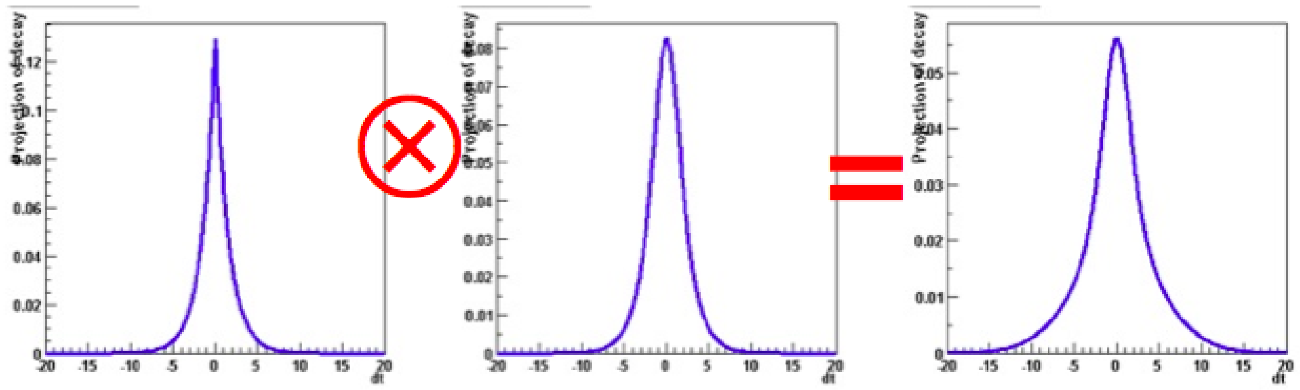
```
RooRealVar x("x","Bad Gaussian",-10.,10.);
RooRealVar mean("mean","mean of Gaussian",0.,-10.,10.) ;
RooRealVar sigma1("sigma1","width of narrow Gaussian",1.2,0.1,10.) ;
RooRealVar sigma2("sigma2","width of wide Gaussian",3.4,2.,10.) ;

RooGaussian gauss1("gauss1","Narrow Gaussian",x,mean,sigma1);
RooGaussian gauss2("gauss2","Wide Gaussian",x,mean,sigma2);

RooRealVar N1(”N1",”events in narrow Gaussian”,6000.);
RooRealVar N2(”N2",”events in wide Gaussian”,3000.);
RooAddPdf twogauss("twogauss","Two Gaussians pdf",
                    RooArgList(gauss1,gauss2),RooArgList(N1,N2));
```

It will be useful in tomorrow exercises.

Convolution

- Model representing a convolution of a theory model and a resolution model often useful

$$f(x)\otimes g(x)=\int_{-\infty}^{+\infty} f(x)g(x-x')dx'$$



- But numeric calculation of convolution integral can be challenging. No one-size-fits-all solution, but 3 options available
  - Analytical convolution (BW⊗Gauss, various B physics decays)
  - Brute-force numeric calculation (slow)
  - FFT numeric convolution (fast, but some side effects)

Example of convolution: BW+Gauss

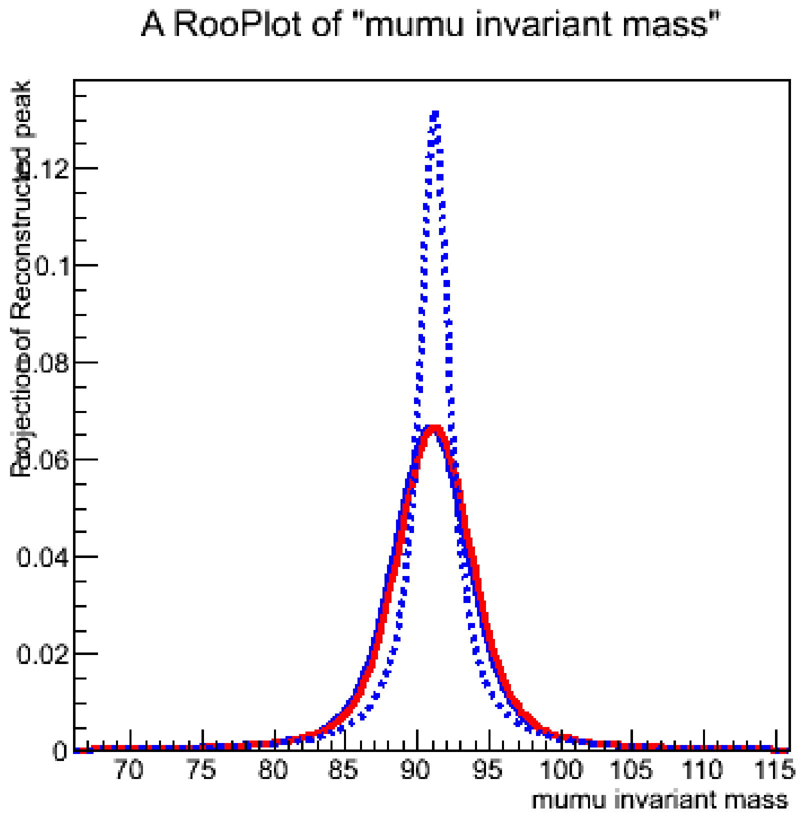
- Let’s assume we want to describe the widening of the Z resonance peak (Breit-Wigner shape) due to detector resolutio (Gaussian).

```
RooRealVar m("m","mumu invariant mass",91.,66.,116.);

RooRealVar mZ("mZ","Z mass",91.1876);
RooRealVar GZ("GZ","Z width",2.4952);
RooBreitWigner peak("peak","Z peak",m,mZ,GZ);

RooRealVar mres("mres","auxiliary variable",0.,-20.,20.);
RooRealVar sigma("sigma","mass resolution",2.,0.,10.);
RooGaussian gauss("gauss","Resolution function",m,RooConst(0.),sigma);

RooFFTConvPdf conv("conv","Reconstructed peak",m,peak,gauss);
RooVoigtian voigt("voigt","Reconstructed peak",m,mZ,GZ,sigma);
```

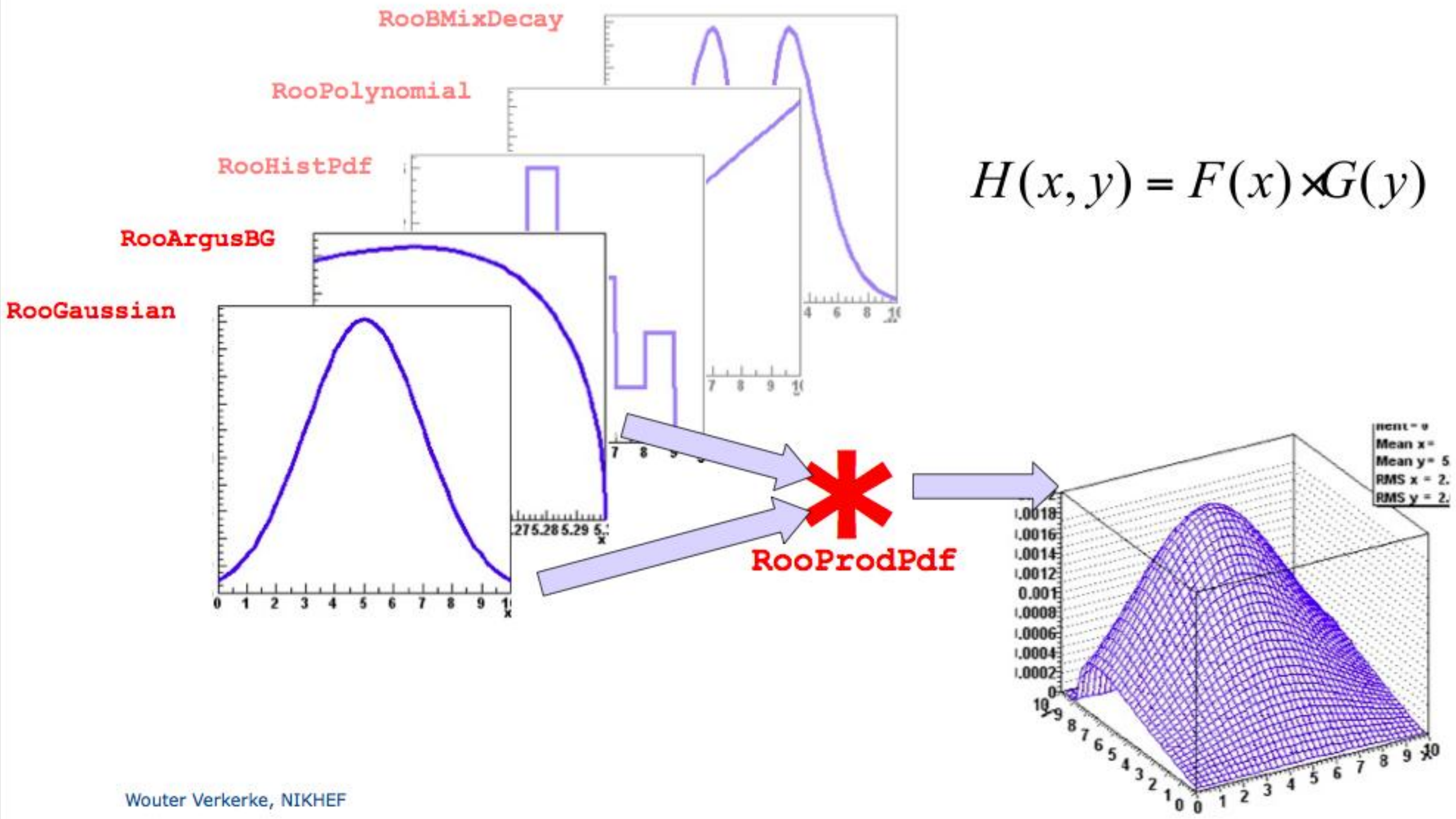
Let’s run the `Convolution.C` macro.

Comparing FFT vs Analytical

```
TCanvas cfun("cfun","Plot of a function",400,400);

RooPlot* xframe = m.frame(Range(66.,116.));

conv.plotOn(xframe);
voigt.plotOn(xframe,LineColor(kRed));
peak.plotOn(xframe,LineStyle(kDashed));

xframe.Draw();
```



Model building – Products of ~~un~~correlated p.d.f.s



$$H(x,y) = F(x) \times G(y)$$

Product of PDF: Poisson with uncertainty on μ

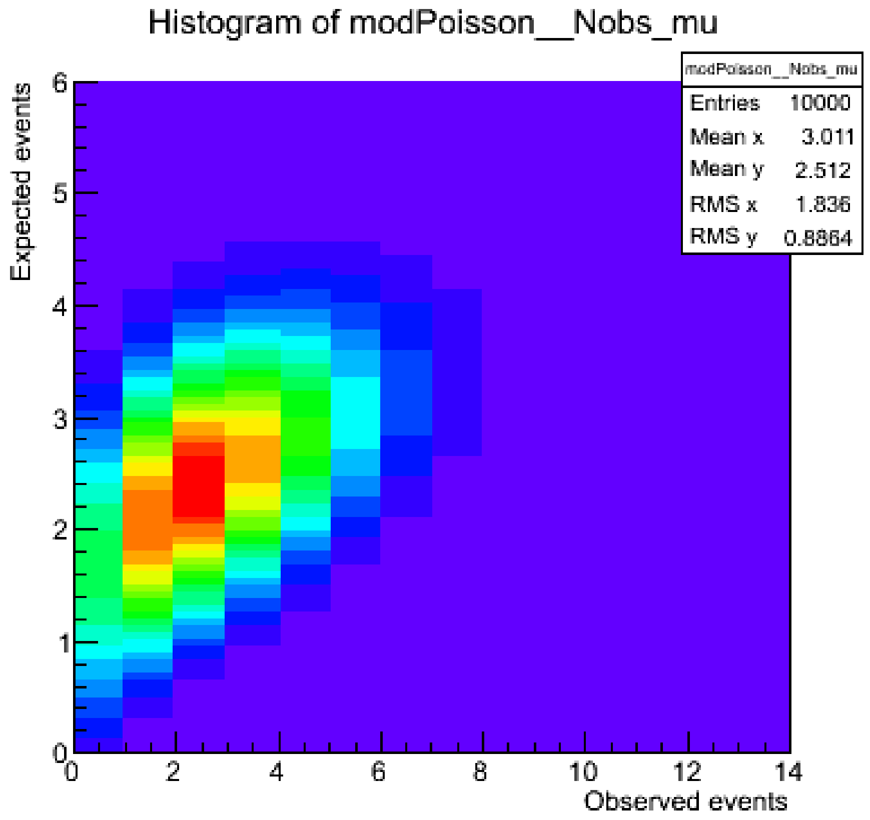
- We expect data to be distributed according to a Poisson distribution
  ...but we have some uncertainty on value of μ.
- We can describe the problem in RooFit as product of two probability distribution, one *conditional* on the other.

```
RooRealVar Nobs("Nobs","Observed events",1,0.,14.);
RooRealVar mu("mu","Expected events",0.,6.);
RooPoisson observed("observed","Observed events",Nobs,mu);

RooGaussian mupdf("mupdf","Mu value",mu,RooConst(2.5),RooConst(0.9));

RooProdPdf modPoisson("modPoisson","Poisson with uncertainty on mu",
                 mupdf,Conditional(observed,Nobs));
```

Let's run the `PoissonComposition.C` macro.

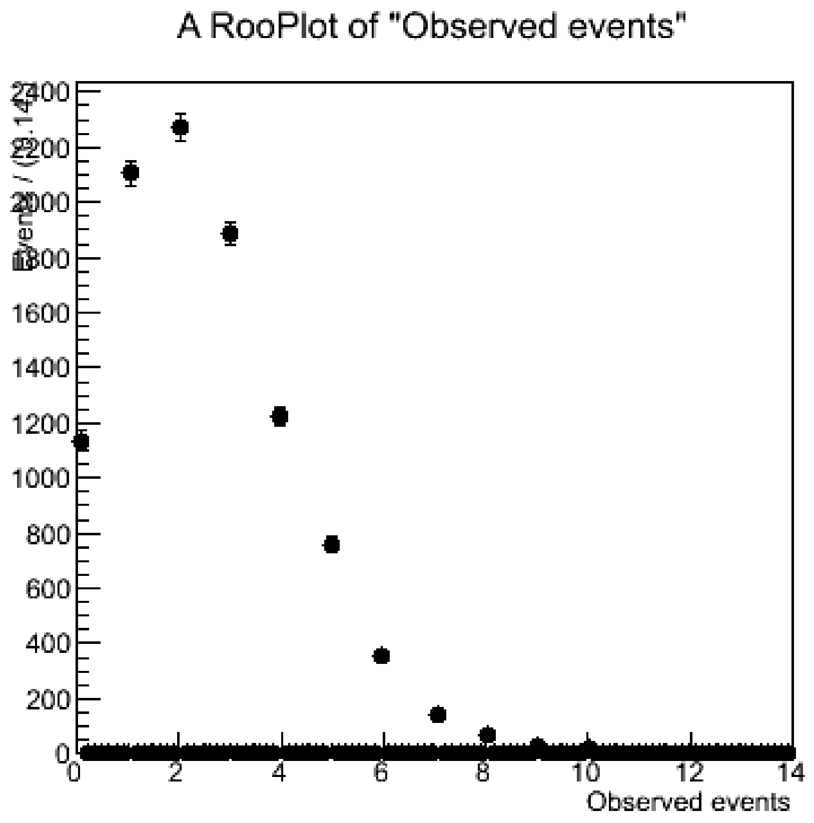
Plotting the 2D pdf

```
RooDataSet* mydata = modPoisson.generate(RooArgSet(Nobs,mu),10000);

TCanvas c2D("cfun","2D distribution",400,400);
TH2* hh = modPoisson.createHistogram("Nobs,mu");
hh->Draw("COL");

TCanvas cmu("cmu","mu distribution",400,400);
RooPlot* frame1 = mu.frame();
mydata->plotOn(frame1);
frame1.Draw();

TCanvas cN("cN","Nobs distribution",400,400);
RooPlot* frame2 = Nobs.frame();
mydata->plotOn(frame2);
frame2.Draw();
```



Plotting the N_{obs} pdf

```
RooDataSet* mydata = modPoisson.generate(RooArgSet(Nobs,mu),10000);

TCanvas c2D("cfun","2D distribution",400,400);
TH2* hh = modPoisson.createHistogram("Nobs,mu");
hh->Draw("COL");

TCanvas cmu("cmu","mu distribution",400,400);
RooPlot* frame1 = mu.frame();
mydata->plotOn(frame1);
frame1.Draw();

TCanvas cN("cN","Nobs distribution",400,400);
RooPlot* frame2 = Nobs.frame();
mydata->plotOn(frame2);
frame2.Draw();
```


A RooPlot of "Observed events"
Events / (0.14)
Observed events