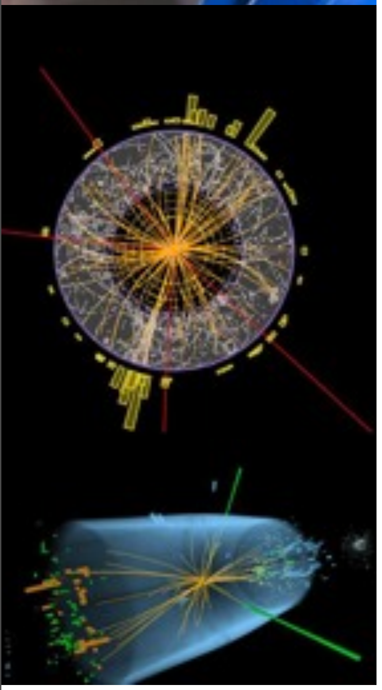
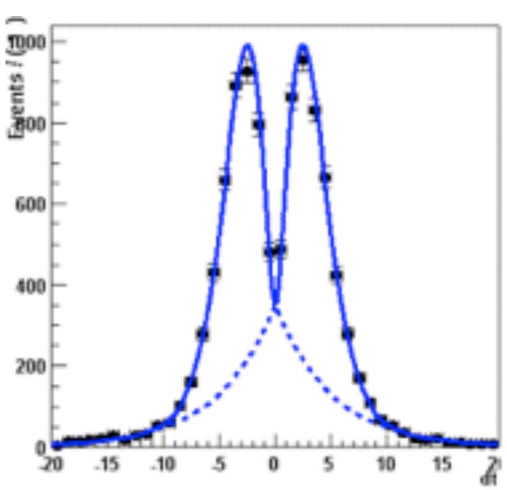
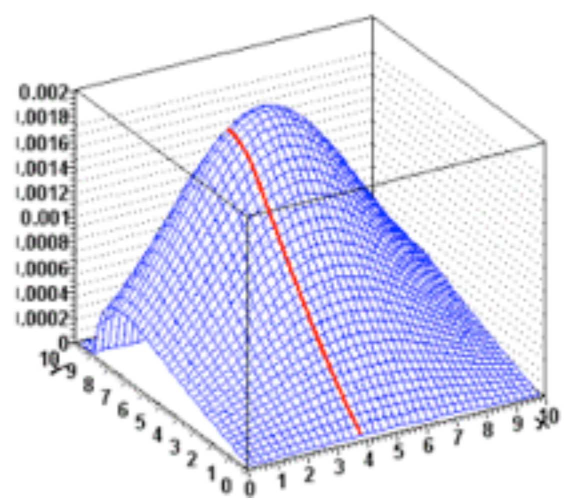


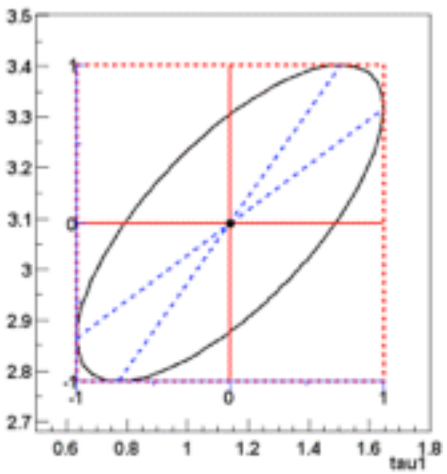
Introduction to RooFit

ROOT Training at IRMM

1st March 2013







Dr Lorenzo Moneta
CERN PH-SFT
CH-1211 Geneva 23
sftweb.cern.ch
root.cern.ch

Outline

- RooFit
  - Introduction and overview of basic functionality
  - Creation and basic use of models
  - Using the RooFit workspace class (**`RooWorkspace`**)
  - Building composite models
- We will follow the lecture slides with some simple hands-on exercises
- Introduction to RooStats

Credit

- RooFit sides and example are extrated from material prepared by *W. Verkerke (NIKHEF)*, author of RooFit
  - more information and additional slides from W. Verkerke are available at
    - http://indico.in2p3.fr/getFile.py/access?contribId=15&resId=0&materialId=slides&confId=750

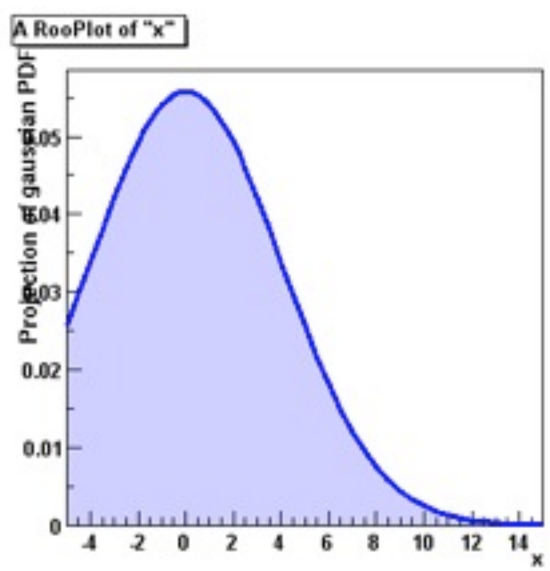
What is RooFit ?

- A toolkit distributed with ROOT and based on its core functionality.
- It is used to model distributions, which can be used for fitting and statistical data analysis.
  - model distribution of observable ***x*** in terms of parameters ***p***
    - probability density function (p.d.f.): $\mathcal{P}(x;p)$
    - p.d.f. are normalized over allowed range of observables ***x*** with respect to the parameters ***p***

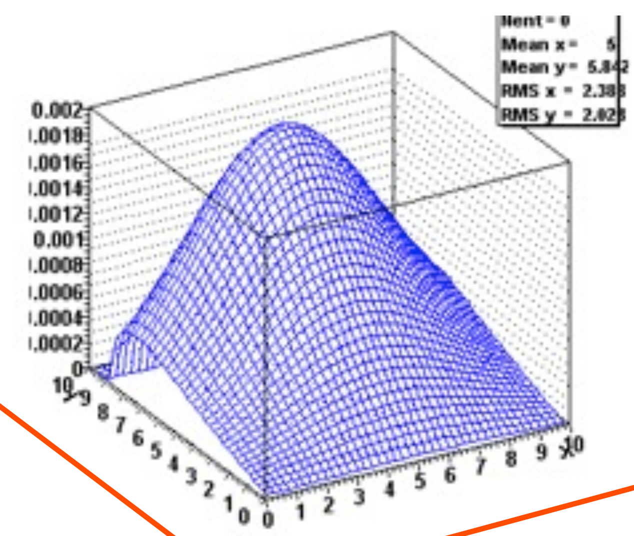
Mathematic – Probability density functions

- 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

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



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

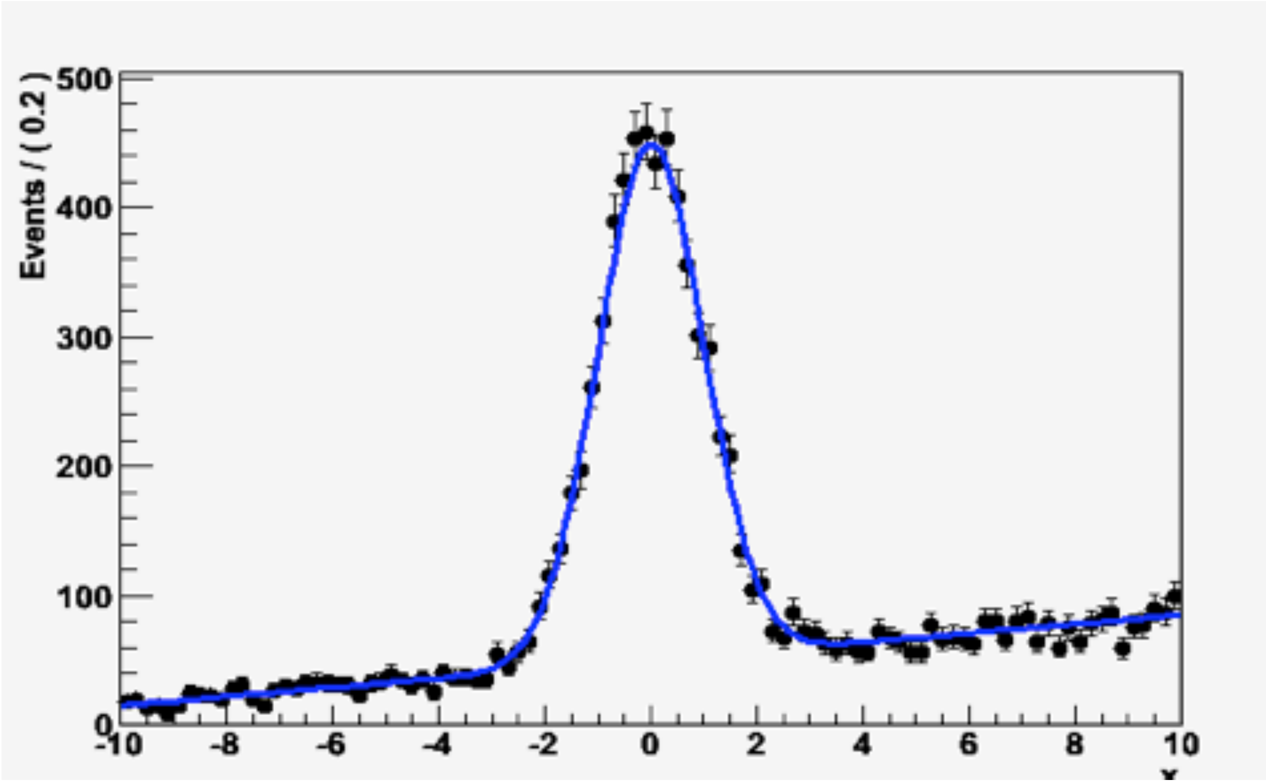


$$\int F(x,y)\,dx\,dy \equiv 1$$

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

Coding Probability Density Function

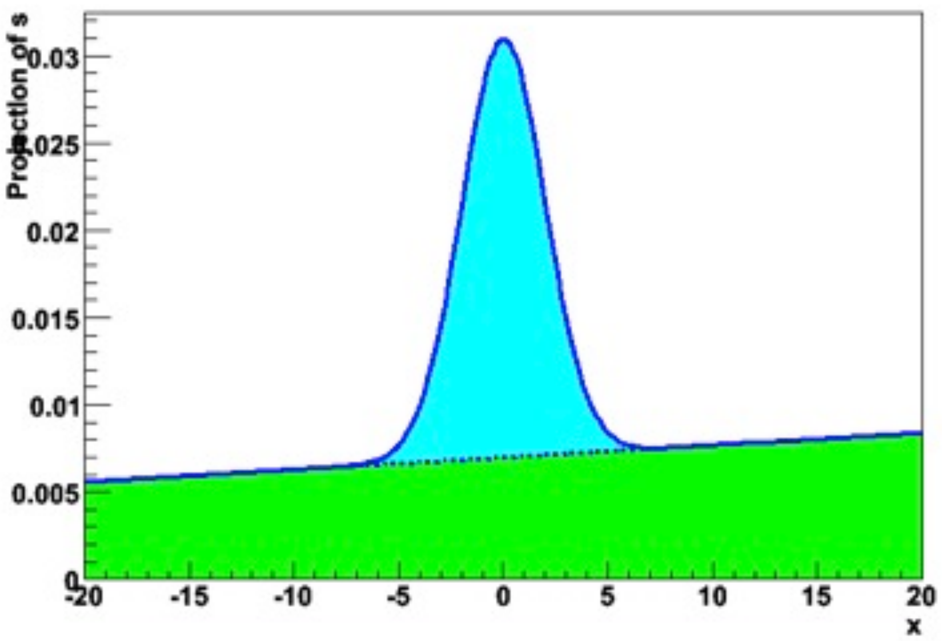
- How do we formulate the p.d.f. in ROOT
  - For ‘simple’ problems (gauss, polynomial) this is easy



  - But if we want to do complex likelihood fits using non-trivial functions and composing several p.d.f., or to work with multidimensional functions it is difficult to do it in ROOT
    - we need some tools to help us !

Math – Functions vs probability

- Why use *probability density* functions rather than 'plain' functions to model the data?
  - *Easier to interpret the models*.
    If Blue and Green pdf are each guaranteed to be normalized to 1, then fractions of Blue,Green can be cleanly interpreted as #events
  - Many statistical techniques only function properly with p.d.f.
    (e.g maximum likelihood fits)



- So why is not everybody always using them
  - The normalization can be hard to calculate
    (e.g. it can be different for each set of parameter values p)
  - In >1 dimension (numeric) integration can be particularly hard
  - RooFit aims to simplify these tasks

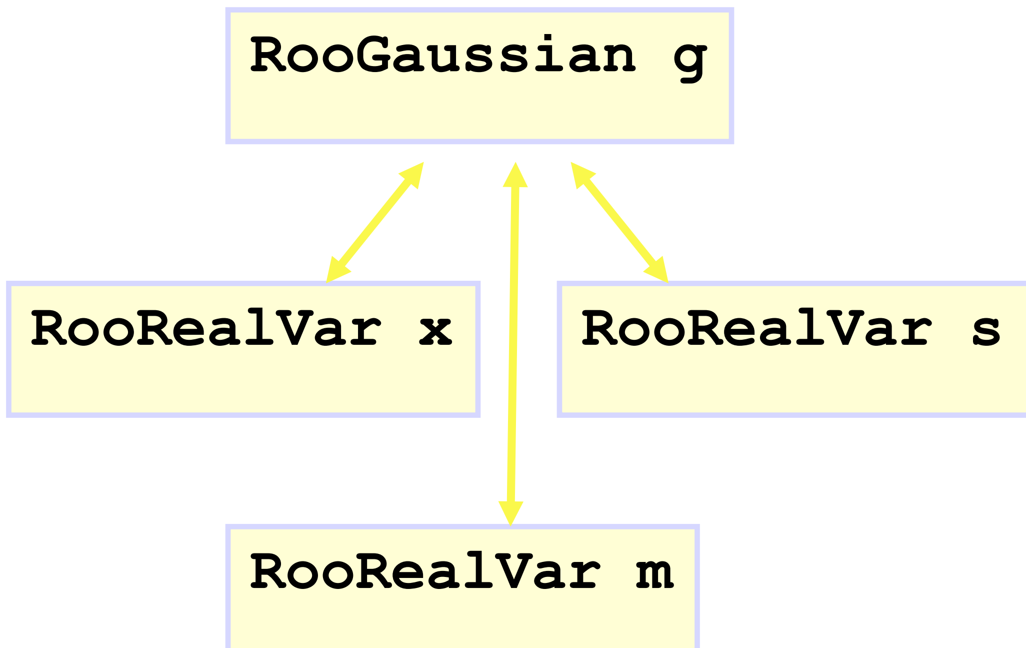
RooFit Modeling

Mathematical concepts 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`

RooFit Modeling

Example: Gaussian pdf $Gaus(x,m,s)$



RooFit code

```
RooRealVar x(“x”,”x”,2,-10,10)
RooRealVar s(“s”,”s”,3) ;
RooRealVar m(“m”,”m”,0) ;
RooGaussian g(“g”,”g”,x,m,s)
```

The simplest possible example

- We make a Gaussian p.d.f. with three variables: mass, mean and sigma

Name of object

Title of object

Initial range

Objects representing a 'real' value.

```
RooRealVar x("x","Observable",-10,10) ;
RooRealVar mean("mean","B0 mass",0.00027);
RooRealVar sigma("sigma","B0 mass width",5.2794) ;

RooGaussian model("model","signal pdf",x,mean,sigma)
```

Initial value

PDF object

References to variables

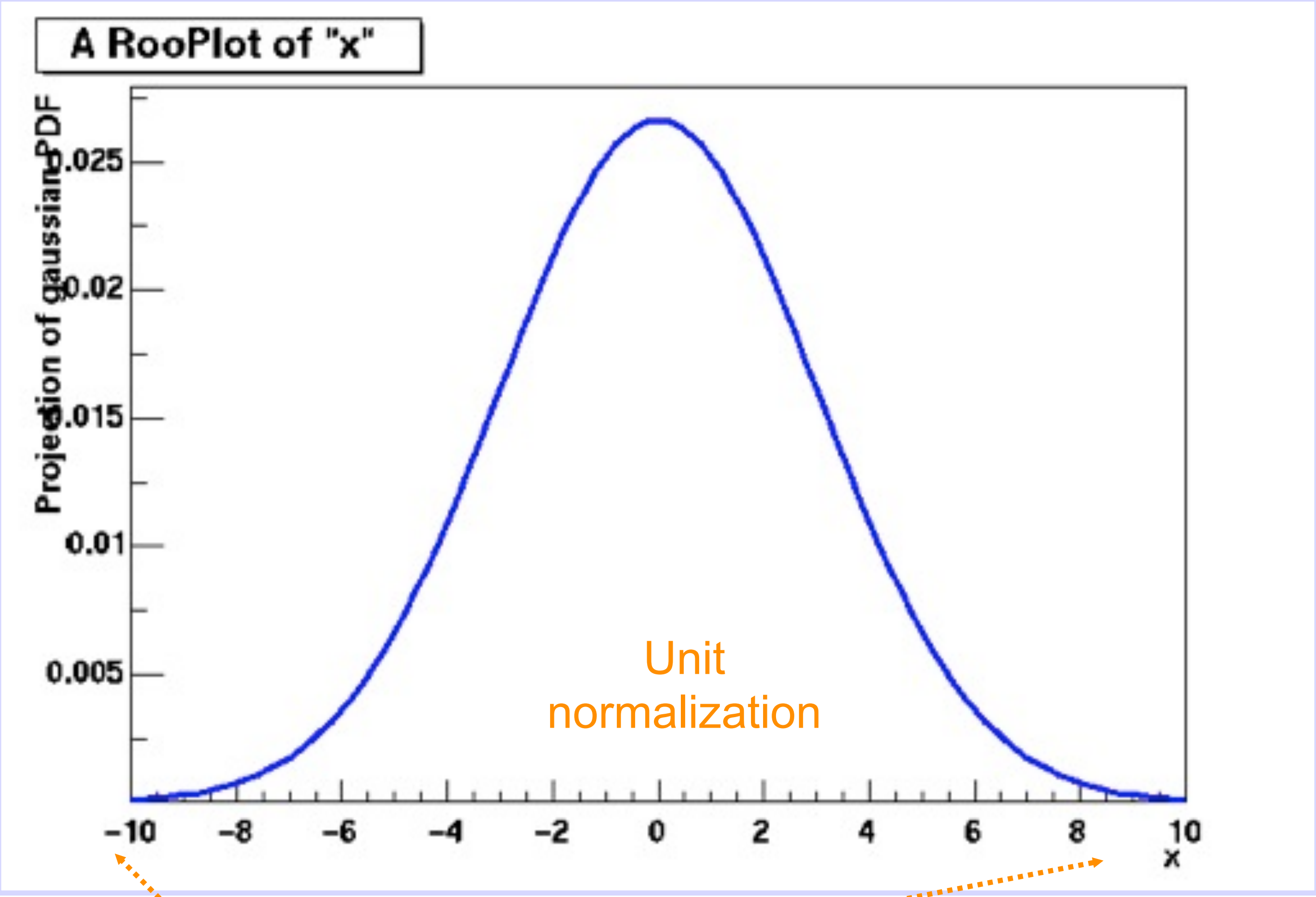
Creating and plotting a Gaussian p.d.f

Setup gaussian PDF and plot

```
// Create an empty plot frame
RooPlot* xframe = x.frame() ;

// Plot model on frame
model.plotOn(xframe) ;

// Draw frame on canvas
xframe->Draw() ;
```



Axis label from `gauss` title

A `RooPlot` is an empty frame capable of holding anything plotted versus it variable

Plot range taken from limits of `x`

Basics – Generating toy MC events

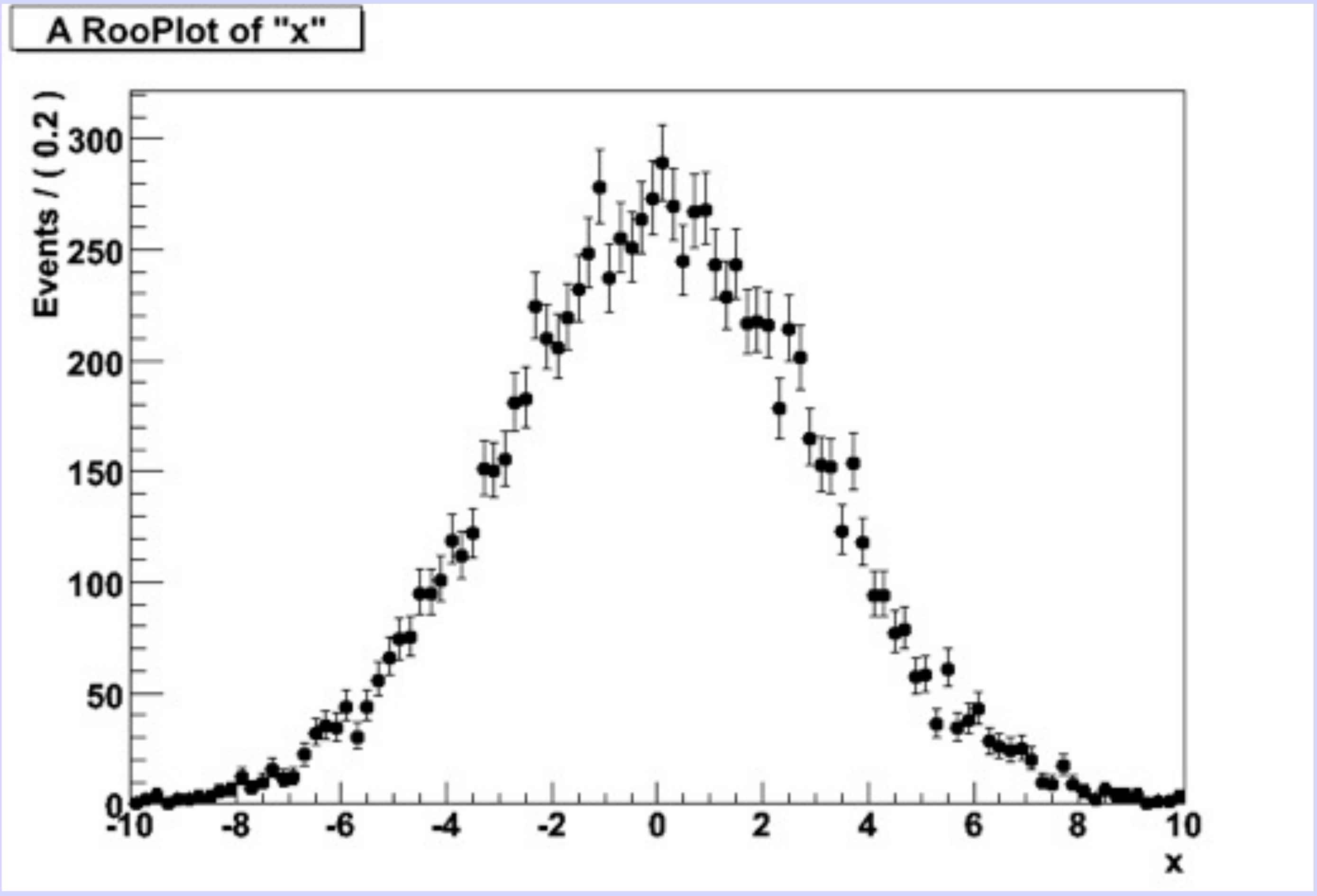
Generate 10000 events from Gaussian p.d.f and show distribution

```
// Generate an unbinned toy MC set
RooDataSet* data = gauss.generate(x,10000) ;

// Generate an binned toy MC set
RooDataHist* data = gauss.generateBinned(x,10000) ;

// Plot PDF
RooPlot* xframe = x.frame() ;
data->plotOn(xframe) ;
xframe->Draw() ;
```

Can generate both binned and unbinned datasets



Basics – Importing data

- Unbinned data can also be imported from ROOT `TTrees`

```
// Import unbinned data
RooDataSet data("data","data",x,Import(*myTree)) ;
```

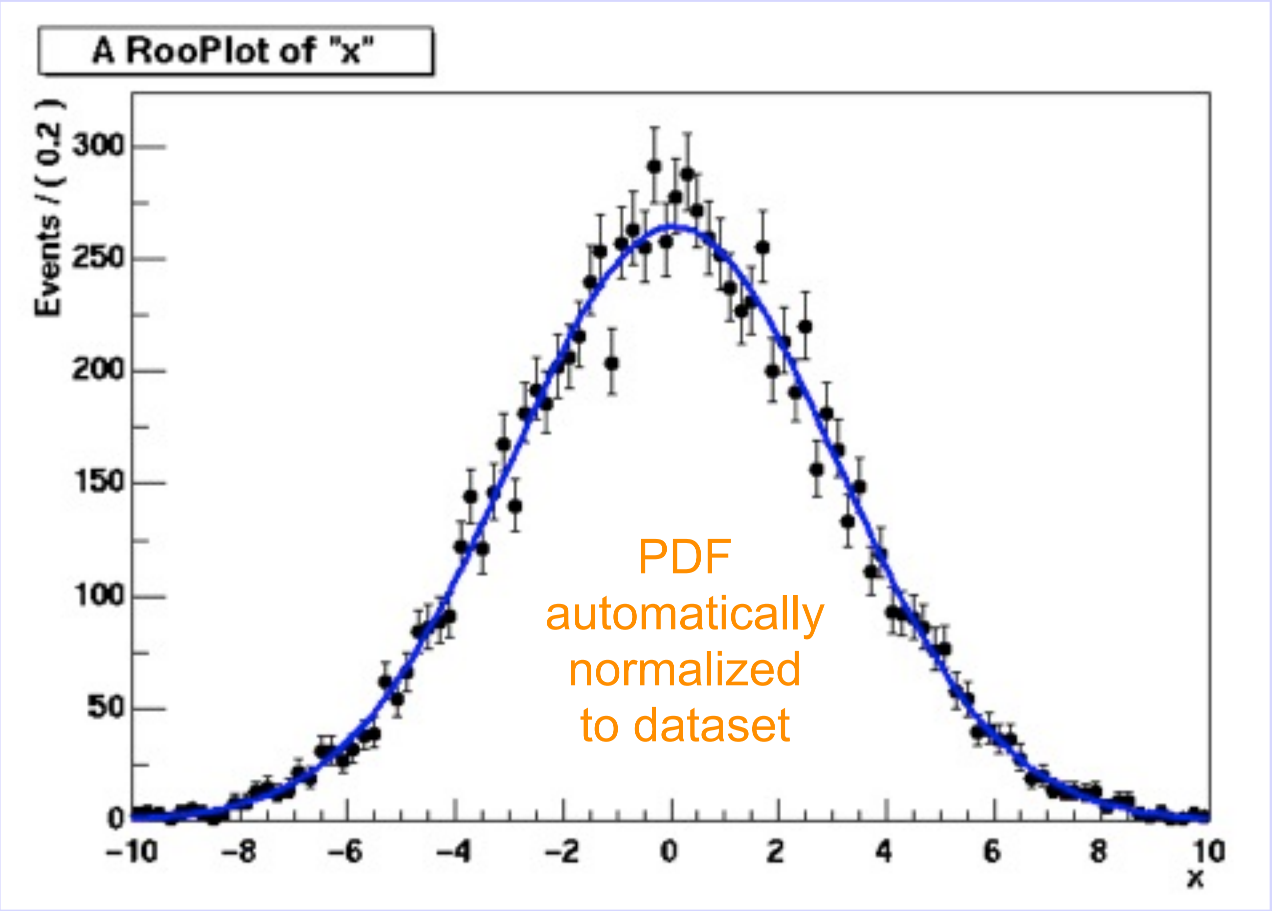
  - Imports `TTree` branch named "x".
  - Can be of type `Double_t`, `Float_t`, `Int_t` or `UInt_t`. All data is converted to Double_t internally
  - Specify a `RooArgSet` of multiple observables to import multiple observables

- Binned data can be imported from ROOT `THx` histograms

```
// Import unbinned data
RooDataHist data("data","data",x,Import(*myTH1)) ;
```

  - Imports values, binning definition *and* errors (if defined)
  - Specify a `RooArgList` of observables when importing a TH2/3.

Fit of p.d.f to the data




```
// ML fit of gauss to data
gauss.fitTo(*data) ;
(Minimization printout omitted)

// Parameters if gauss now
// reflect fitted values
mean.Print()
RooRealVar::mean = 0.0172335 +/- 0.0299542
sigma.Print()
RooRealVar::sigma = 2.98094  +/- 0.0217306

// Plot fitted PDF and toy data overlaid
RooPlot* xframe = x.frame() ;
data->plotOn(xframe) ;
gauss.plotOn(xframe) ;
```

RooFit Factory

```
RooRealVar x("x","x",2,-10,10)
RooRealVar s("s","s",3) ;
RooRealVar m("m","m",0) ;
RooGaussian g("g","g",x,m,s)
```

Provides a factory to auto-generate objects from a math-like language

```
RooWorkspace w;
w.factory("Gaussian::g(x[2,-10,10],m[0],s[3])")
```

We will work in the example and exercises using the workspace factory to build models

RooWorkspace

- Workspace class in RooFit (**`RooWorkSpace`**) with:
  - full model configuration
    - PDF and parameter/observables descriptions
    - uncertainty/shape of nuisance parameters
  - (multiple) data sets
- Maintain a complete description of all the model
  - possibility to save entire model in a ROOT file
- Combination of results joining workspaces in a single one
- All information is available for further analysis
  - common format for combining and sharing physics results

```
RooWorkspace workspace("Example_workspace");
workspace.import(*data);
workspace.import(*pdf);
workspace.defineSet("obs","x");
workspace.defineSet("poi","mu");
workspace.importClassCode();
workspace.writeToFile("myWorkspace")
```

Using the workspace

- Workspace
  - A generic container class for all RooFit objects of your project
  - Helps to organize analysis projects
- Creating a workspace

```
RooWorkspace w("w") ;
```

- Putting variables and functions into a workspace
  - When importing a function, all its components (variables) are automatically imported too

```
RooRealVar x("x","x",-10,10) ;
RooRealVar mean("mean","mean",5) ;
RooRealVar sigma("sigma","sigma",3)  ;
RooGaussian f("f","f",x,mean,sigma) ;

// imports f,x,mean and sigma
w.import(f) ;
```

Using the workspace

- Looking into a workspace

```
w.Print() ;

variables
---------
(mean,sigma,x)

p.d.f.s
-------
RooGaussian::f[ x=x mean=mean sigma=sigma ] = 0.249352
```

- Getting variables and functions out of a workspace

```
// Variety of accessors available
RooPlot* frame = w.var("x")->frame() ;
w.pdf("f")->plotOn(frame) ;
```

Using the workspace

- Workspace can be written to a file with all its contents
  - Writing workspace and contents to file

```
w.writeToFile(“wspace.root”) ;
```

- Organizing your code – Separate construction and use of models

```
void driver() {
  RooWorkspace w(“w”) ;
  makeModel(w) ;
  useModel(w) ;
}

void makeModel(RooWorkspace& w) {
  // Construct model here
}

void useModel(RooWorkspace& w) {
  // Make fit, plots etc here
}
```

Factory and Workspace

- *One C++ object per math symbol* provides ultimate level of control over each objects functionality, but results in lengthy user code for even simple macros
- Solution: add factory that auto-generates objects from a math-like language. Accessed through factory() method of workspace
- Example: reduce construction of Gaussian pdf and its parameters from 4 to 1 line of code

```
w.factory("Gaussian::f(x[-10,10],mean[5],sigma[3])") ;
```

```
RooRealVar x("x","x",-10,10) ;
RooRealVar mean("mean","mean",5) ;
RooRealVar sigma("sigma","sigma",3)  ;
RooGaussian f("f","f",x,mean,sigma) ;
```

Factory syntax

- Rule #1 – Create a variable

```
x[-10,10]    // Create variable with given range
x[5,-10,10]  // Create variable with initial value and range
x[5]         // Create initially constant variable
```

- Rule #2 – Create a function or pdf object

```
ClassName::Objectname(arg1,[arg2],...)
```

  – Leading 'Roo' in class name can be omitted
  – Arguments are names of objects that already exist in the workspace
  – Named objects must be of correct type, if not factory issues error
  – Set and List arguments can be constructed with brackets {}

```
Gaussian::g(x,mean,sigma)
// equivalent to RooGaussian("g","g",x,mean,sigma)

Polynomial::p(x,{a0,a1})
// equivalent to  RooPolynomial("p","p",x",RooArgList(a0,a1));
```

Factory syntax

- Rule #3 – Each creation expression returns the name of the object created
  - Allows to create input arguments to functions 'in place' rather than in advance

```
Gaussian::g(x[-10,10],mean[-10,10],sigma[3])
//-->   x[-10,10]
//      mean[-10,10]
//      sigma[3]
//      Gaussian::g(x,mean,sigma)
```

- Miscellaneous points
  - You can always use numeric literals where values or functions are expected

```
Gaussian::g(x[-10,10],0,3)
```

  - It is not required to give component objects a name, e.g.

```
SUM::model(0.5*Gaussian(x[-10,10],0,3),Uniform(x)) ;
```

Time for Exercises!

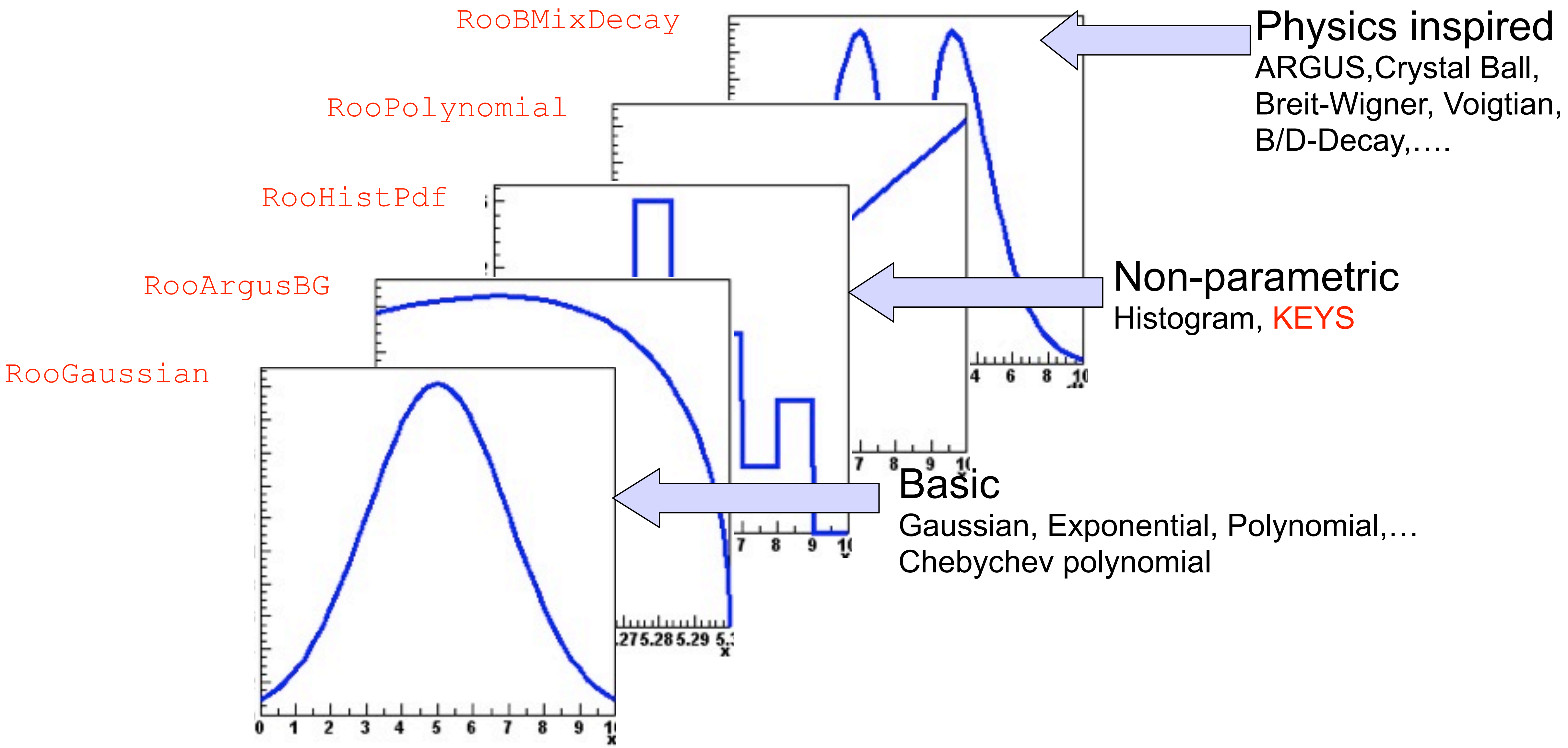
Put in practice the concepts to which you were just exposed: read the instructions here

https://twiki.cern.ch/twiki/bin/view/Main/RootIRMMTutorial2013RooFitExercises

and solve exercise 1 and 2

Model building

- RooFit provides a collection of compiled standard PDF classes



Easy to extend the library: each p.d.f. is a separate C++ class

(Re)using standard components

- List of most frequently used pdfs and their factory spec

Gaussian	`Gaussian::g(x,mean,sigma)`
Breit-Wigner	`BreitWigner::bw(x,mean,gamma)`
Landau	`Landau::l(x,mean,sigma)`
Exponential	`Exponential::e(x,alpha)`
Polynomial	`Polynomial::p(x,{a0,a1,a2})`
Chebychev	`Chebychev::p(x,{a0,a1,a2})`
Kernel Estimation	`KeysPdf::k(x,dataSet)`
Poisson	`Poisson::p(x,mu)`
Voigtian (=BW⊗G)	`Voigtian::v(x,mean,gamma,sigma)`

Making your own Model

- Interpreted expressions

```
w.factory("EXPR::mypdf('sqrt(a*x)+b',x,a,b)") ;
```

- Customized class, compiled and linked on the fly

```
w.factory("CEXPR::mypdf('sqrt(a*x)+b',x,a,b)") ;
```

- Custom class written by you
  - Offer option of providing analytical integrals, custom handling of toy MC generation (details are given in RooFit User Guide)
- Compiled classes are faster in use, but require O(1-2) seconds startup overhead
  - Best choice depends on usage context

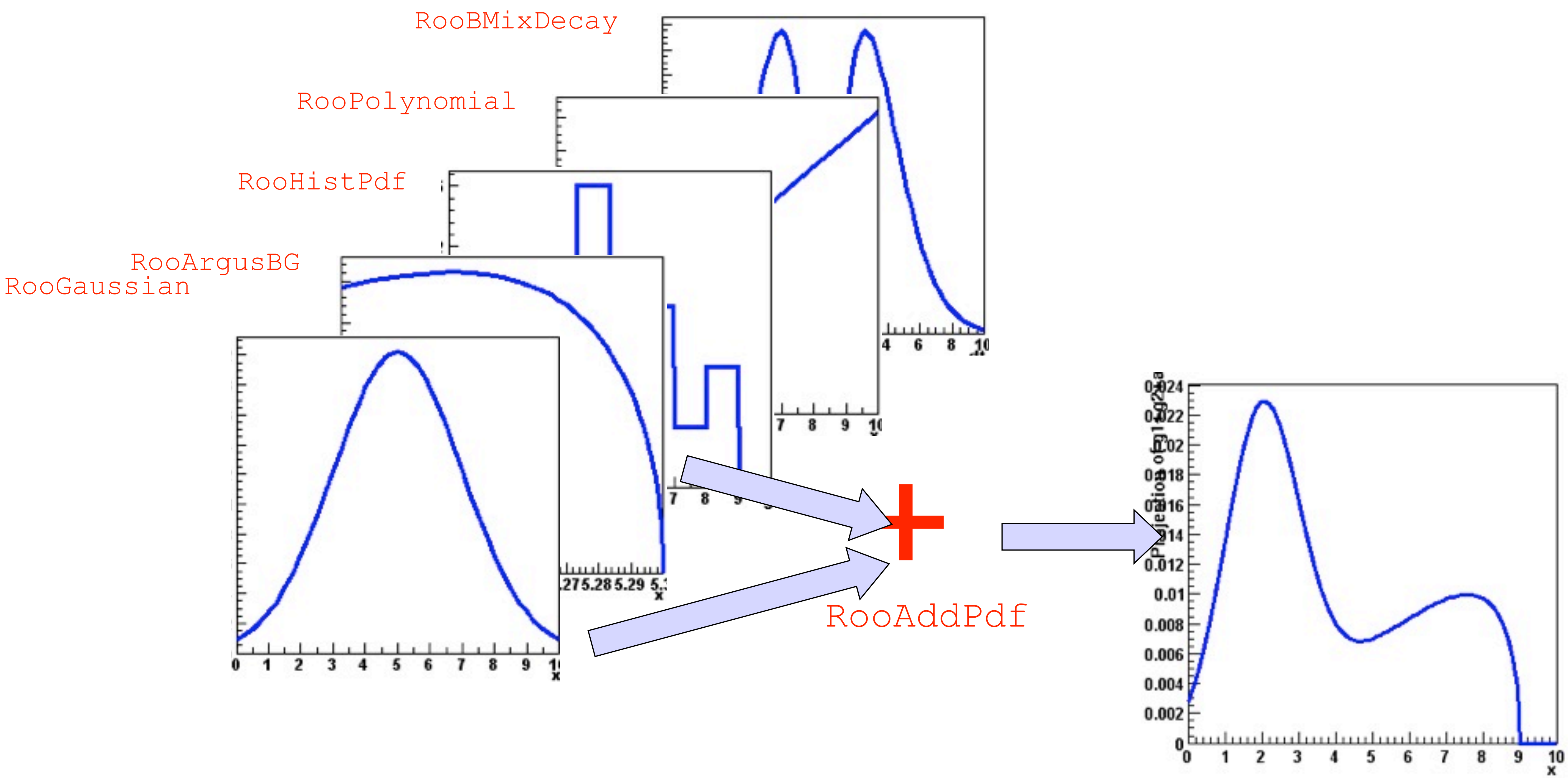
Adjusting Model Parameterization

- RooFit pdf classes do not require their parameter arguments to be variables, one can plug in functions as well.
- Simplest tool performing reparameterization is the interpreted formula expression

```
w.factory("expr::w('(1-D)/2',D[0,1])") ;
```

  – Note lower case: **expr** builds function, **EXPR** builds pdf

- 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`**



Adding p.d.f.s – Factory syntax

- Additions created through a SUM expression

```
SUM::name(frac1*PDF1,PDFN)
```

$$S(x) = fF(x) + (1 - f)G(x)$$

```
SUM::name(frac1*PDF1,frac2*PDF2,...,PDFN)
```

  - Note that last PDF does not have an associated fraction in case of floating overall normalization
    - when the normalization is fitted from the observed events

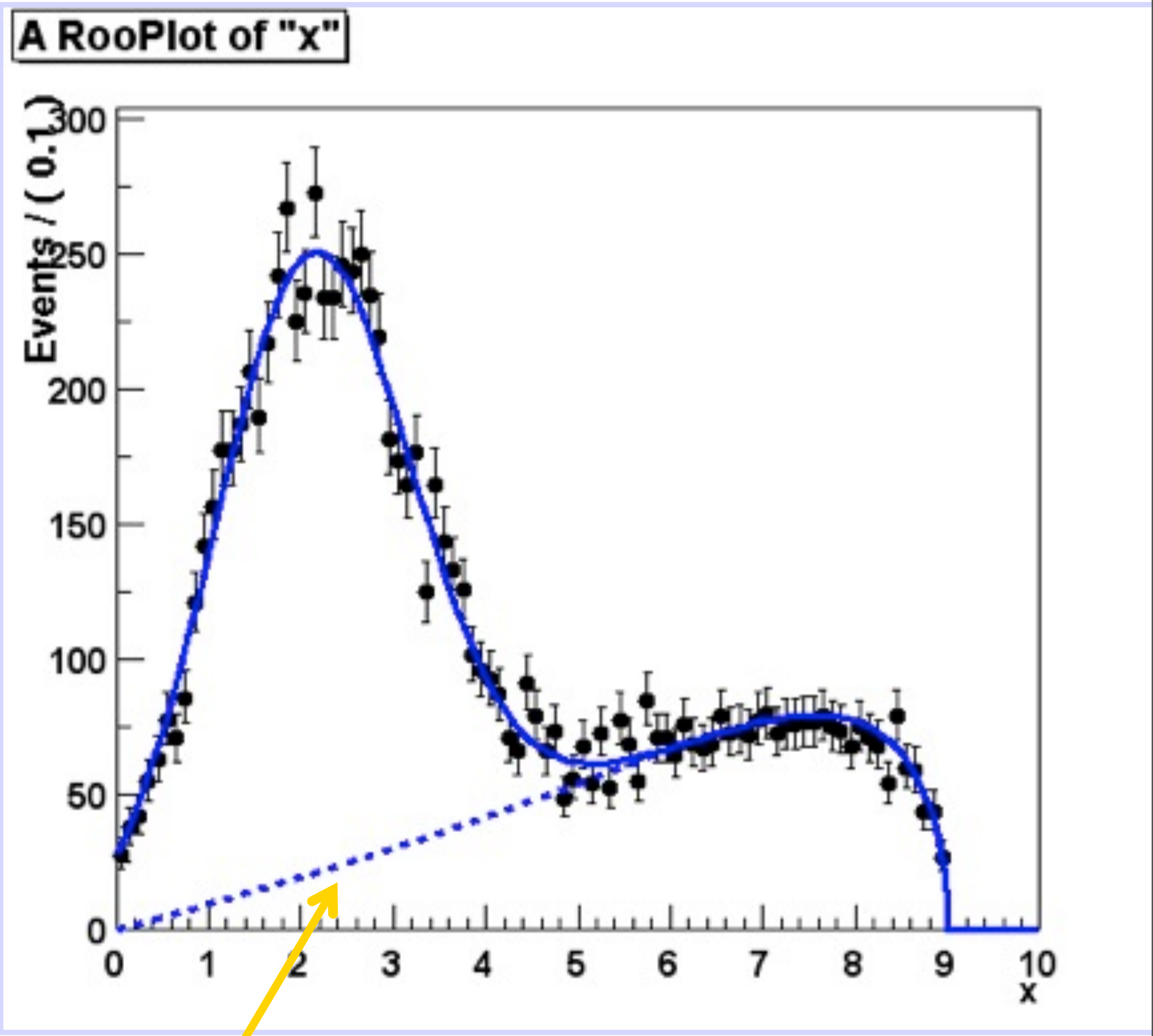
- Complete example

```
w.factory("Gaussian::gauss1(x[0,10],mean1[2],sigma[1]") ;
w.factory("Gaussian::gauss2(x,mean2[3],sigma)") ;
w.factory("ArgusBG::argus(x,k[-1],9.0)") ;

w.factory("SUM::sum(g1frac[0.5]*gauss1, g2frac[0.1]*gauss2, argus)")
```

Plotting Components of a p.d.f.

- Plotting, toy event generation and fitting works identically for composite p.d.f.s
  - Several optimizations applied behind the scenes that are specific to composite models (e.g. delegate event generation to components)
- Extra plotting functionality specific to composite p.d.f.s
  - Component plotting




```
// Plot only argus components
w::sum.plotOn(frame,Components("argus"),LineStyle(kDashed)) ;

// Wildcards allowed
w::sum.plotOn(frame,Components("gauss*"),LineStyle(kDashed)) ;
```

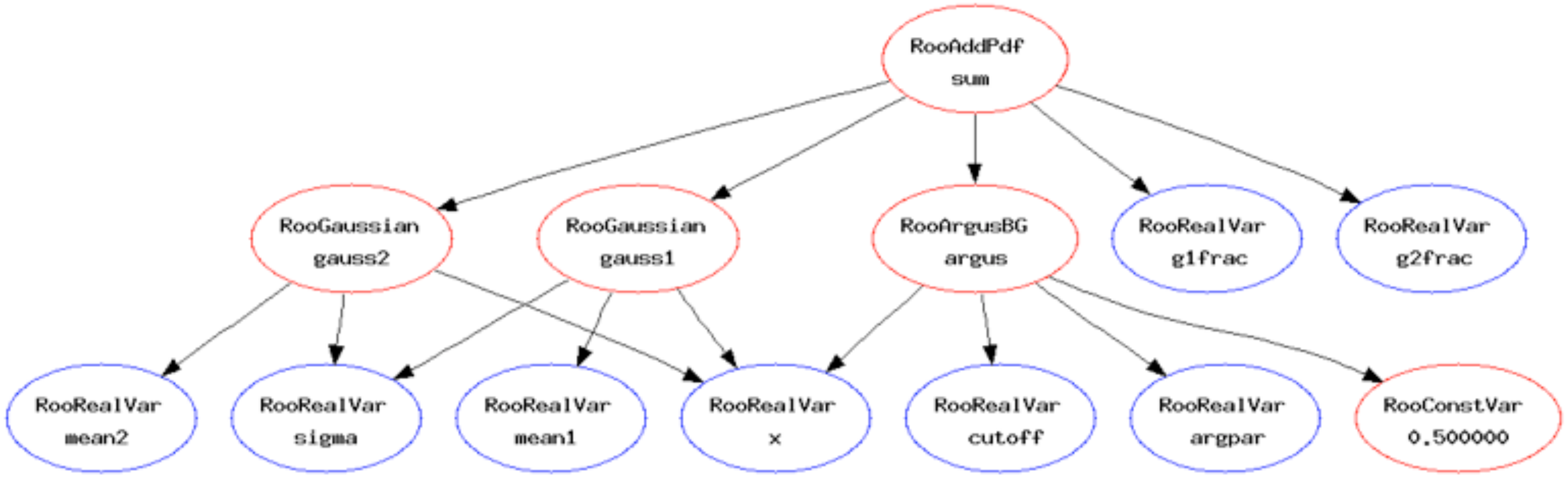
Operations on specific to composite pdfs

- Tree printing mode of workspace reveals component structure

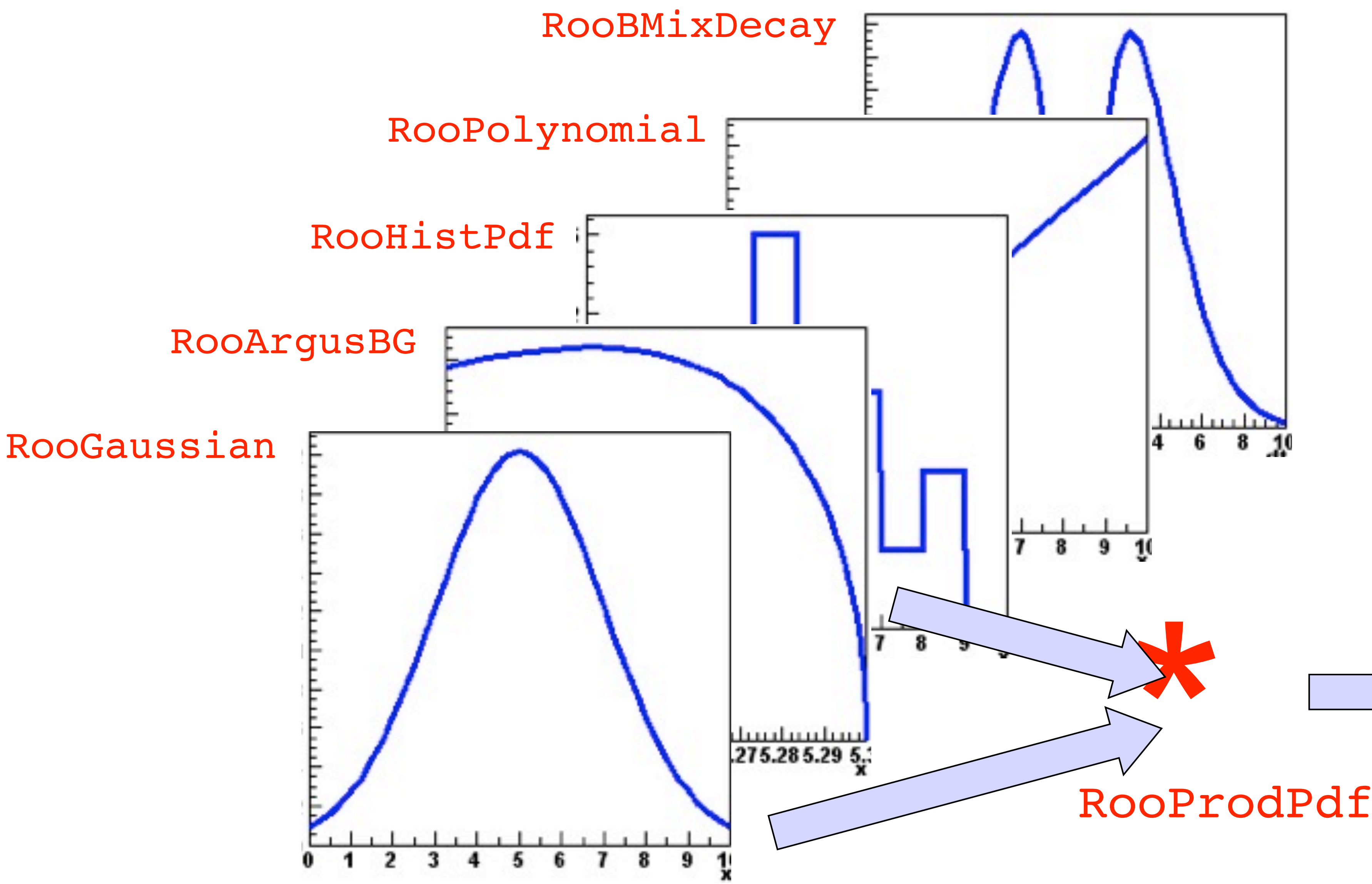
```
w.pdf("sum")->Print("t");
  RooAddPdf::sum[ g1frac * g1 + g2frac * g2 + [%] * argus ] = 0.0687785
      RooGaussian::g1[ x=x mean=mean1 sigma=sigma ] = 0.135335
      RooGaussian::g2[ x=x mean=mean2 sigma=sigma ] = 0.011109
      RooArgusBG::argus[ m=x m0=k c=9 p=0.5 ] = 0
```

  – Can also make input files for GraphViz visualization

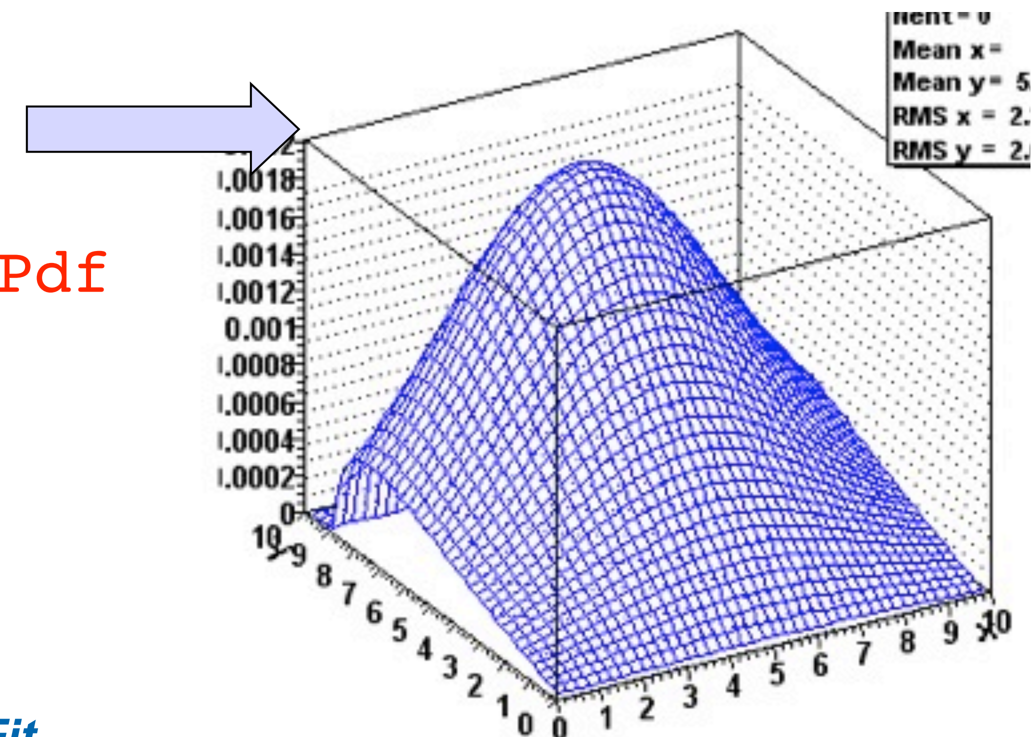
```
w.pdf("sum")->graphVizTree("myfile.dot");
```



Products of uncorrelated p.d.f.s



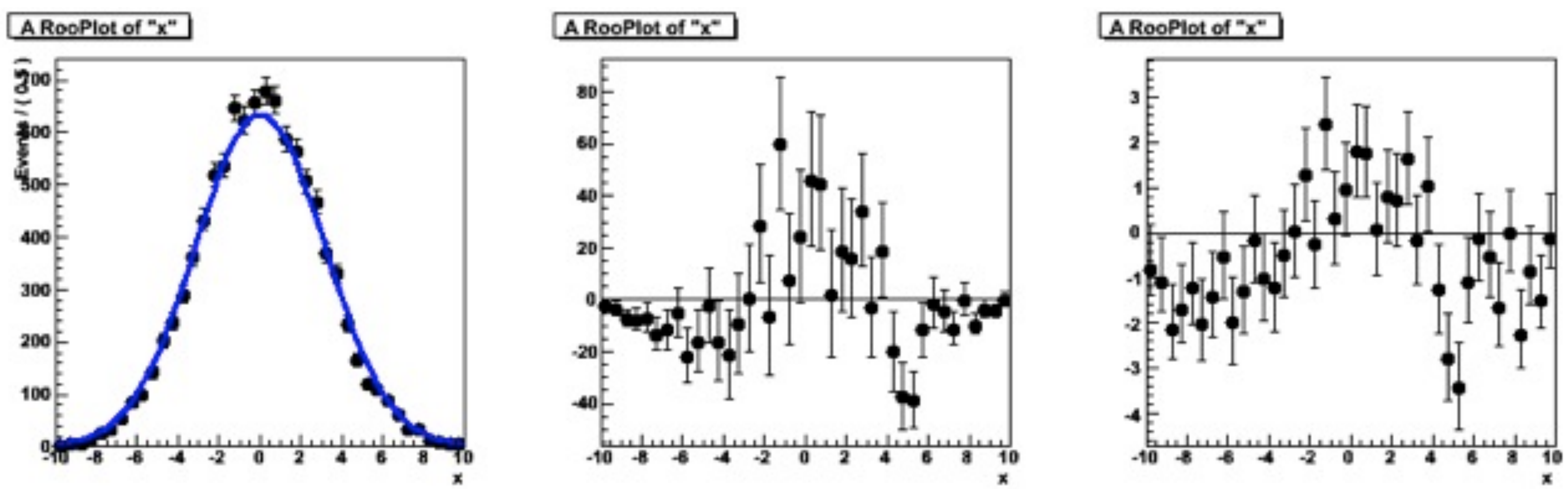
$$H(x,y) = F(x) \times G(y)$$



How do you know if your fit was ‘good’

- Goodness-of-fit broad issue in statistics in general, will just focus on a few specific tools implemented in RooFit here
- For one-dimensional fits, a χ^2 is usually the right thing to do
  - Some tools implemented in `RooPlot` to be able to calculate χ^2/`ndf` of curve w.r.t data

```
double chi2 = frame->chisquare(nFloatParam);
```

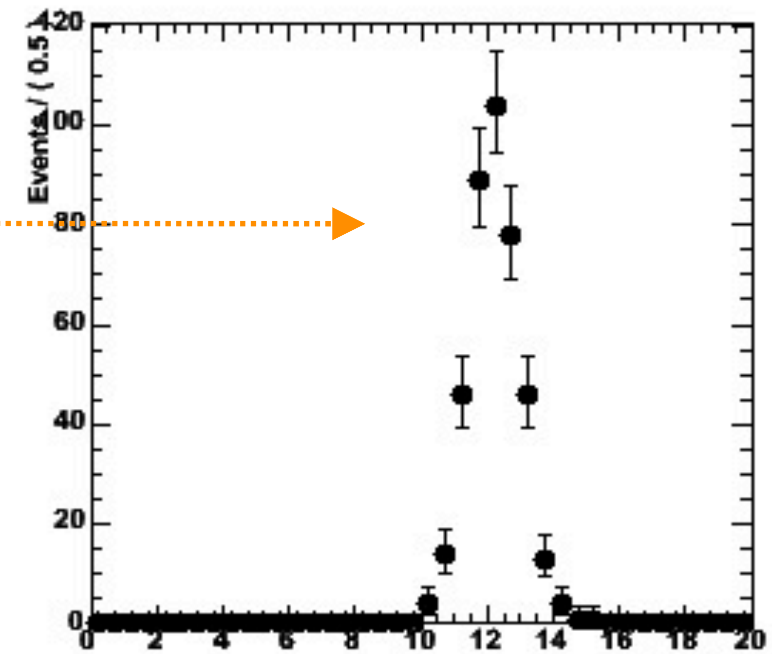


  - Also tools exists to plot residual and pull distributions from curve and histogram in a RooPlot

```
frame->makePullHist();
frame->makeResidHist();
```

Fit Validation Study – The pull distribution

- What about the validity of the error?
  - Distribution of error from simulated experiments is difficult to interpret…
  - We don't have equivalent of N_{sig}(generated) for the error

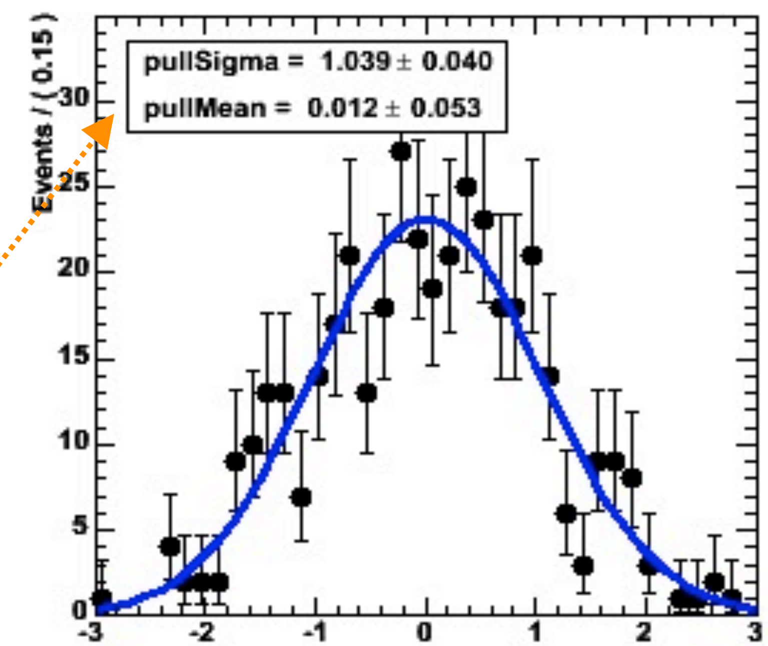


$\sigma(N_{sig})$

- Solution: look at the ***pull distribution***
  - Definition:

$$\mathrm{pull}(N_{sig}) = \frac{N_{sig}^{fit} - N_{sig}^{true}}{\sigma_N^{fit}}$$

  - Properties of pull:
    - **Mean is 0 if there is no bias**
    - **Width is 1 if error is correct**
  - In this example: no bias, correct error within statistical precision of study



pull(N_{sig})

Composition of p.d.f.s

- RooFit pdf building blocks do not require variables as input, just real-valued functions
  - Can substitute any variable with a function expression in parameters and/or observables

$$f(x;p) \Rightarrow f(x,p(y,q)) = f(x,y;q)$$

  - Example: Gaussian with shifting mean

```
w.factory("expr::mean('a*y+b',y[-10,10],a[0.7],b[0.3])") ;
w.factory("Gaussian::g(x[-10,10],mean,sigma[3])") ;
```

  - No assumption made in function on a,b,x,y being observables or parameters, any combination will work

RooFit Summary

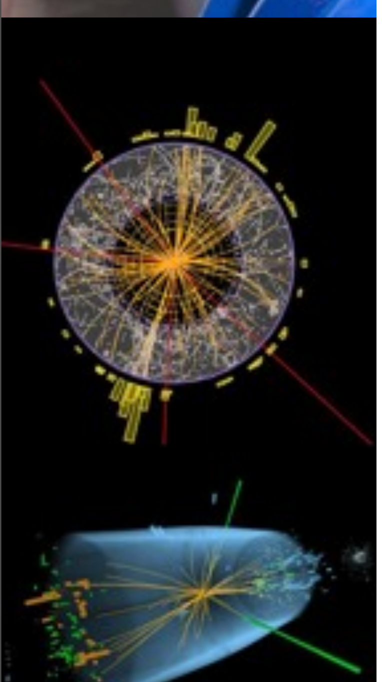
- We have learned how to build a RooFit model which can then be used to fit observed data sets.
- We have also learned about the workspace class which can be used to:
  - build models with the factory syntax
  - storing and sharing the models for further use (e.g. combination of results)
- We will briefly see now how this functionality will be used by the RooStats statistical framework

Time for Exercises!

Put in practice the concepts to which you were just exposed: read the instructions here

https://twiki.cern.ch/twiki/bin/view/Main/RootIRMMTutorial2013RooFitExercises

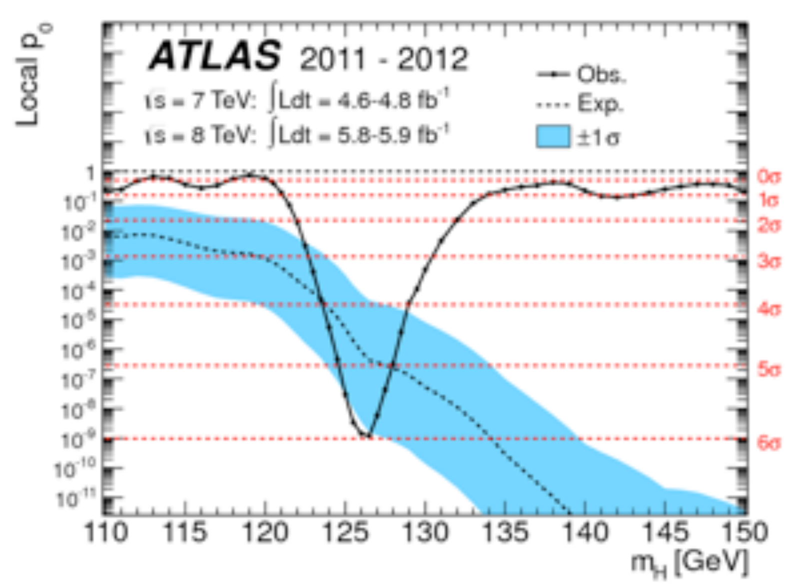
and solve exercise 3

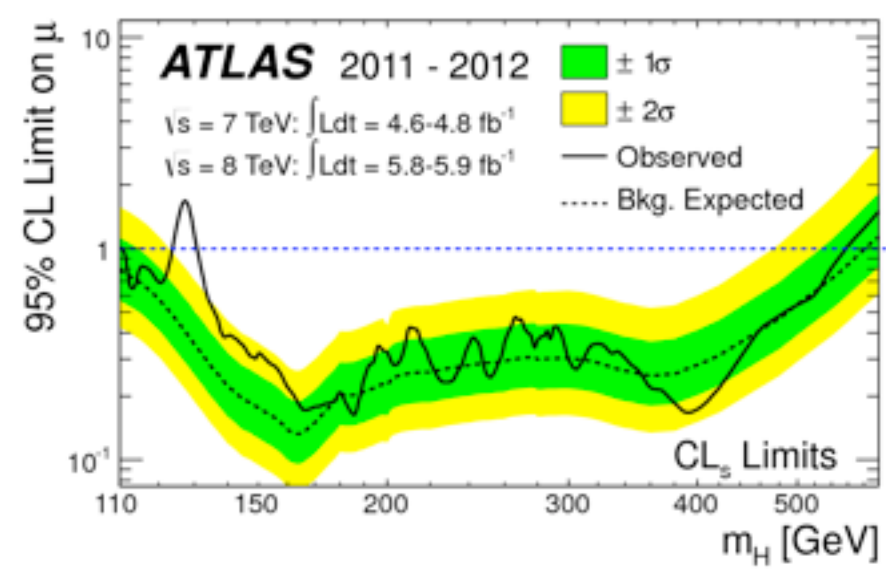


Introduction to RooStats

ROOT Training at IRMM

1st March 2013





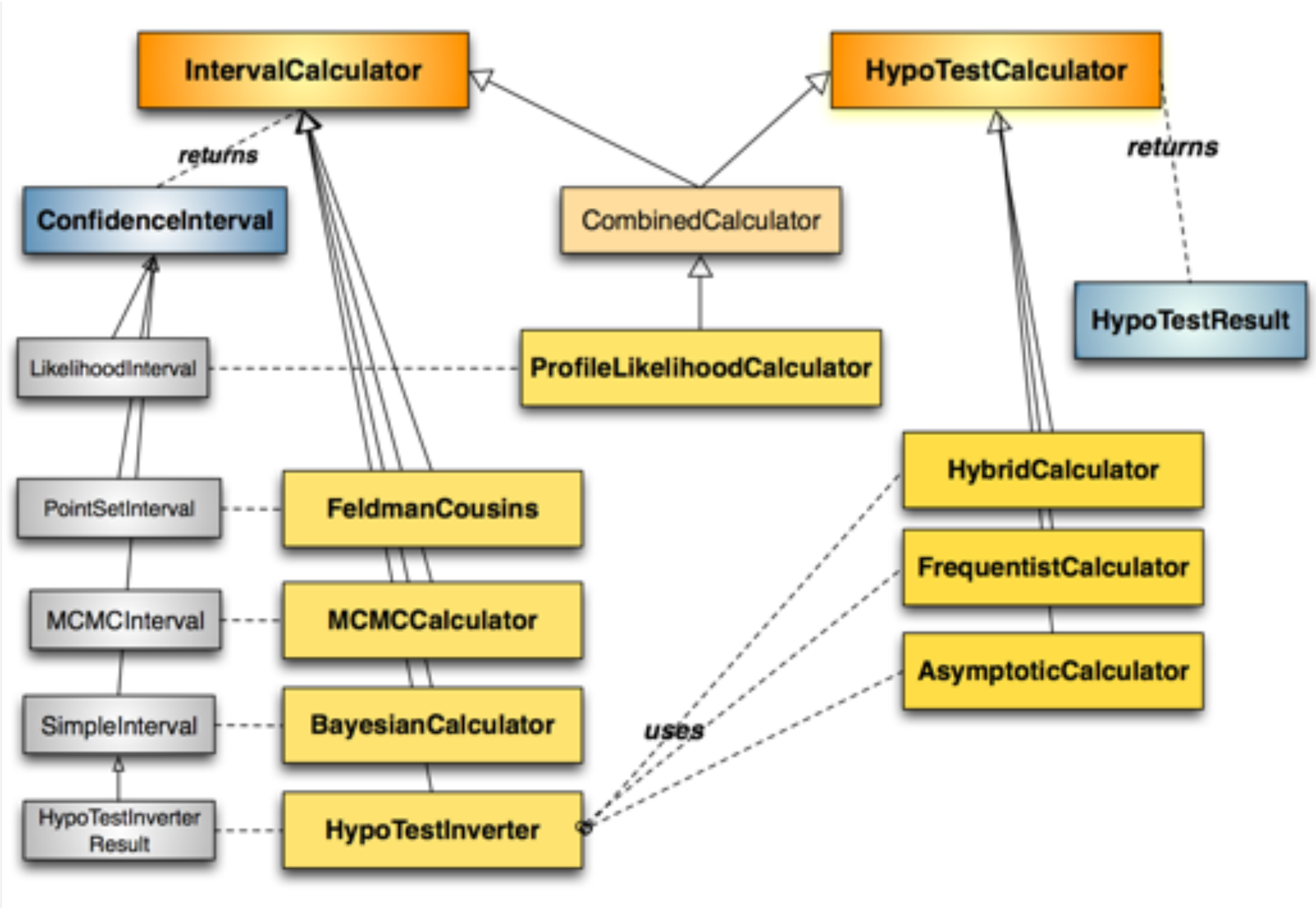
Dr Lorenzo Moneta
CERN PH-SFT
CH-1211 Geneva 23
sftweb.cern.ch
root.cern.ch

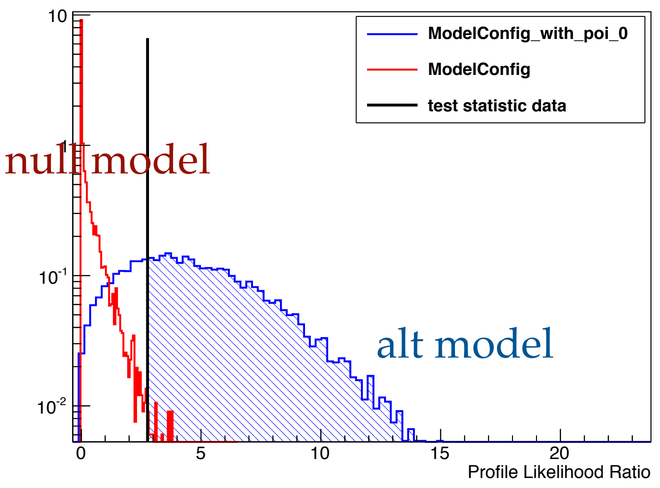
RooStats

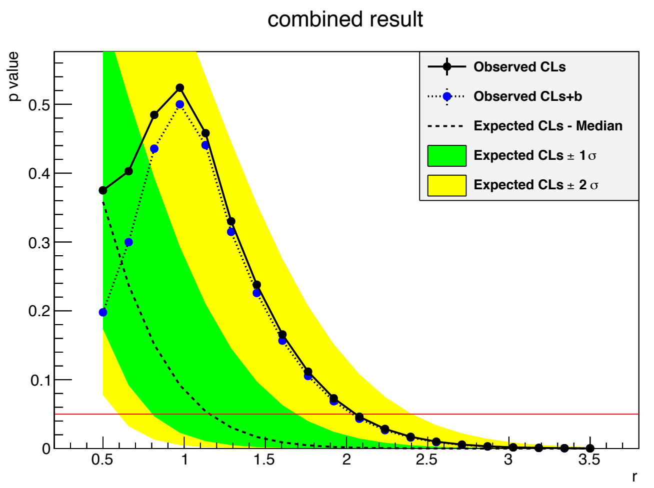
- Goals of RooStats:
  - Provide a common framework for statistical calculations
    - work on arbitrary models and datasets
    - implement most accepted techniques
      - frequentists, Bayesian and likelihood based tools
    - possible to easy compare different statistical methods
    - provide utility for combinations of results
    - using same tools across experiments facilities combinations of results
- Common Purposes:
  - estimation of confidence (credible) intervals
    - multi-dimensional contours or just a lower/higher limit
  - hypothesis tests: evaluation of p-value for one or multiple hypotheses (discovery significance)

RooStats Design

- C++ classes and interfaces mapping statistical concepts
  - Calculators for interval estimation (based on the Likelihood, Bayesian or Frequentist statistics)
  - Calculator for hypothesis test (Likelihood or Frequentist).







Example: Bayesian Analysis

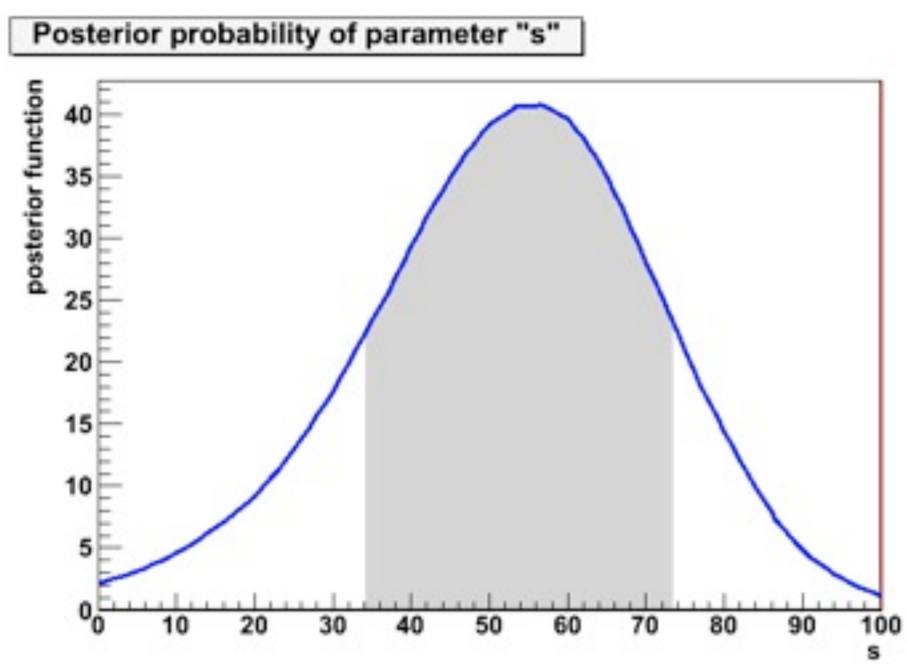
- **`RooStats`** provides classes for
  - marginalize posterior and estimate credible interval

posterior probability, likelihood function, prior probability, nuisance parameters marginalization

$$\underbrace{P(\mu|x)}_{\text{POI} \quad \text{data}} = \frac{\int L(x|\mu,\nu)\Pi(\mu,\nu)d\nu}{\underbrace{\int\int L(x|\mu,\nu)\Pi(\mu,\nu)d\mu d\nu}_{\text{normalisation term}}}$$

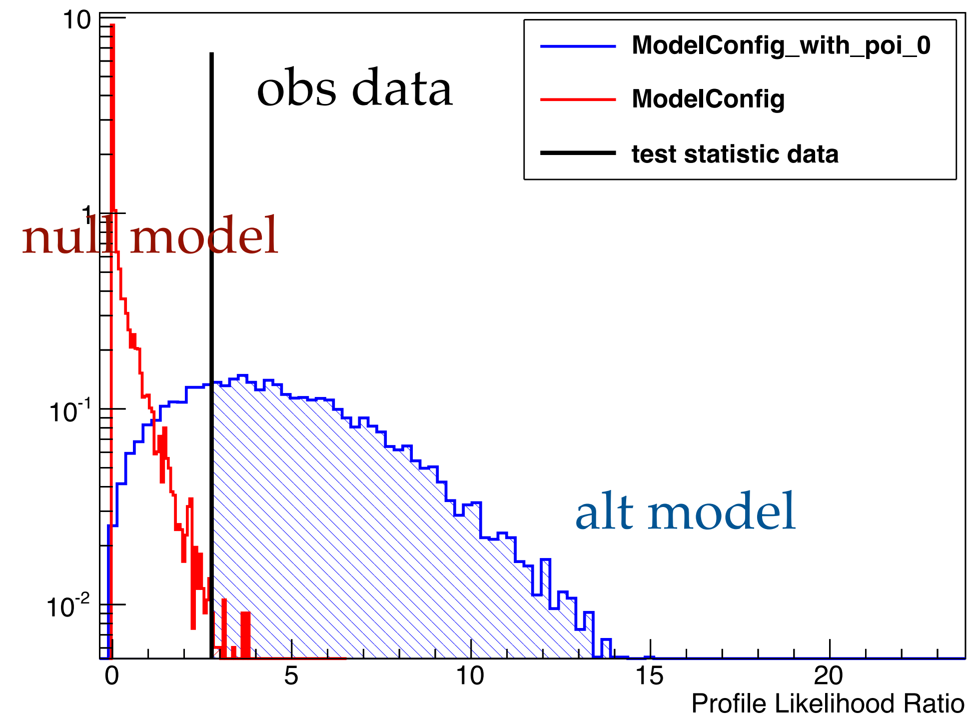
Bayes Theorem

  - support for different integration algorithms:
    - adaptive (numerical)
    - MC integration
    - Markov-Chain
  - can work with models with many parameters (e.g few hundreds)



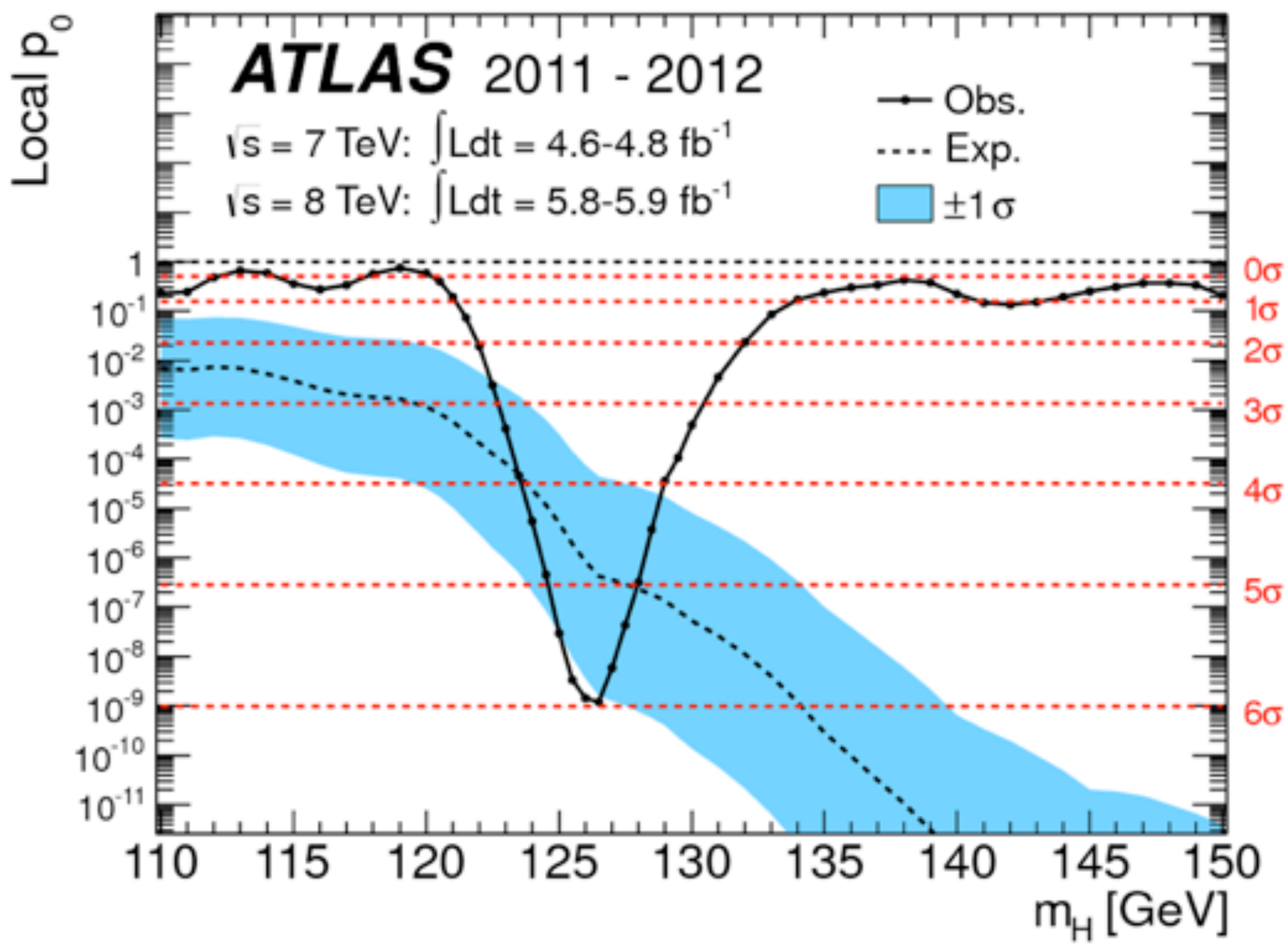
Example: Hypothesis Tests

- Frequentist hypothesis test in RooStats
- Example: Compute discovery significance
  - we need to define first:
    - null hypothesis: no signal, background only
    - alternate hypothesis: signal is present with background
    - test statistics: a function of the data needed to compute the p-values (e.g. a χ^2 or a likelihood-ratio).
  - generate pseudo-experiments for the null and alternate model to get the test statistics distributions
  - from the observed data value of the test statistics:
    - compute p-value for the null-model (p_0)
    - translate in a discovery significance



Example: Significance of Discovery

- Performing the tests for different mass hypotheses (*i.e* different signal models)



Summary

- We have learned what are RooFit and RooStats and how can be used for advance statistical data analysis
  - estimate parameters of a model and their confidence intervals
  - test of hypothesis for estimating the significance of discovery
- These are the tools currently used by the LHC experiments to produce their final results:
  - Observation of Higgs Bosons by the ATLAS and CMS experiments