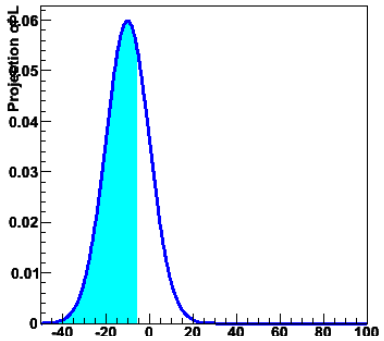
Advanced Statistics Course – Part II

W. Verkerke (NIKHEF)

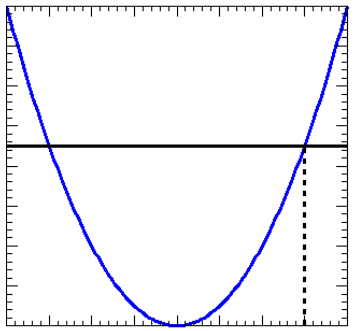
Setting limits 3 ways

U.L. in Poisson Process, n=3 observed: 3 ways

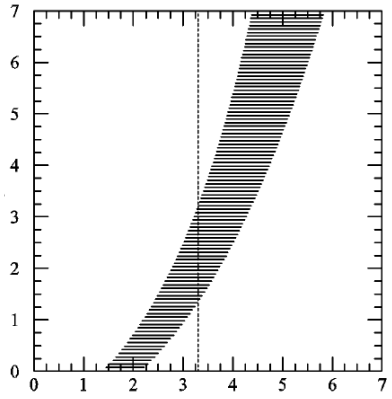
- Bayesian interval
  at 90% credibility:
  find μ_u such that posterior probability $p(\mu>\mu_u) = 0.1$.



- Likelihood ratio method for approximate 90% C.L. U.L.:
  find μ_u such that $L(\mu_u) / L(3)$ has prescribed value.



- Frequentist one-sided 90% C.L. upper limit: find μ_u such that $P(n\leq 3 \mid \mu_u) = 0.1$.



Main focus of this morning

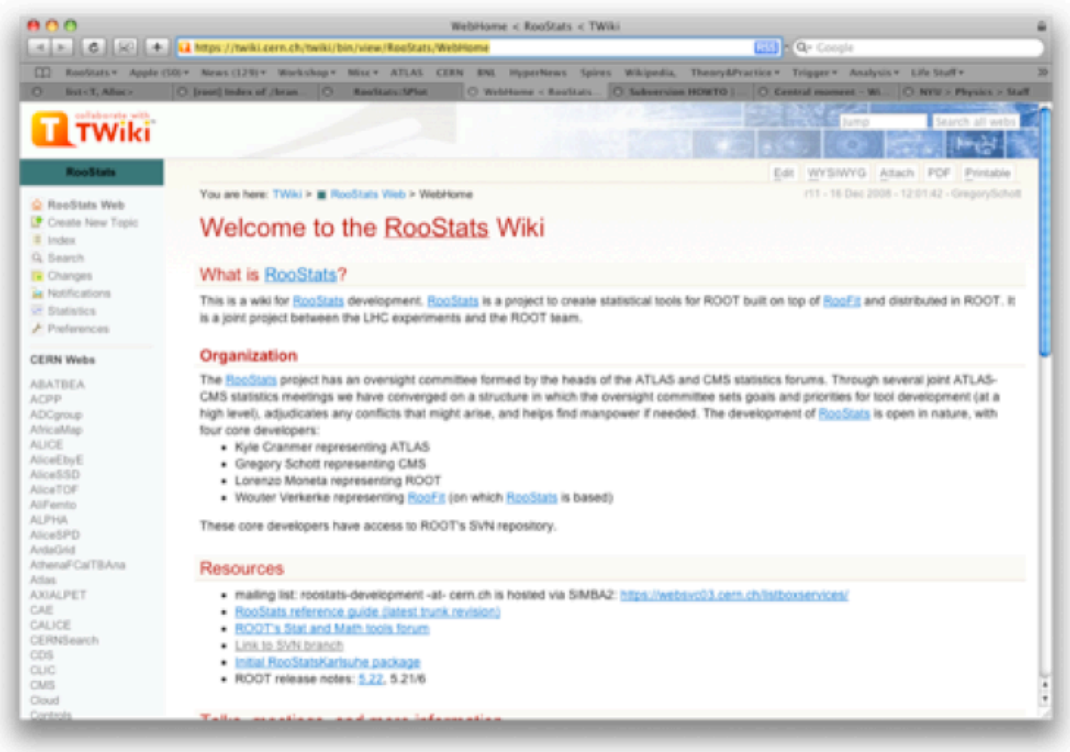
This afternoon – focus on software
(that can do all three techniques)

RooStats Project – Overview

- Goals:
  - Standardize interface for major statistical procedures so that they can work on an arbitrary RooFit model & dataset and handle many parameters of interest and nuisance parameters.
  - Implement most accepted techniques from Frequentist, Bayesian, and Likelihood-based approaches
  - Provide utilities to perform combined measurements

- Design:
  - Essentially all methods start with the basic probability density function or likelihood function. *Building a good model is the hard part*. Want to re-use it for multiple methods → Use RooFit to construct models
  - Build series of tools that perform statistical procedures on RooFit models

The RooStats project



https://twiki.cern.ch/twiki/bin/view/RooStats/WebHome

Release notes:
http://root.cern.ch/root/v524/Version524.news.html#roofit

Code documentation:
http://root.cern.ch/root/html/ROOFIT_ROOSTATS_Index.html

There is also a category in ROOT's Savannah bug tracking system

Core developers

- K. Cranmer (ATLAS)
- Gregory Schott (CMS)
- Wouter Verkerke (RooFit)
- Lorenzo Moneta (ROOT)
- open project, you are welcome to join. Contributions from
  - Max Baak, Kevin Belasco, Danilo Piparo, Giacinto Piacquadio, Maurizio Pierini, George H. Lewis, Alfio Lazzaro, Mario Pelliccioni, Matthias Wolf
  - Included since ROOT v5.22
- Example macros in
  - $ROOTSYS/tutorials/roostats

Documentation

- code doc. via ROOT
- users manual currently 32 pages, linked from Wiki page

RooStats Project – Structure

- **RooFit** (data modeling) - today
  - Data modeling language (pdfs and likelihoods). Scales to arbitrary complexity
  - Support for efficient integration, toy MC generation
  - Workspace
    - Persistent container for data models
    - Completely self-contained (including custom code)
    - Complete introspection and access to components
  - Workspace factory provides easy scripting language to populate the workspace

- **RooStats** (limits, interval calculators & utilities) - tomorrow
  - (Profile) Likelihood calculator
  - Neyman construction (FC)
  - Bayesian calculator (BAT & native MCMC)
  - Utilities (combinations, construct pdfs corresponding to standard number counting problems)

RooStats Project – Example

- Create a model - Example

$$Poisson(x \mid s \cdot r_s + b \cdot r_b) \cdot Gauss(r_s,1,0.05) \cdot Gauss(r_b,1,0.1)$$

Create workspace with above model (using factory)

```
RooWorkspace* w = new RooWorkspace("w");
w->factory("Poisson::P(obs[150,0,300],
                       sum::n(s[50,0,120]*ratioSigEff[1.,0,2.],
                              b[100,0,300]*ratioBkgEff[1.,0.,2.]))");
w->factory("PROD::PC(P, Gaussian::sigCon(ratioSigEff,1,0.05),
                        Gaussian::bkgCon(ratioBkgEff,1,0.1))");
```

Contents of workspace from above operation

```
RooWorkspace(w) w contents

variables
---------
(b,obs,ratioBkgEff,ratioSigEff,s)

p.d.f.s
-------
RooProdPdf::PC[ P * sigCon * bkgCon ] = 0.0325554
  RooPoisson::P[ x=obs mean=countingModel_2 ] = 0.0325554
    RooAddition::n[ s * ratioSigEff + b * ratioBkgEff ] = 150
  RooGaussian::sigCon[ x=ratioSigEff mean=1 sigma=0.05 ] = 1
  RooGaussian::bkgCon[ x=ratioBkgEff mean=1 sigma=0.1 ] = 1
```

RooStats Project – Example

- Confidence intervals calculated with model
  - Profile likelihood

```
ProfileLikelihoodCalculator plc;
plc.SetPdf(w::PC);
plc.SetData(data); // contains [obs=160]
plc.SetParameters(w::s);
plc.SetTestSize(.1);
ConfInterval* lrint = plc.GetInterval(); // that was easy.
```

  - Feldman Cousins

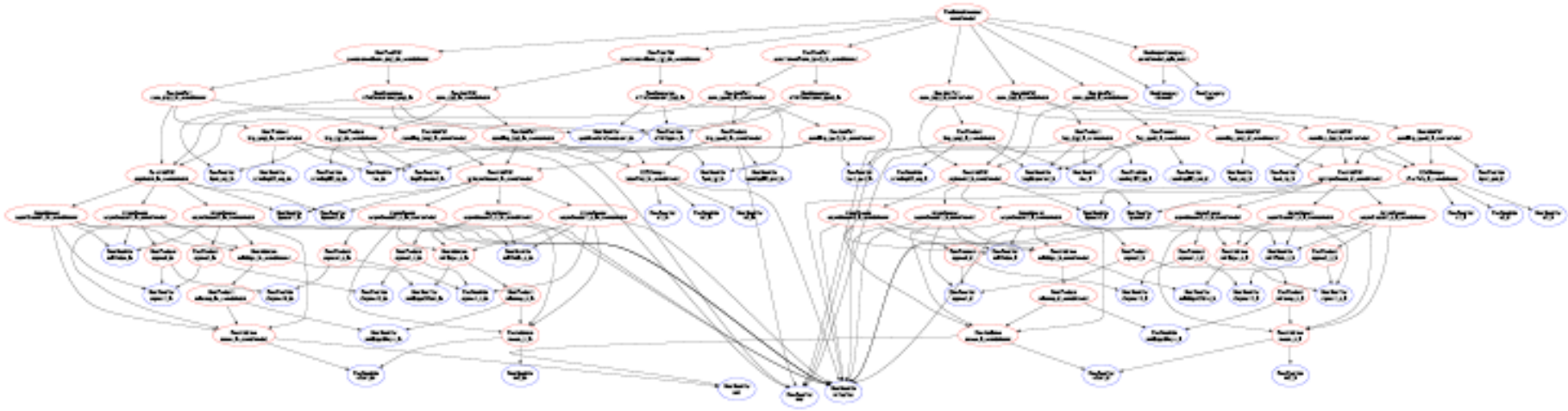
```
FeldmanCousins fc;
fc.SetPdf(w::PC);
fc.SetData(data); fc.SetParameters(w::s);
fc.UseAdaptiveSampling(true);
fc.FluctuateNumDataEntries(false);
fc.SetNBins(100); // number of points to test per parameter
fc.SetTestSize(.1);
ConfInterval* fcint = fc.GetInterval(); // that was easy.
```

  - Bayesian (MCMC)

```
UniformProposal up;
MCMCCalculator mc;
mc.SetPdf(w::PC);
mc.SetData(data);  mc.SetParameters(s);
mc.SetProposalFunction(up);
mc.SetNumIters(100000); // steps in the chain
mc.SetTestSize(.1); // 90% CL
mc.SetNumBins(50); // used in posterior histogram
mc.SetNumBurnInSteps(40);
ConfInterval* mcmcint = mc.GetInterval();
```

RooStats Project – Example

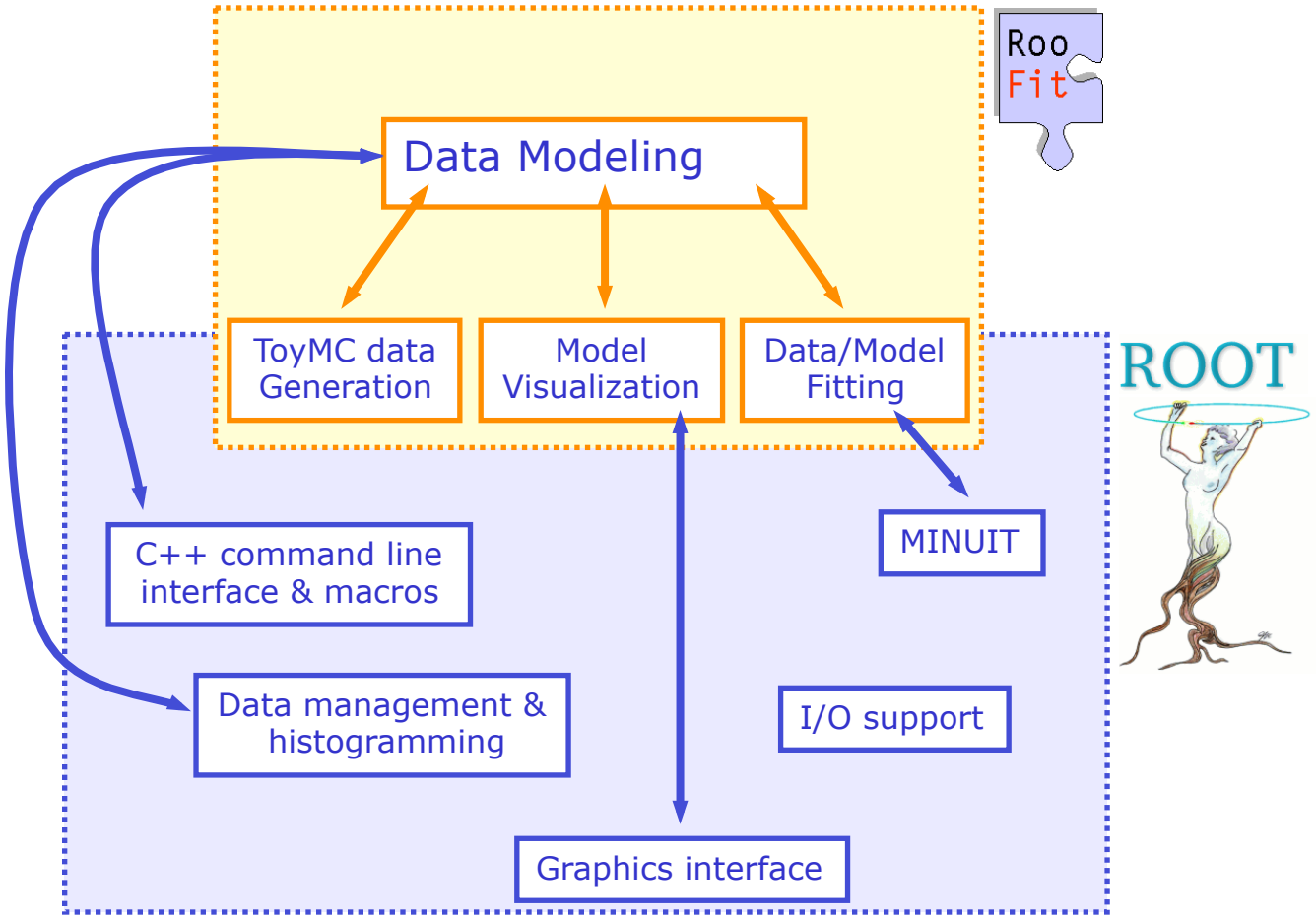
- **Interval calculators make no assumptions on internal structure of model**. Can feed model of arbitrary complexity to same calculator (computational limitations still apply!)



RooFit basics, the workspace

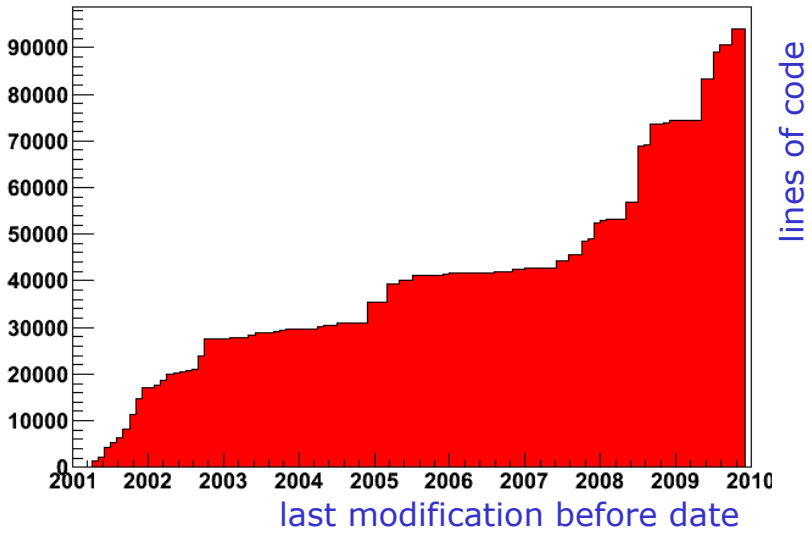
RooFit - Introduction – Relation to ROOT

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



Project timeline

- 1999 : Project started
  - First application: 'sin2b' measurement of BaBar (model with 5 observables, 37 floating parameters, simultaneous fit to multiple CP and control channels)
- 2000 : Complete overhaul of design based on experience with sin2b fit
  - Very useful exercise: new design is still current design
- 2003 : Public release of RooFit with ROOT
- 2004 : Over 50 BaBar physics publications using RooFit
- 2007 : Integration of RooFit in ROOT CVS source
- 2008 : Upgrade in functionality as part of RooStats project
  - Improved analytical and numeric integration handling, improved toy MC generation, addition of workspace
- 2009 : Now ~100K lines of code
  - (For comparison RooStats proper is ~10000 lines of code)



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`

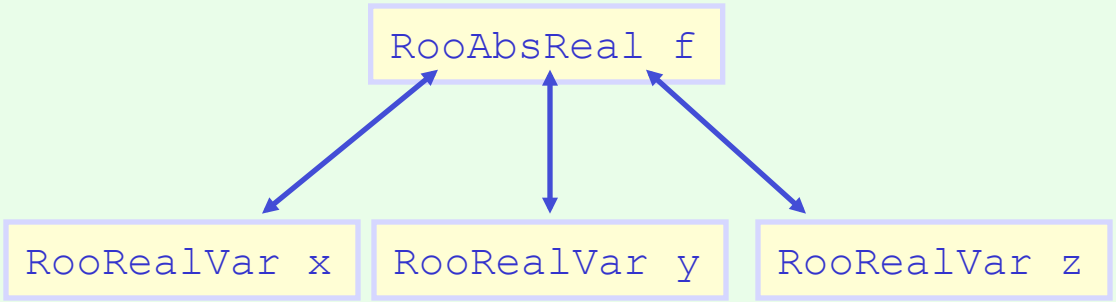
RooFit core design philosophy

- Represent relations between variables and functions as client/server links between objects

Math

$$f(x,y,z)$$

RooFit diagram



RooFit code

```
RooRealVar x("x","x",5) ;
RooRealVar y("y","y",5) ;
RooRealVar z("z","z",5) ;
RooBogusFunction f("f","f",x,y,z) ;
```

Object-oriented data modeling

- All objects are **self documenting**
  - **Name** - Unique identifier of object
  - **Title** – More elaborate description of object

Initial range

Objects representing a 'real' value.

```
RooRealVar mass("mass","Invariant mass",5.20,5.30) ;
RooRealVar width("width","B0 mass width",0.00027,"GeV");
RooRealVar mb0("mb0","B0 mass",5.2794,"GeV") ;
```

Initial value Optional unit

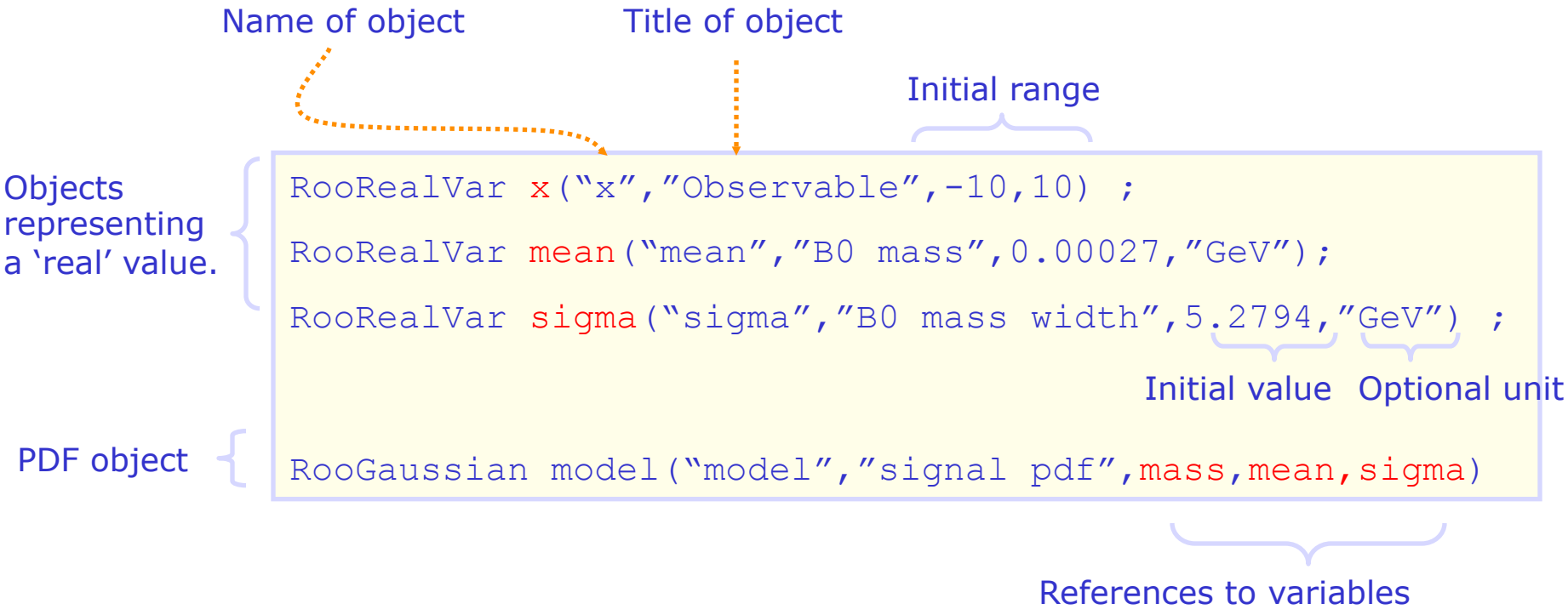
PDF object

```
RooGaussian b0sig("b0sig","B0 sig PDF",mass,mb0,width);
```

References to variables

The simplest possible example

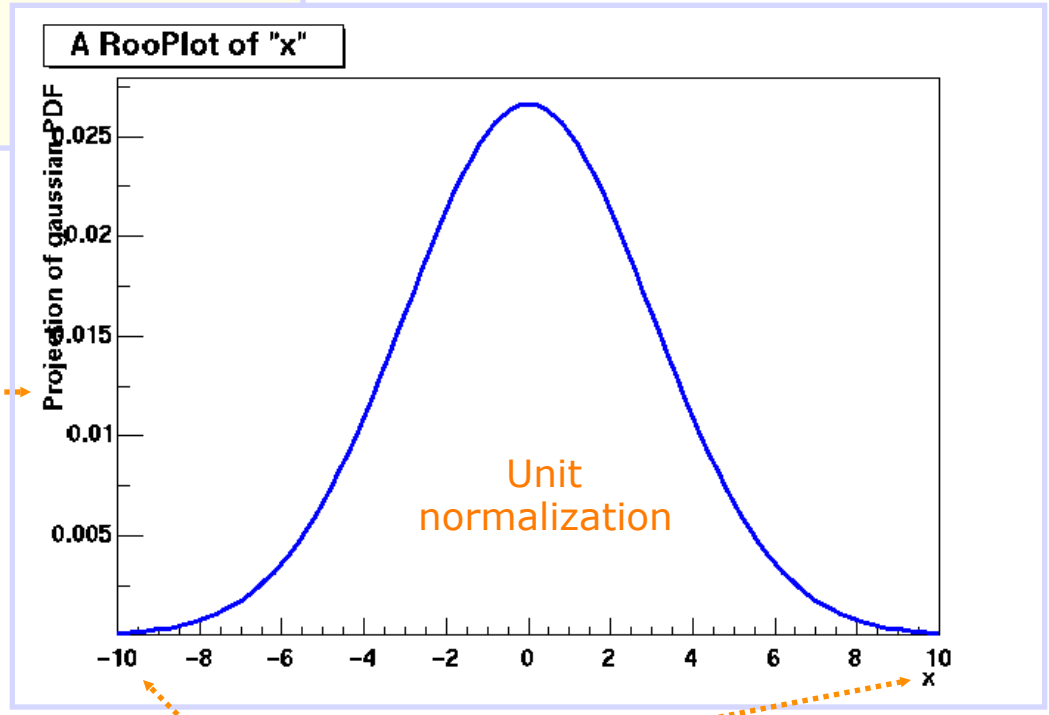
- We make a Gaussian p.d.f. with three variables: mass, mean and sigma



Basics – Creating and plotting a Gaussian p.d.f

Setup gaussian PDF and plot

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

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

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



Axis label from `gauss` title

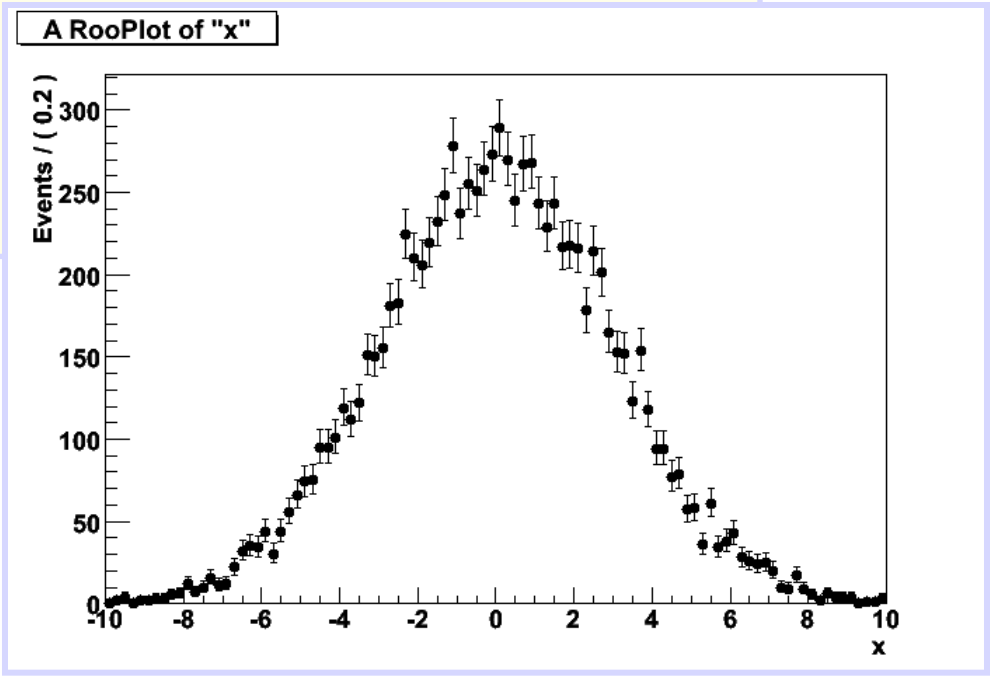
A `RooPlot` is an empty frame capable of holding anything plotted versus it variable

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 = w::gauss.generate(w::x,10000) ;

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

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

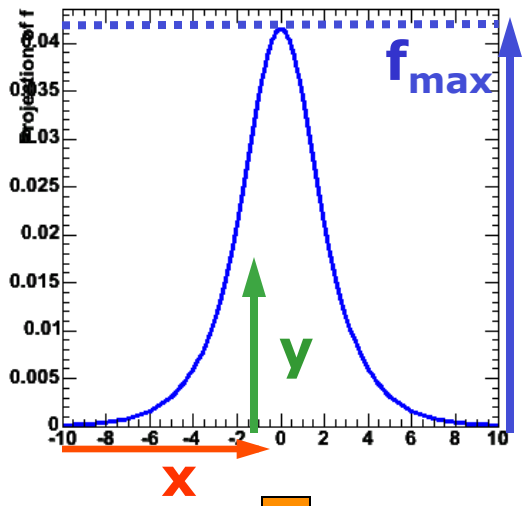


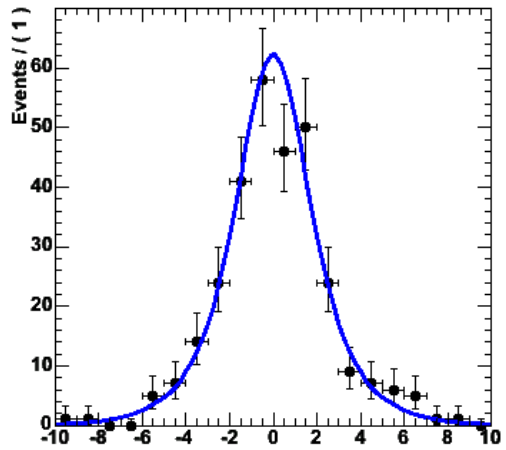
Can generate both binned and unbinned datasets

Toy MC generation – Accept/reject sampling

- *How to sample events directly from your fit function?*
- Simplest: accept/reject sampling

1) Determine maximum of function f_{max}
2) Throw random number **x**
3) Throw another random number **y**
4) **If y<f(x)/f_{max} keep x, otherwise return to step 2)**



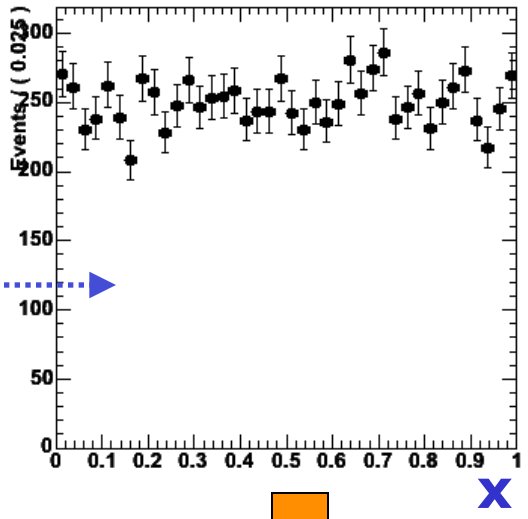


- PRO: Easy, always works
- CON: It can be inefficient if function is strongly peaked.
  Finding maximum empirically through random sampling can be lengthy in >2 dimensions

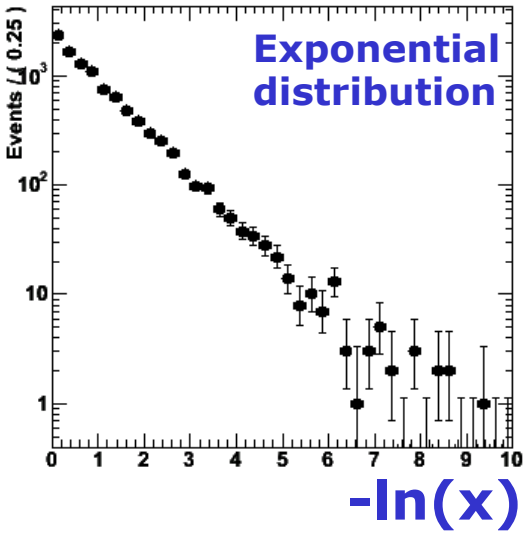
Toy MC generation – Inversion method

- Fastest: function inversion

1) Given f(x) find inverted function F(x) so that f(F(x)) = x
2) Throw uniform random number x
3) Return F(x)



Take –log(x)

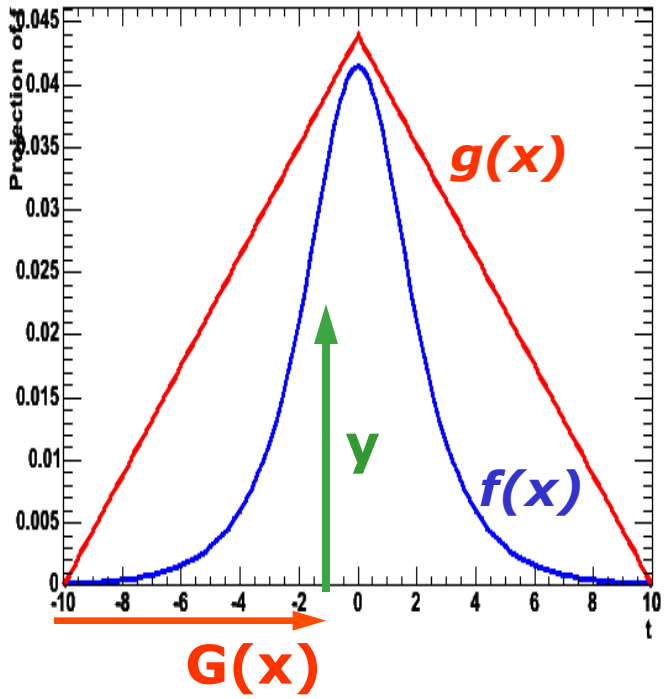


- PRO: Maximally efficient
- CON: Only works for invertible functions

Toy MC Generation in a nutshell

- Hybrid: Importance sampling

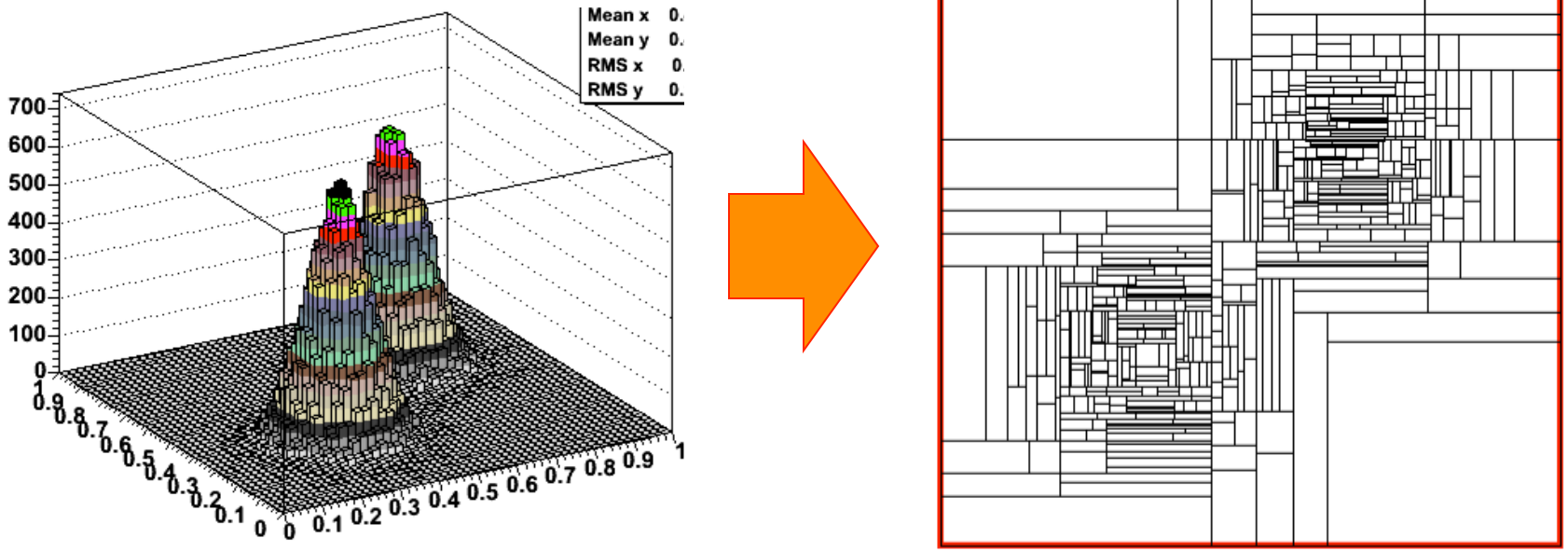
1) Find 'envelope function' g(x) that is invertible into G(x) and that fulfills g(x)>=f(x) for all x
2) Generate random number x from G using inversion method
3) Throw random number 'y'
4) If y<f(x)/g(x) keep x, otherwise return to step 2



- PRO: Faster than plain accept/reject sampling
  Function does not need to be invertible
- CON: Must be able to find invertible envelope function

Toy MC Generation in a nutshell

- General algorithms exists that can construct empirical envelope function
  - Divide observable space recursively into smaller boxes and take uniform distribution in each box
  - Example shown below from FOAM algorithm



Basics – Importing data

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

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

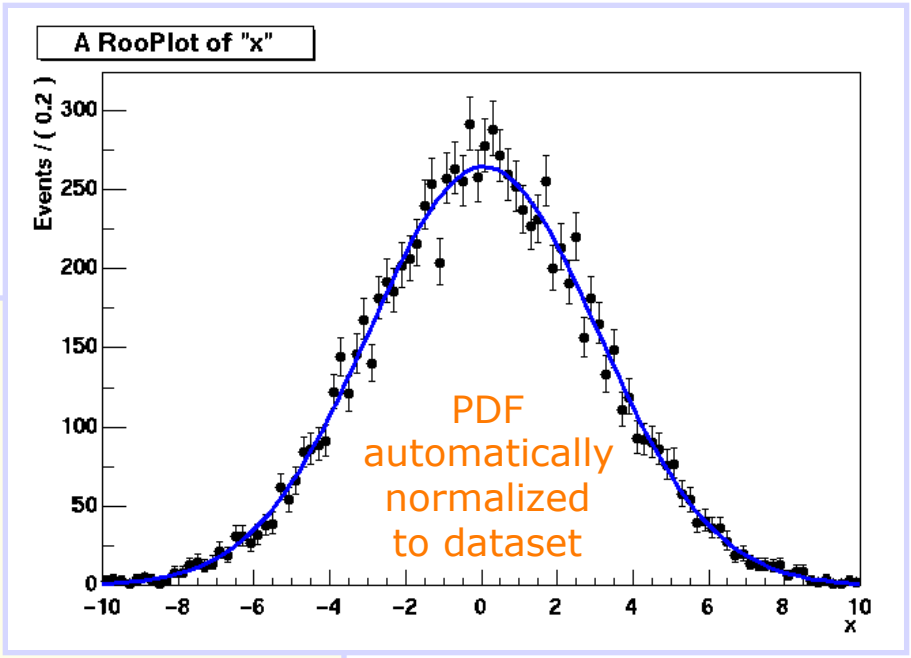
  - Imports **TTree** branch named "x".
  - Can be of type **Double_t**, **Float_t**, **Int_t** or **UInt_t**.
    All data is converted to Double_t internally
  - Specify a **RooArgSet** of multiple observables to import multiple observables

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

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

  - Imports values, binning definition *and* SumW2 errors (if defined)
  - Specify a **RooArgList** of observables when importing a TH2/3.

Basics – ML fit of p.d.f to *unbinned* data




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

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

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

Basics – ML fit of p.d.f to *unbinned* data

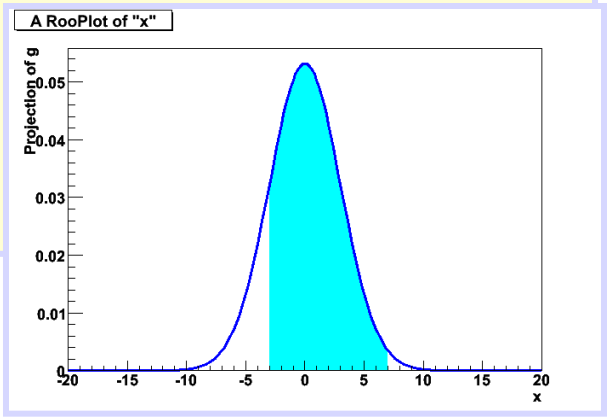
- Can also choose to save full detail of fit

```
RooFitResult* r = w::gauss.fitTo(*data,Save()) ;

r->Print() ;
  RooFitResult: minimized FCN value: 25055.6,
                estimated distance to minimum: 7.27598e-08
                coviarance matrix quality:
                Full, accurate covariance matrix

    Floating Parameter    FinalValue +/-  Error
  --------------------  --------------------------
                  mean    1.7233e-02 +/-  3.00e-02
                 sigma    2.9809e+00 +/-  2.17e-02


r->correlationMatrix().Print() ;

2x2 matrix is as follows

     |      0    |      1    |
-------------------------------
   0 |          1   0.0005869
   1 |  0.0005869           1
```

Basics – Integrals over p.d.f.s

- It is easy to create an object *representing integral* over a normalized p.d.f in a sub-range

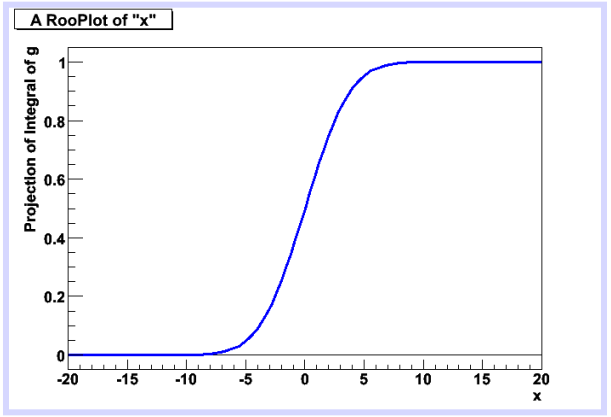
```
w::x.setRange("sig",-3,7) ;
RooAbsReal* ig = w::g.createIntegral(x,NormSet(x),Range("sig")) ;
cout << ig.getVal() ;
0.832519
mean=-1 ;
cout << ig.getVal() ;
0.743677
```



- Similarly, one can also request the *cumulative distribution function*

$$C(x) = \int_{x_{\min}}^{x} F(x')dx'$$

```
RooAbsReal* cdf = gauss.createCdf(x) ;
```



The workspace

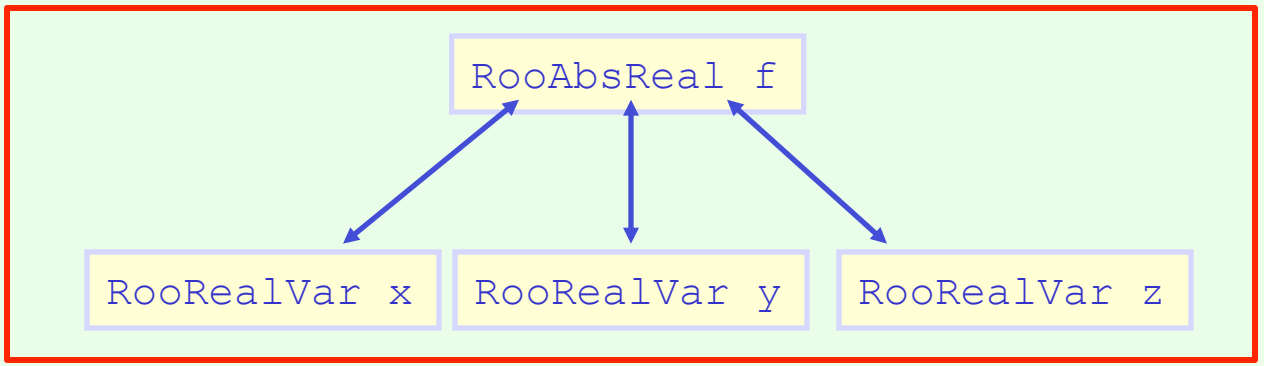
RooFit core design philosophy - Workspace

- The workspace serves a container class for all objects created

Math

$$f(x,y,z)$$

RooFit diagram



RooFit code

```
RooRealVar x("x","x",5) ;
RooRealVar y("y","y",5) ;
RooRealVar z("z","z",5) ;
RooBogusFunction f("f","f",x,y,z) ;
RooWorkspace w("w") ;
w.import(f) ;
```

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 function into a workspace
  - When importing a function or pdf, 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(myFunction) ;
```

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

- Alternative access to contents through namespace
  - Uses CINT extension of C++, works in interpreted code only

```
// Variety of accessors available
w.exportToCint() ;
RooPlot* frame = w::x.frame() ;
w::f.plotOn(frame) ;
```

- Writing workspace and contents to file

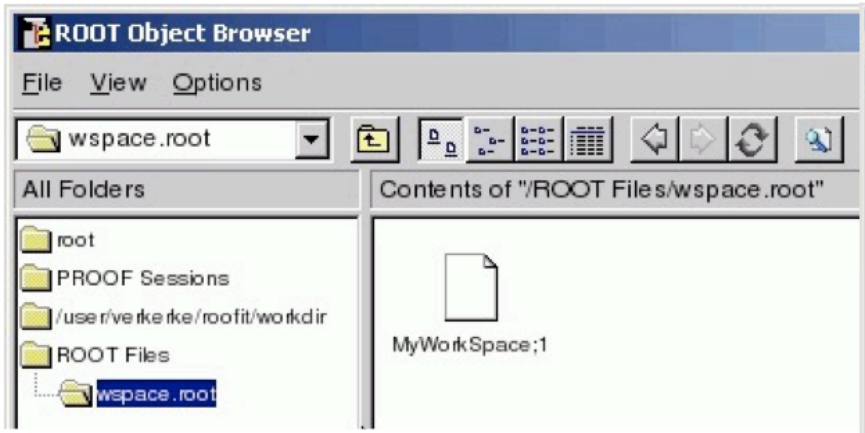
```
w.writeToFile("wspace.root") ;
```

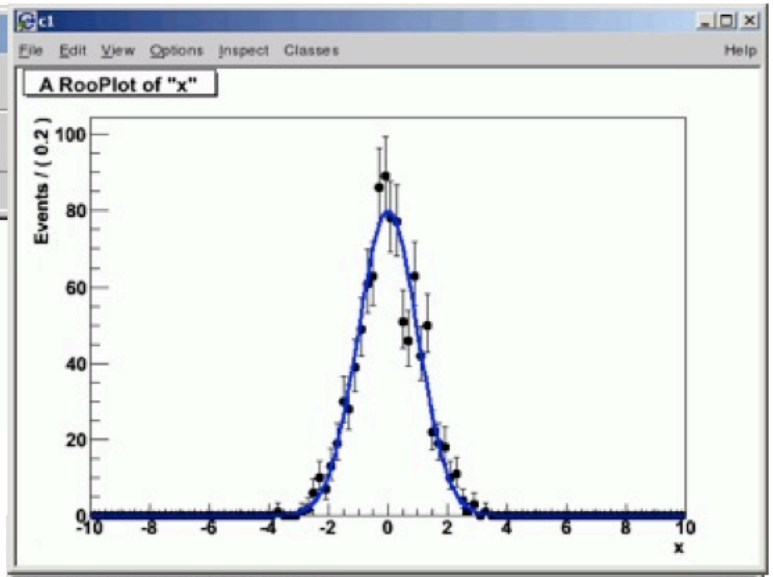
Using the workspace

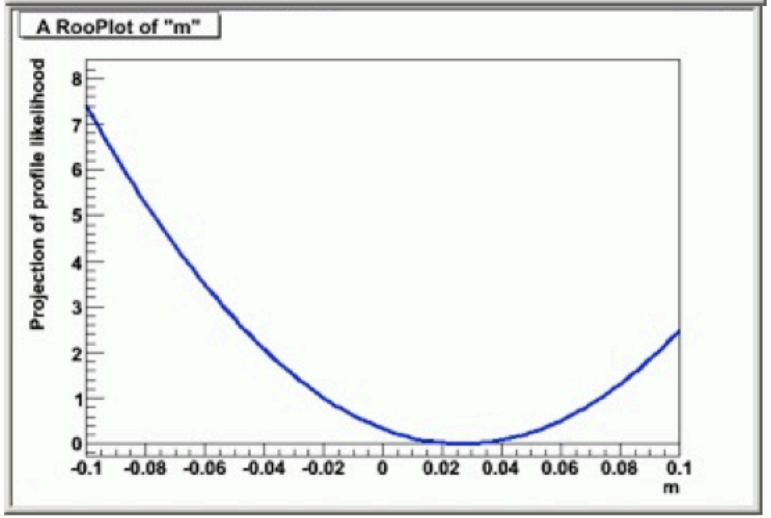
- Organizing your code –
  Separate construction and use of models

```
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
  // CAN BE IN A DIFFERENT ROOT SESSION
  // (if workspace is written to file)
}
```

Simple example of workspace in file







RooFit's Workspace now provides the ability to save in a ROOT file the full likelihood model, any priors you might want, and the minimal data necessary to reproduce likelihood function.

Can also evaluate integrals over x necessary for Neyman construction!

Need this for combinations, great potential for publishing results.

Full Higgs combination in workspace

```
Unix> root -l higgs_125.root
Root> combWS->Print("t")
```

```
RooFit v3.53 -- Developed by Wouter Verkerke and David Kirkby
                Copyright (C) 2000-2012 NIKHEF, University of California & Stanford University
                All rights reserved, please read http://roofit.sourceforge.net/license.txt


Attaching file atlas_higgs_125.root as _file0...


RooWorkspace(combWS) combWS contents

…
p.d.f.s

-------

RooSimultaneous::combPdf[ indexCat=master_channel ATLAS_H_2e2mu_channel_125O_1=model_ATLAS_H_2e2mu_channel_125O_1_h1_edit ATLAS_H_2mu2e_channel_125O_1=model_ATLAS_H_2mu2e_channel_125O_1_h1_edit
ATLAS_H_4e_channel_125O_1=model_ATLAS_H_4e_channel_125O_1_h1_edit ATLAS_H_4mu_channel_125O_1=model_ATLAS_H_4mu_channel_125O_1_h1_edit HHChannel=model_HHChannel_simPdf_edit_hh_edit WH=model_WH_wh_edit ZH=model_ZH_zh_edit
chan0j=model_chan0j_simPdf_edit_ll_edit chan1j=model_chan1j_simPdf_edit_ll_edit chan2jVBF=model_chan2jVBF_simPdf_edit_ll_edit chan2jVH=model_chan2jVH_simPdf_edit_ll_edit channelAllVBFHMET=model_channelAllVBFHMET_simPdf_edit_lh_edit
channelEle0jetHMET=model_channelEle0jetHMET_simPdf_edit_lh_edit channelEle0jetLMET=model_channelEle0jetLMET_simPdf_edit_lh_edit channelEle1jetinHMET=model_channelEle1jetinHMET_simPdf_edit_lh_edit
channelMuo0jetHMET=model_channelMuo0jetHMET_simPdf_edit_lh_edit channelMuo0jetLMET=model_channelMuo0jetLMET_simPdf_edit_lh_edit channelMuo1jetinHMET=model_channelMuo1jetinHMET_simPdf_edit_lh_edit conv_bad=conv_bad_new_gg_edit
conv_good_highpt=model_conv_good_highpt_gg_edit conv_good_lowpt=model_conv_good_lowpt_gg_edit conv_medium_highpt=conv_medium_highpt_new_gg_edit conv_medium_lowpt=conv_medium_lowpt_new_gg_edit ee_mainControlBK_0j=model_ee_mainControlBK_0j_ww2l_edit
ee_mainControlBK_1j=model_ee_mainControlBK_1j_ww2l_edit ee_mainControlLM_0j=model_ee_mainControlLM_0j_ww2l_edit ee_mainControlLM_1j=model_ee_mainControlLM_1j_ww2l_edit ee_signalLikeBK_0j=model_ee_signalLikeBK_0j_ww2l_edit
ee_signalLikeBK_1j=model_ee_signalLikeBK_1j_ww2l_edit ee_signalLikeBK_2j=model_ee_signalLikeBK_2j_ww2l_edit ee_signalLikeLM_0j=model_ee_signalLikeLM_0j_ww2l_edit ee_signalLikeLM_1j=model_ee_signalLikeLM_1j_ww2l_edit
ee_signalLikeLM_2j=model_ee_signalLikeLM_2j_ww2l_edit ee_topboxBK_1j=model_ee_topboxBK_1j_ww2l_edit ee_topboxBK_2j=model_ee_topboxBK_2j_ww2l_edit ee_topboxLM_1j=model_ee_topboxLM_1j_ww2l_edit ee_topboxLM_2j=model_ee_topboxLM_2j_ww2l_edit
em_mainControlBK_0j=model_em_mainControlBK_0j_ww2l_edit em_mainControlBK_1j=model_em_mainControlBK_1j_ww2l_edit em_mainControlLM_0j=model_em_mainControlLM_0j_ww2l_edit em_mainControlLM_1j=model_em_mainControlLM_1j_ww2l_edit
em_signalLikeBK_0j=model_em_signalLikeBK_0j_ww2l_edit em_signalLikeBK_1j=model_em_signalLikeBK_1j_ww2l_edit em_signalLikeBK_2j=model_em_signalLikeBK_2j_ww2l_edit em_signalLikeLM_0j=model_em_signalLikeLM_0j_ww2l_edit
em_signalLikeLM_1j=model_em_signalLikeLM_1j_ww2l_edit em_signalLikeLM_2j=model_em_signalLikeLM_2j_ww2l_edit em_topboxBK_1j=model_em_topboxBK_1j_ww2l_edit em_topboxBK_2j=model_em_topboxBK_2j_ww2l_edit em_topboxLM_1j=model_em_topboxLM_1j_ww2l_edit
em_topboxLM_2j=model_em_topboxLM_2j_ww2l_edit mm_mainControlBK_0j=model_mm_mainControlBK_0j_ww2l_edit mm_mainControlBK_1j=model_mm_mainControlBK_1j_ww2l_edit mm_mainControlLM_0j=model_mm_mainControlLM_0j_ww2l_edit
mm_mainControlLM_1j=model_mm_mainControlLM_1j_ww2l_edit mm_signalLikeBK_0j=model_mm_signalLikeBK_0j_ww2l_edit mm_signalLikeBK_1j=model_mm_signalLikeBK_1j_ww2l_edit mm_signalLikeBK_2j=model_mm_signalLikeBK_2j_ww2l_edit
mm_signalLikeLM_0j=model_mm_signalLikeLM_0j_ww2l_edit mm_signalLikeLM_1j=model_mm_signalLikeLM_1j_ww2l_edit mm_signalLikeLM_2j=model_mm_signalLikeLM_2j_ww2l_edit mm_topboxBK_1j=model_mm_topboxBK_1j_ww2l_edit
mm_topboxBK_2j=model_mm_topboxBK_2j_ww2l_edit mm_topboxLM_1j=model_mm_topboxLM_1j_ww2l_edit mm_topboxLM_2j=model_mm_topboxLM_2j_ww2l_edit unconv_good_highpt=unconv_good_highpt_new_gg_edit unconv_good_lowpt=unconv_good_lowpt_new_gg_edit
unconv_rest_highpt=unconv_rest_highpt_new_gg_edit unconv_rest_lowpt=unconv_rest_lowpt_new_gg_edit ] = 4.24265e-06

  RooProdPdf::model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ lumi_Pdf * alpha_ATLAS_EFF_SigConstraint_h1 * alpha_ATLAS_E_CUTS_EFFConstraint_h1 * ATLAS_E_EFF_Pdf * ATLAS_M_EFF_Pdf * alpha_BRConstraint_h1 * QCDscale_ggH_Pdf * pdf_gg_Pdf *
QCDscale_qqH_Pdf * pdf_qqbar_Pdf * QCDscale_VH_Pdf * alpha_ATLAS_QCDshape_VVConstraint_h1 * QCDscale_VV_Pdf * alpha_ATLAS_Norm_ZbbConstraint_h1 * alpha_ATLAS_Shape_ZbbConstraint_h1 * alpha_ATLAS_Norm_ZConstraint_h1 *
alpha_ATLAS_Shape_ZConstraint_h1 * ATLAS_H_2e2mu_channel_125O_1_model_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit ] = 7.9227e-05

    RooRealSumPdf::ATLAS_H_2e2mu_channel_125O_1_model_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ binWidth_obs_ATLAS_H_2e2mu_channel_125O_1_0_h1 *
L_x_ATLAS_Signal_gg_H_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit + binWidth_obs_ATLAS_H_2e2mu_channel_125O_1_1_h1 *
L_x_ATLAS_Signal_VBF_H_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit + binWidth_obs_ATLAS_H_2e2mu_channel_125O_1_2_h1 *
L_x_ATLAS_Signal_qq_WH_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit + binWidth_obs_ATLAS_H_2e2mu_channel_125O_1_3_h1 *
L_x_ATLAS_Signal_qq_ZH_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit + binWidth_obs_ATLAS_H_2e2mu_channel_125O_1_4_h1 *
L_x_ATLAS_Bkg_qq_ZZ_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_HistSyst_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit + binWidth_obs_ATLAS_H_2e2mu_channel_125O_1_5_h1 *
L_x_ATLAS_Bkg_Zbb_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_HistSyst_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit + binWidth_obs_ATLAS_H_2e2mu_channel_125O_1_6_h1 *
L_x_ATLAS_Bkg_Z_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_HistSyst_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit + binWidth_obs_ATLAS_H_2e2mu_channel_125O_1_7_h1 *
L_x_ATLAS_Bkg_tt_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit ] = 7.9227e-05

      RooProduct::L_x_ATLAS_Signal_gg_H_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ lumi_fix39 *
ATLAS_Signal_gg_H_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit ] = 0

        RooProduct::ATLAS_Signal_gg_H_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ ATLAS_Signal_gg_H_ATLAS_H_2e2mu_channel_125O_1_nominal_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit *
ATLAS_Signal_gg_H_ATLAS_H_2e2mu_channel_125O_1_overallNorm_x_sigma_epsilon_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit ] = 0

          RooHistFunc::ATLAS_Signal_gg_H_ATLAS_H_2e2mu_channel_125O_1_nominal_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ depList=(obs_ATLAS_H_2e2mu_channel_125O_1) ] = 0

          RooProduct::ATLAS_Signal_gg_H_ATLAS_H_2e2mu_channel_125O_1_overallNorm_x_sigma_epsilon_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ mu *
ATLAS_Signal_gg_H_ATLAS_H_2e2mu_channel_125O_1_epsilon_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit ] = 0

            RooStats::HistFactory::FlexibleInterpVar::ATLAS_Signal_gg_H_ATLAS_H_2e2mu_channel_125O_1_epsilon_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ paramList=
(alpha_ATLAS_EFF_Sig_h1,alpha_ATLAS_E_CUTS_EFF_h1,ATLAS_E_EFF,ATLAS_M_EFF,alpha_BR_h1,QCDscale_ggH,pdf_gg) ] = 1

          RooStats::HistFactory::FlexibleInterpVar::lumi_fix39[ paramList=(lumi) ] = 1

      RooProduct::L_x_ATLAS_Signal_VBF_H_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ lumi_fix39 *
ATLAS_Signal_VBF_H_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit ] = 0

        RooProduct::ATLAS_Signal_VBF_H_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ ATLAS_Signal_VBF_H_ATLAS_H_2e2mu_channel_125O_1_nominal_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit *
ATLAS_Signal_VBF_H_ATLAS_H_2e2mu_channel_125O_1_overallNorm_x_sigma_epsilon_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit ] = 0

          RooHistFunc::ATLAS_Signal_VBF_H_ATLAS_H_2e2mu_channel_125O_1_nominal_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ depList=(obs_ATLAS_H_2e2mu_channel_125O_1) ] = 0

          RooProduct::ATLAS_Signal_VBF_H_ATLAS_H_2e2mu_channel_125O_1_overallNorm_x_sigma_epsilon_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ mu *
ATLAS_Signal_VBF_H_ATLAS_H_2e2mu_channel_125O_1_epsilon_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit ] = 0

            RooStats::HistFactory::FlexibleInterpVar::ATLAS_Signal_VBF_H_ATLAS_H_2e2mu_channel_125O_1_epsilon_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ paramList=
(alpha_ATLAS_EFF_Sig_h1,alpha_ATLAS_E_CUTS_EFF_h1,ATLAS_E_EFF,ATLAS_M_EFF,alpha_BR_h1,QCDscale_qqH,pdf_qqbar) ] = 1

          RooStats::HistFactory::FlexibleInterpVar::lumi_fix39[ paramList=(lumi) ] = 1

      RooProduct::L_x_ATLAS_Signal_qq_WH_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ lumi_fix39 *
ATLAS_Signal_qq_WH_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit ] = 0

        RooProduct::ATLAS_Signal_qq_WH_ATLAS_H_2e2mu_channel_125O_1_overallSyst_x_Exp_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ ATLAS_Signal_qq_WH_ATLAS_H_2e2mu_channel_125O_1_nominal_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit *
ATLAS_Signal_qq_WH_ATLAS_H_2e2mu_channel_125O_1_overallNorm_x_sigma_epsilon_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit ] = 0

          RooHistFunc::ATLAS_Signal_qq_WH_ATLAS_H_2e2mu_channel_125O_1_nominal_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ depList=(obs_ATLAS_H_2e2mu_channel_125O_1) ] = 0

          RooProduct::ATLAS_Signal_qq_WH_ATLAS_H_2e2mu_channel_125O_1_overallNorm_x_sigma_epsilon_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit[ mu *
ATLAS_Signal_qq_WH_ATLAS_H_2e2mu_channel_125O_1_epsilon_h1_model_ATLAS_H_2e2mu_channel_125O_1_h1_edit ] = 0
```

(truncated – full printout ~9000 lines)

RooFit - the factory

RooFit core design philosophy - Factory

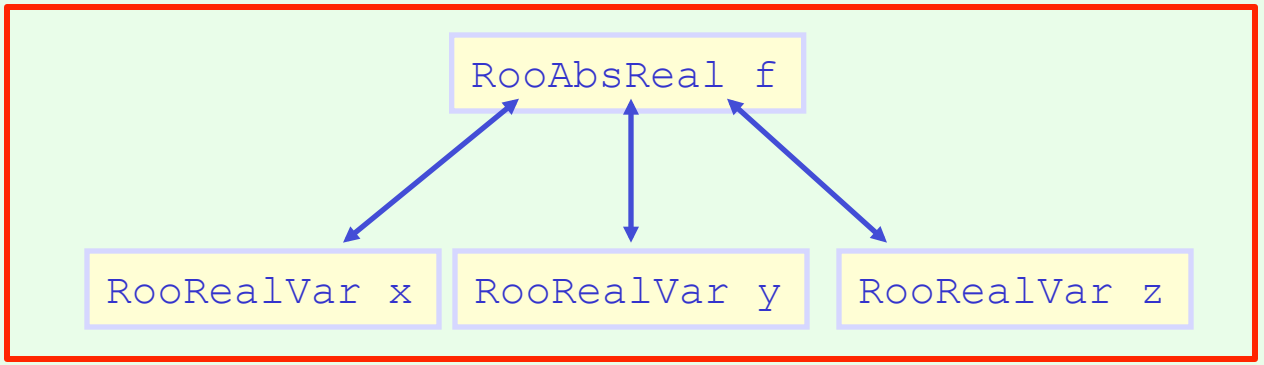
- The factory allows to fill a workspace with pdfs and variables using a simplified scripting language

Math

$$f(x,y,z)$$

RooFit diagram

RooWorkspace



RooFit code

```
RooWorkspace w("w") ;
w.factory("BogusFunction::f(x[5],y[5],z[5])") ;
```

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 language – Goal and scope

- Aim of factory language is to be very simple.
- The goal is to construct pdfs, functions and variables
  - This limits the scope of the factory language (and allows to keep it simple)
  - Objects can be customized after creation
- The language syntax *has only three elements*

1. Simplified expression for creation of variables
2. Expression for creation of functions and pdf is trivial 1-to-1 mapping of C++ constructor syntax of corresponding object
3. Multiple objects (e.g. a pdf and its variables) can be nested in a single expression

- Operator classes (sum,product) provide alternate syntax in factory that is closer to math notation

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)
        → RooGaussian("g","g",x,mean,sigma)

Polynomial::p(x,{a0,a1})
        → 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)
```

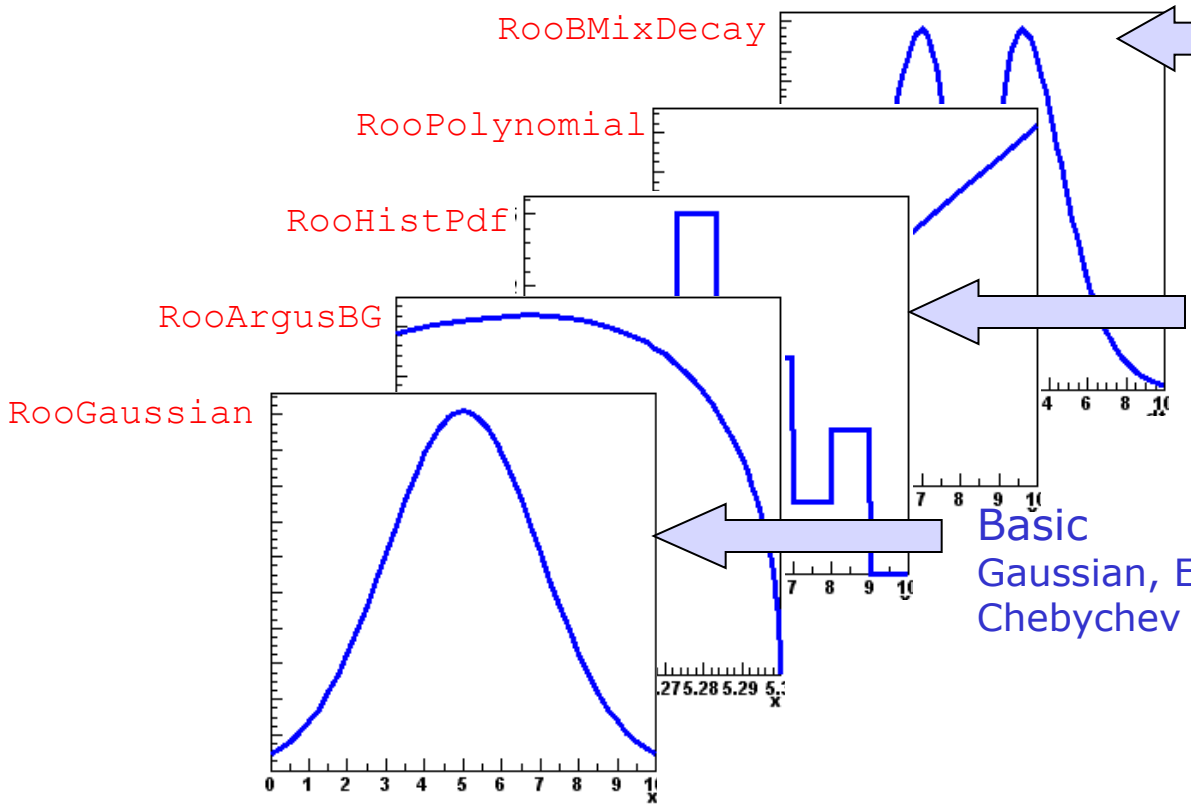
  - It is not required to give component objects a name, e.g.

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

Discovery models: signal models

Model building – (Re)using standard components

- RooFit provides a collection of compiled standard PDF classes



Physics inspired
ARGUS,Crystal Ball,
Breit-Wigner, Voigtian,
B/D-Decay,….

Non-parametric
Histogram, KEYS

Basic
Gaussian, Exponential, Polynomial,…
Chebychev polynomial

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

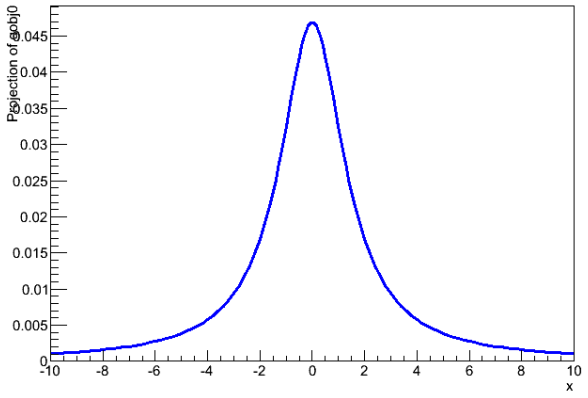
Modeling of (narrow) signal resonances

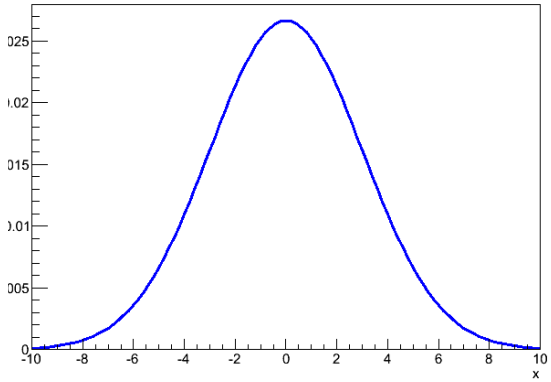
- Distributions for resonances
  - Resonances usually a Breit-Wigner form at the physics level
  - Calibration, reconstruction introduces Gaussian smearing
  - Best choice is Breit-Wigner (x) Gaussian = Voigtian
    - Analytically calculable (for non-relativistic BW)
  - But if smearing dominates with, can approximate with Gaussian
  - If resonance width dominates, can approximate with BreitWigner

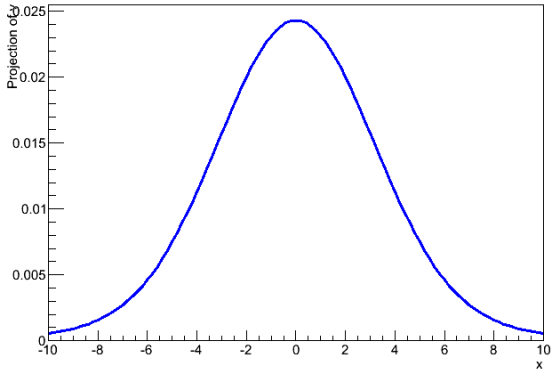
w.factory("**BreitWigner**::bw(x[-10,10],mean[0],gamma[3])")

w.factory("**Gaussian**::g(x[-10,10],mean[0],sigma[3])")

w.factory("**Voigtian**::v(x[-10,10],mean[0],sigma[2],gamma[3])")



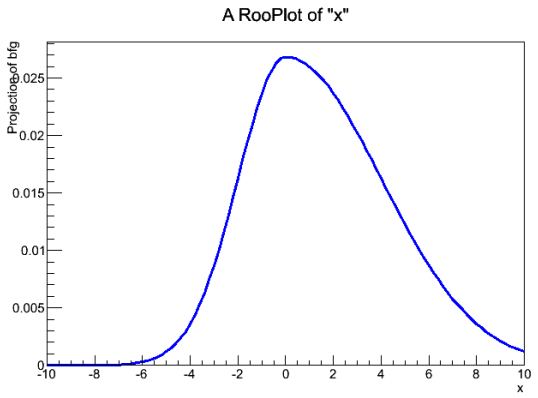




Other empirical signal models

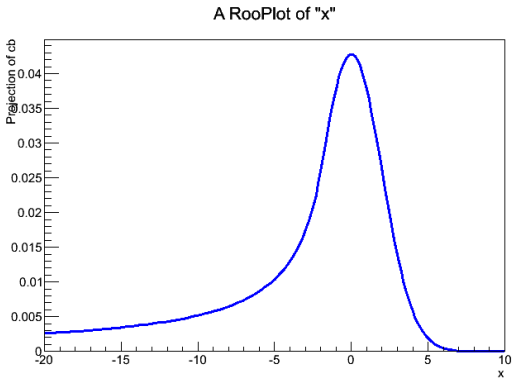
- Bifurcated Gaussian
  - Different sigma for left and right side

```
w.factory("BifurGauss::bfg(x[-10,10],
          mean[0],sigmaL[2],sigmaR[4])")
```



- Crystal Ball distribution
  - Gaussian distribution with an exponential tail 'glued' on at *n* signal with slope *alpha*

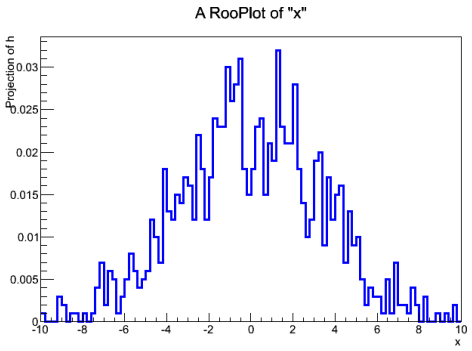
```
w.factory("CBShape::cb(x[-20,10],mean[0],
          sigma[2],alpha[1],n[1])")
```



Data-driven signal models

```
w.import(*histo,Rename("h")) ;
w.factory("HistPdf::hp(x,h)") ;
```

- Histogram-based pdf
  - Simply take shape from output of full simulation (no intrinsic parameters, but can add these a posteriori → tomorrow)



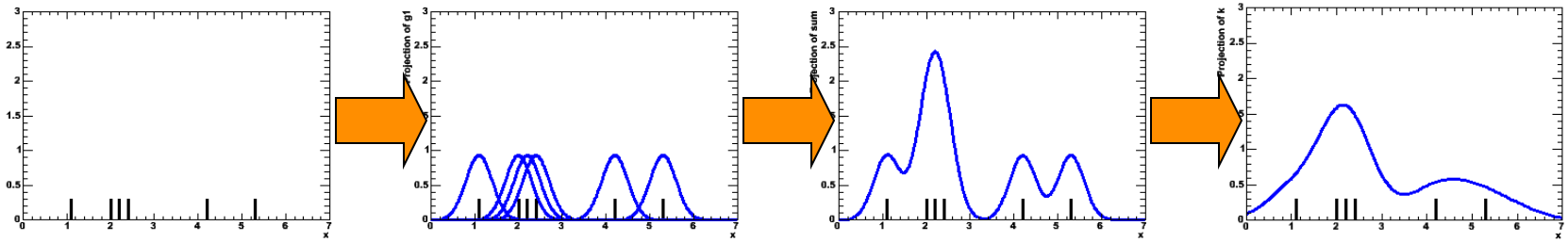
- Kernel estimation
  - Construct smooth pdf from unbinned data, using kernel estimation technique

Sample of events

Gaussian pdf for each event

Summed pdf for all events

Adaptive Kernel: width of Gaussian depends on local event density

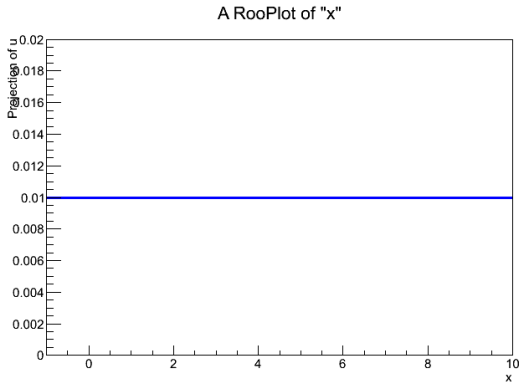


```
w.import(myData,Rename("myData")) ;
w.factory("KeysPdf::k(x,myData)")
```

Background models

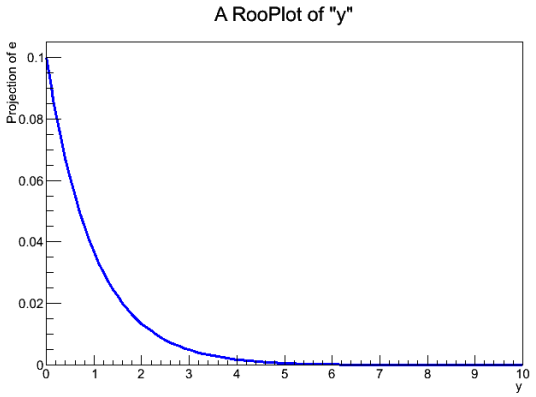
- Uniform
  - The most boring pdf…

    w.factory("**Uniform**::u(x[-1-,10])")



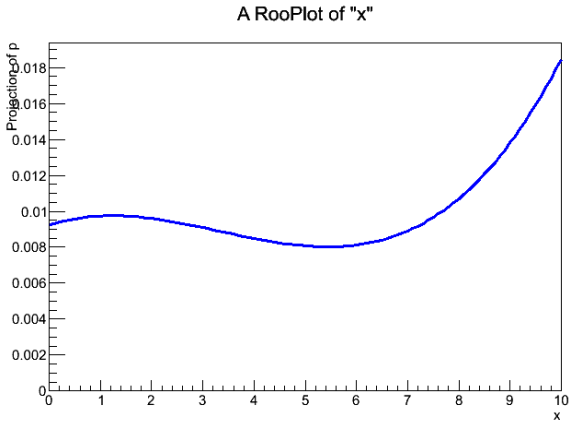
- Exponential
  - Parameter is slope, not lifetime

    w.factory("**Exponential**::e(y[0,10],alpha[-1])")



Background models

- Polynomial
  - Default form: $1+a_1+a_2+\ldots$ (because polynomial is explicitly normalized)
  - Unless interpretation of coefficients is meaningful better to use Chebychev polynomials

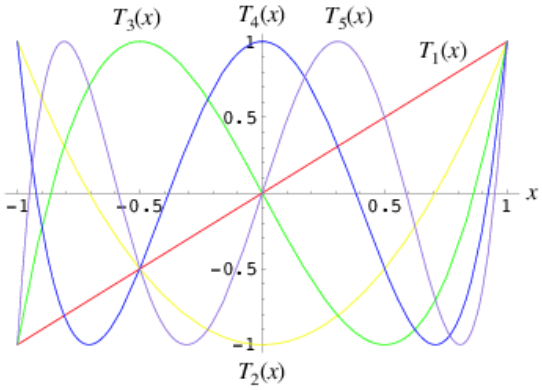


w.factory("**Polynomial**::p(x[0,10],{a1[0.1],a2[-0.01],a3[0.005]})")

- Chebychev polynomial
  - Expressed in orthogonal basis functions

w.factory("**Chebychev**::c(x[0,10], {a1[0.1],a2[-0.01],a3[0.005]})")

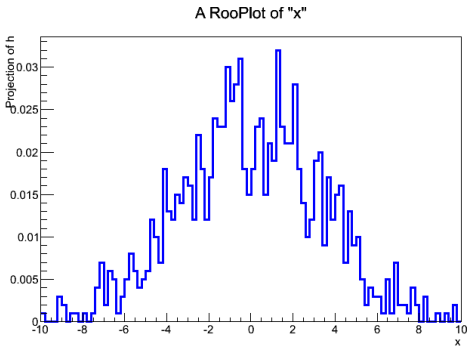
$$
\begin{aligned}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_2(x) &= 2x^2 - 1 \\
T_3(x) &= 4x^3 - 3x \\
T_4(x) &= 8x^4 - 8x^2 + 1 \\
T_5(x) &= 16x^5 - 20x^3 + 5x \\
T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1.
\end{aligned}
$$



Data-driven background models

```
w.import(*histo,Rename("h")) ;
w.factory("HistPdf::hp(x,h)") ;
```

- Histogram-based pdf
  - Simply take shape from output of full simulation (no intrinsic parameters, but can add these a posteriori → tomorrow)



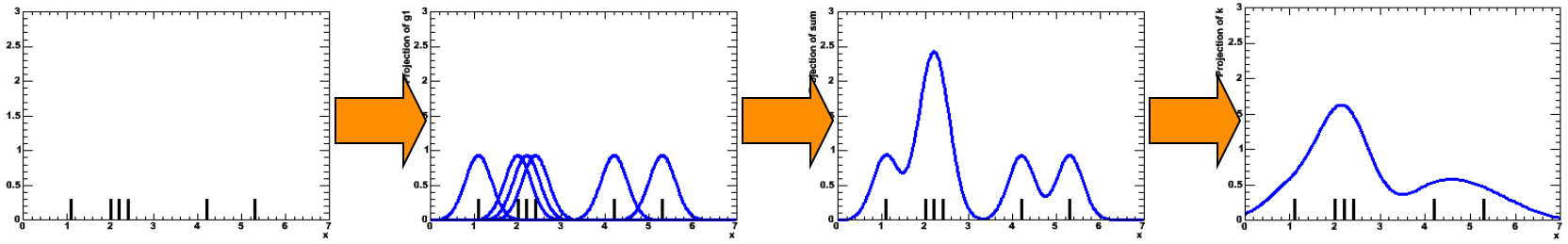
- Kernel estimation
  - Construct smooth pdf from unbinned data, using kernel estimation technique

Sample of events

Gaussian pdf for each event

Summed pdf for all events

Adaptive Kernel: width of Gaussian depends on local event density



```
w.import(myData,Rename("myData")) ;
w.factory("KeysPdf::k(x,myData)")
```

Model building – Making your own

- 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 in RooFit Manual)
- Compiled classes are faster in use, but require O(1-2) seconds startup overhead
  - Best choice depends on use context

Model building – Adjusting parameterization

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

```
w.factory("expr::alpha('-1/tau',tau[1,20])") ;
```

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

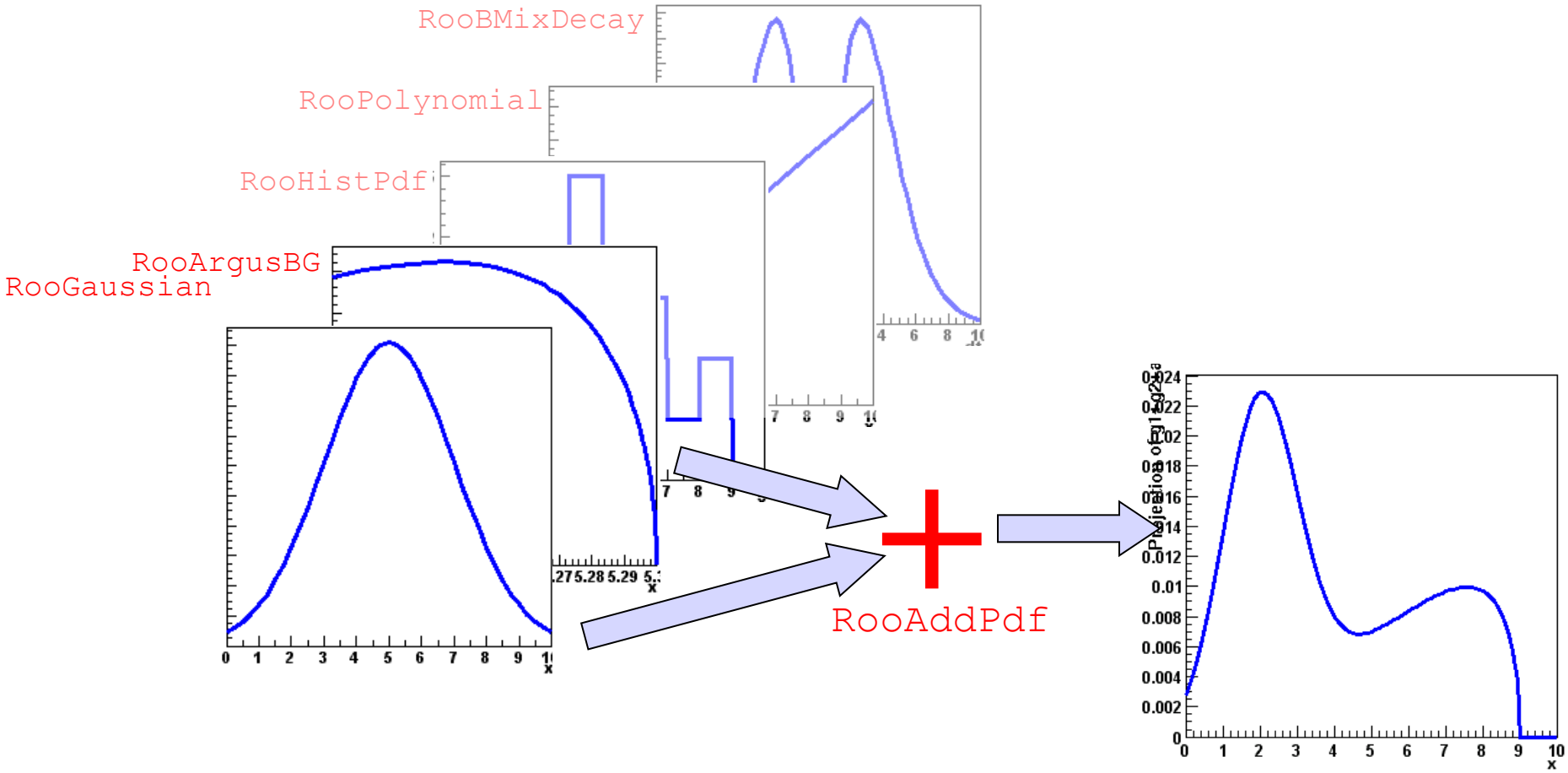
- Example: Re-parameterize pdf that expects slope in terms of a lifetime

```
w.factory("Exponential::model(x[0,20],expr('-1/tau',tau[1,20])) ;
```

Operator pdfs and the factory

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`**



Adding p.d.f.s – Mathematical side

- From math point of view adding p.d.f is simple
  - Two components *F, G*

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

  - Generically for *N* components P_0-P_N

$$S(x) = c_0 P_0(x) + c_1 P_1(x) + \ldots + c_{n-1} P_{n-1}(x) + \left(1 - \sum_{i=0,n-1} c_i\right) P_n(x)$$

- For *N* p.d.f.s, there are *N-1* fraction coefficients that should sum to less 1
  - The remainder is by construction 1 minus the sum of all other coefficients

Adding p.d.f.s – Factory syntax

- Additions created through a SUM expression

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

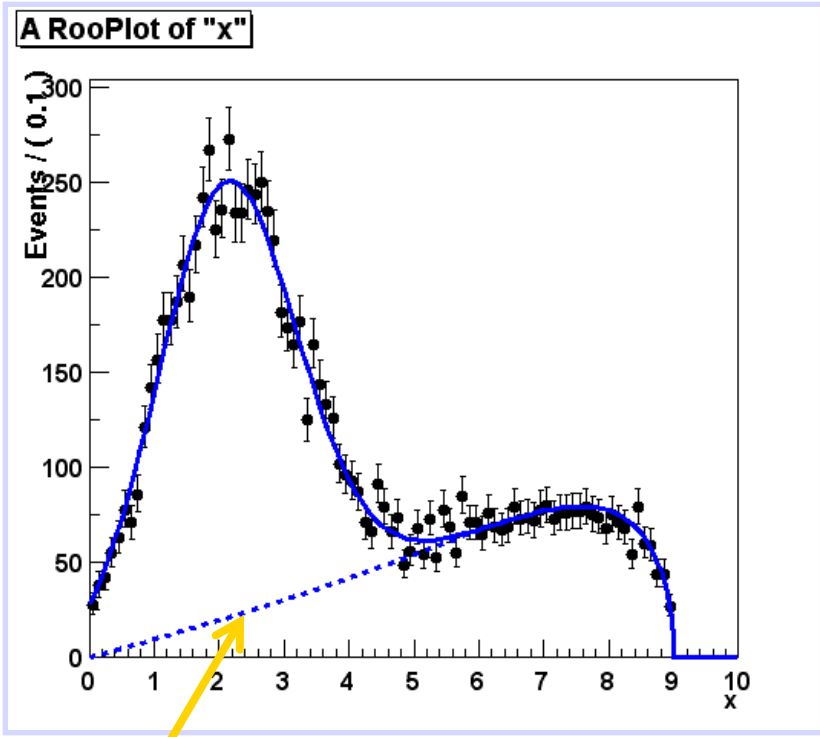
  - Note that last PDF does not have an associated fraction

- 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)")
```

Component plotting - Introduction

- 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 pdfs
  - Component plotting




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

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

Extended ML fits

- In an extended ML fit, an extra term is added to the likelihood

  Poisson(N_{obs},N_{exp})

- This is most useful in combination with a composite pdf

shape normalization

$$F(x) = f \cdot S(x) + (1-f)B(x) \quad ; \quad N_{\text{exp}} = N$$

$$f, N \Rightarrow N_S, N_B$$

$$F(x) = \frac{N_S}{N_S + N_B} \cdot S(x) + \frac{N_B}{N_S + N_B} B(x) \quad ; \quad N_{\text{exp}} = N_S + N_B$$

Write like this,
extended term automatically included in –log(L)

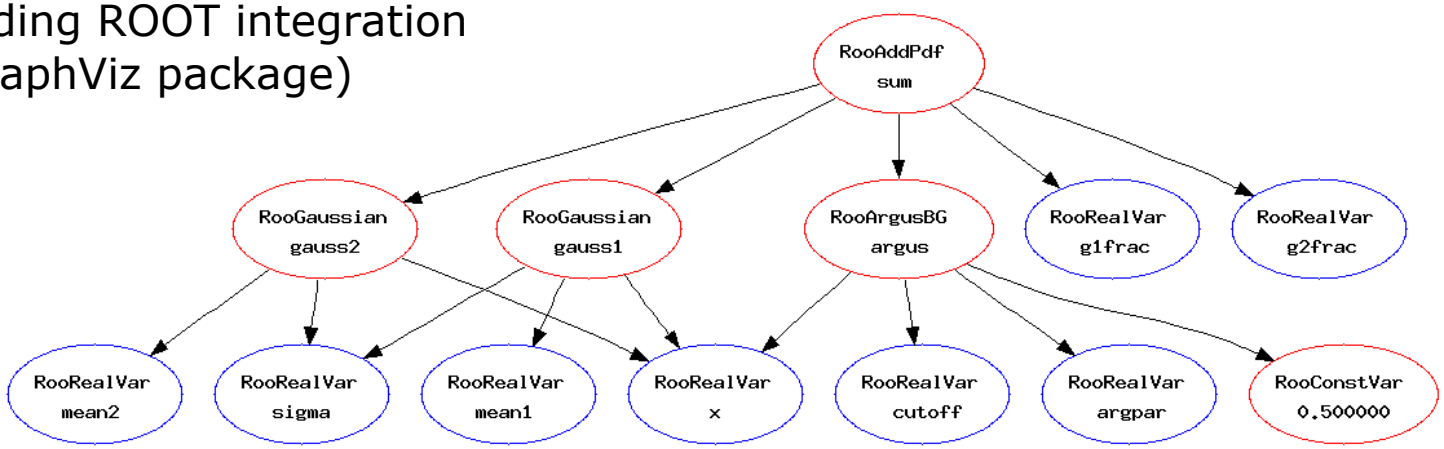
```
SUM::name(Nsig*S,Nbkg*B)
```

Operations on specific to composite pdfs

- Tree printing mode of workspace reveals component structure – **`w.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
```

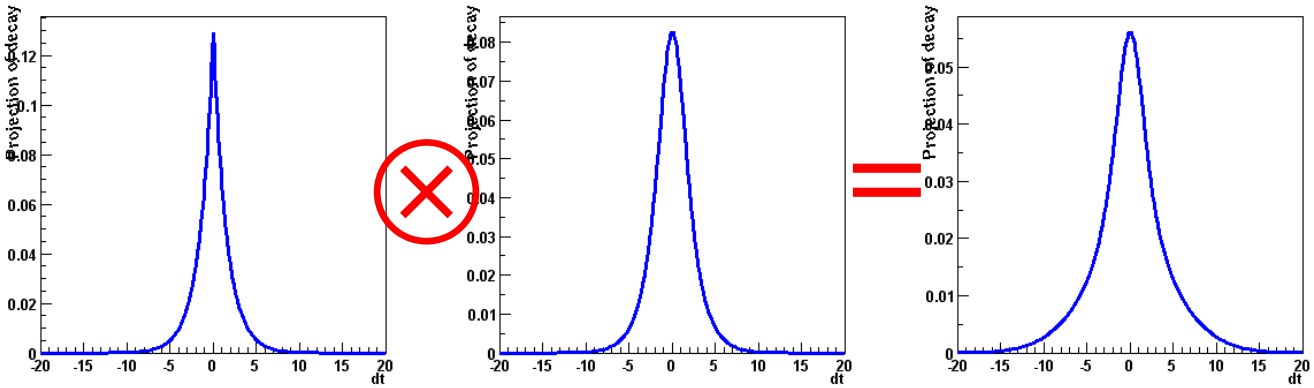
  - Can also make input files for GraphViz visualization (**`w::sum.graphVizTree("myfile.dot")`**)
  - Graph output on ROOT Canvas in near future (pending ROOT integration of GraphViz package)



Convolution

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

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



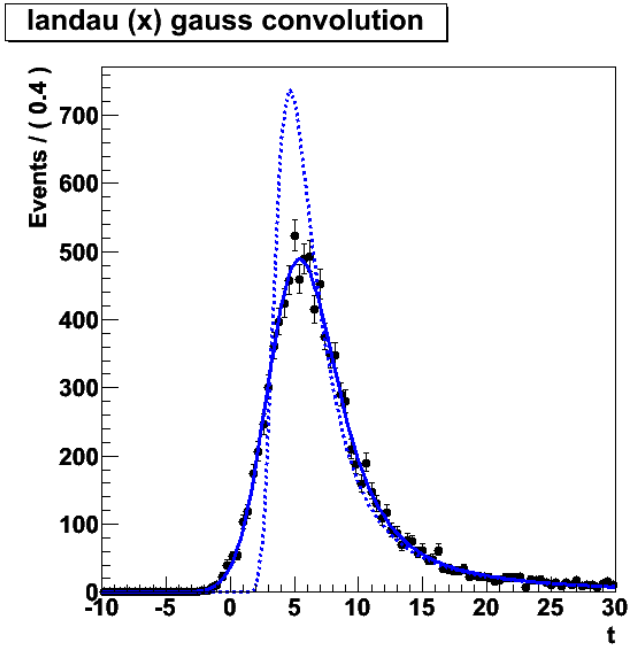
- But numeric calculation of convolution integral can be challenging. No one-size-fits-all solution, but 3 options available
  - Analytical convolution (BW⊗Gauss, various B physics decays)
  - Brute-force numeric calculation (slow)
  - FFT numeric convolution (fast, but some side effects)

Convolution

- Example

```
w.factory("Landau::L(x[-10,30],5,1)") :
w.factory("Gaussian::G(x,0,2)") ;

w::x.setBins("cache",10000) ; // FFT sampling density
w.factory("FCONV::LGf(x,L,G)") ; // FFT convolution

w.factory("NCONV::LGb(x,L,G)") ; // Numeric convolution
```

- FFT usually best
  - Fast: unbinned ML fit to 10K events take ~5 seconds
  - NB: Requires installation of FFTW package (free, but not default)
  - Beware of cyclical effects (some tools available to mitigate)

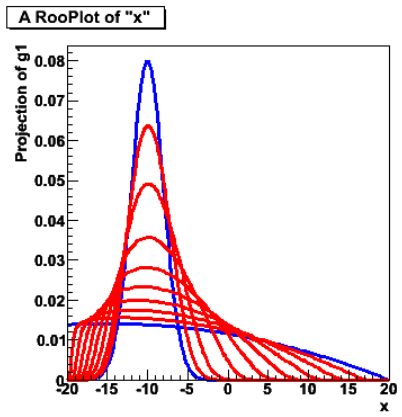


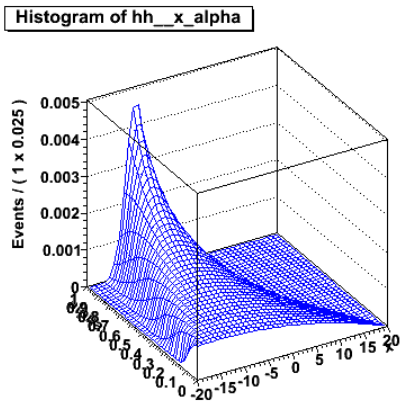
NB: libFFTW not installed for setup of this tutorial

Special pdfs – Morphing interpolation

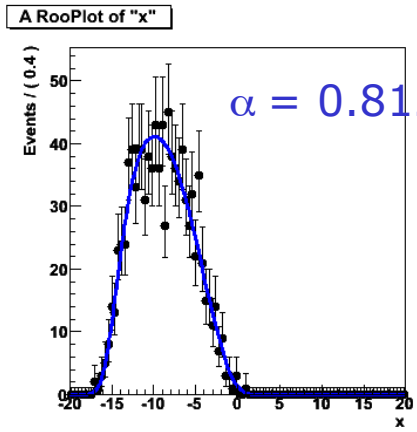
- Special operator pdfs can interpolate existing pdf shapes
  - Ex: interpolation between Gaussian and Polynomial

```
w.factory("Gaussian::g(x[-20,20],-10,2)") ;
w.factory("Polynomial::p(x,{-0.03,-0.001})") ;
w.factory("IntegralMorph::gp(g,p,x,alpha[0,1])") ;
```





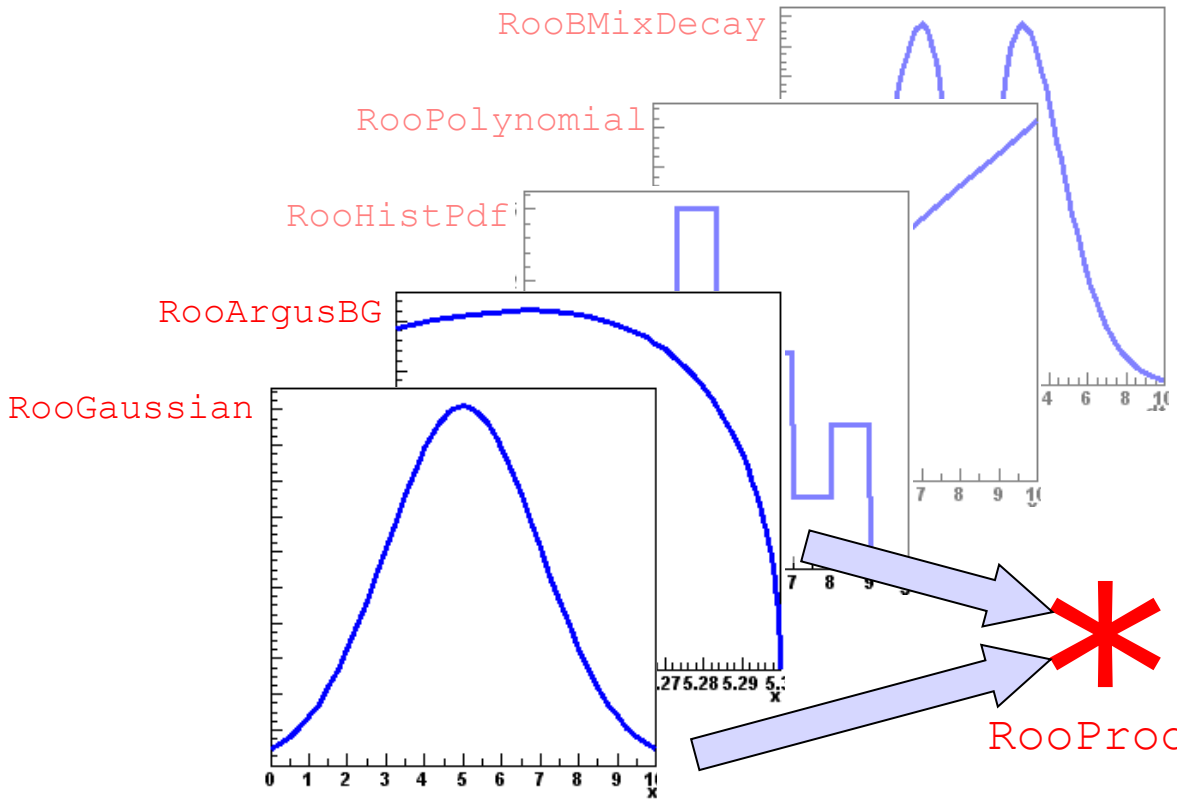
Fit to data



$\alpha = 0.812 \pm 0.008$

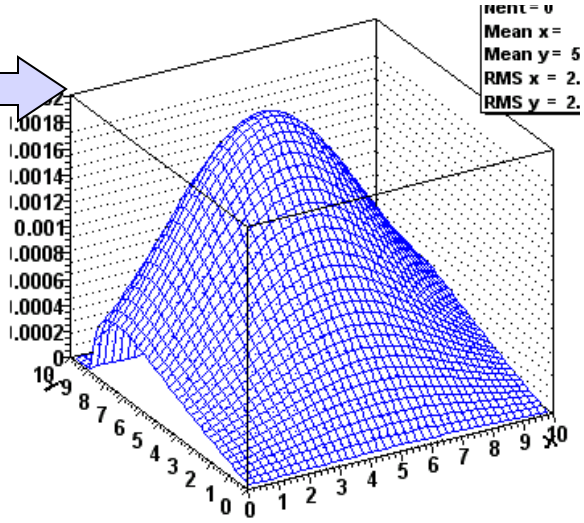
- Multiple morphing algorithms available, e.g.
  - IntegralMorph (Alex Read algorithm).
    CPU intensive, but good with discontinuities
  - MomentMorph (Max Baak).
    Fast, can handling multiple observables (and soon multiple interpolation parameters), but doesn't work well for all pdfs

Model building – Products of uncorrelated p.d.f.s



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

RooProdPdf



Uncorrelated products – Mathematics and constructors

- Mathematical construction of products of uncorrelated p.d.f.s is straightforward

2D

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

nD

$$H(x^{\{i\}}) = \prod_i F^{\{i\}}(x^{\{i\}})$$

  - No explicit normalization required → If input p.d.f.s are unit normalized, product is also unit normalized
  - (Partial) integration and toy MC generation automatically uses factorizing properties of product, e.g. $\int H(x,y)dx \equiv G(y)$ is deduced from structure.

- Corresponding factory operator is PROD

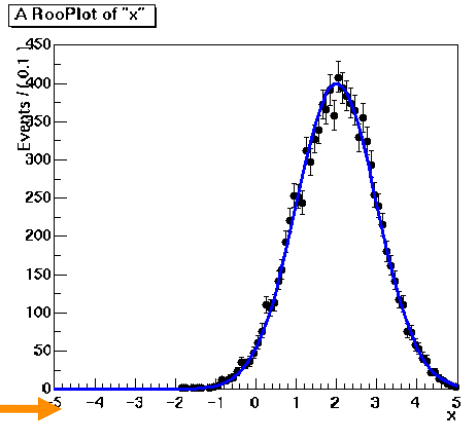
```
w.factory("Gaussian::gx(x[-5,5],mx[2],sx[1])") ;
w.factory("Gaussian::gy(y[-5,5],my[-2],sy[3])") ;

w.factory("PROD::gxy(gx,gy)") ;
```

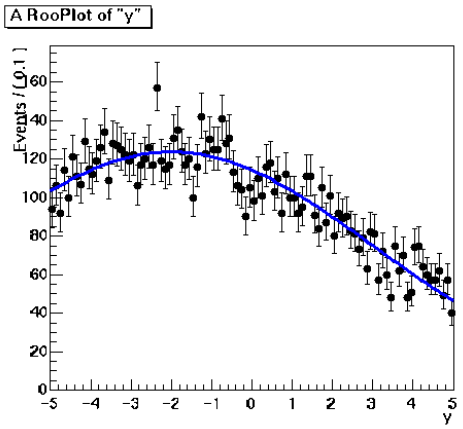
Plotting multi-dimensional models

- N-D models usually projected on 1-D for visualization
  - Happens automatically.
    RooPlots tracks observables of plotted data,
    subsequent models automatically integrated

```
RooDataSet* dxy =
w::gxy.generate(RooArgSet(w::x,w::y,10000));

RooPlot* frame = w::x.frame() ;
dxy->plotOn(frame) ;
w::gxy.plotOn(frame) ;
```

$$P_{gxy}(x) = \int gxy(x,y)dy$$





  - Projection integrals analytically reduced
    whenever possible
    (e.g. in case of factorizing pdf)

- To make 2,3D histogram of pdf

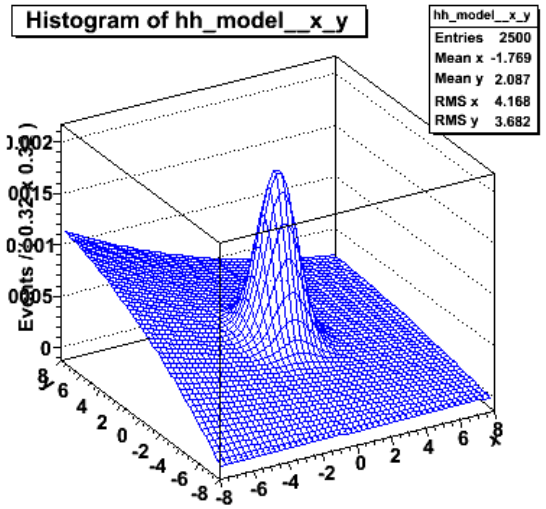
```
TH2* hh = w::gxy.createHistogram("x,y",50,50);
```

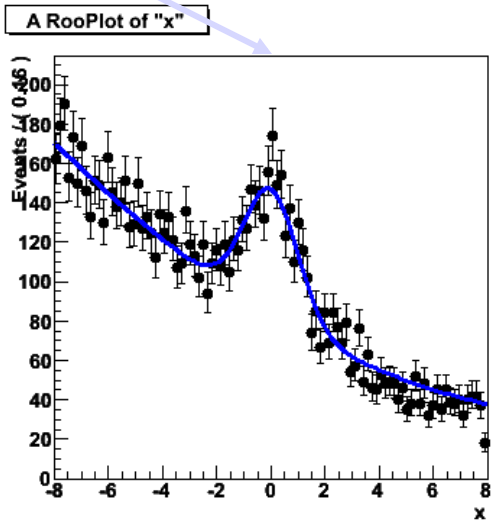
Can also project slices of a multi-dimensional pdf

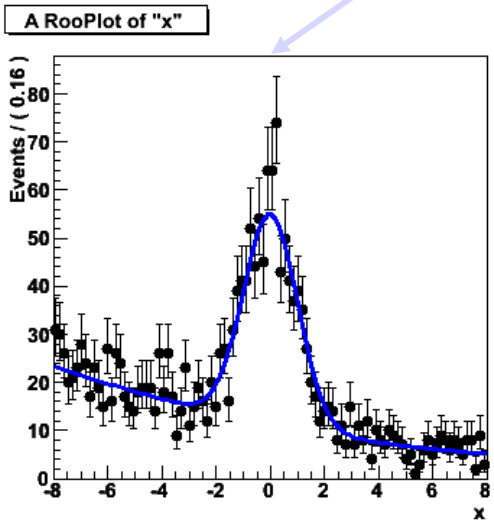
model(x,y) = gauss(x)*gauss(y) + poly(x)*poly(y)

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

```
y.setRange("sig",-1,1) ;
RooPlot* xframe2 = x.frame() ;
data->plotOn(xframe2,CutRange("sig")) ;
model.plotOn(xframe2,ProjectionRange("sig")) ;
```







→ Works also with >2D projections (just specify projection range on all projected observables)

→ Works also with multidimensional p.d.fs that have correlations

Introducing correlations through composition

- 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

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

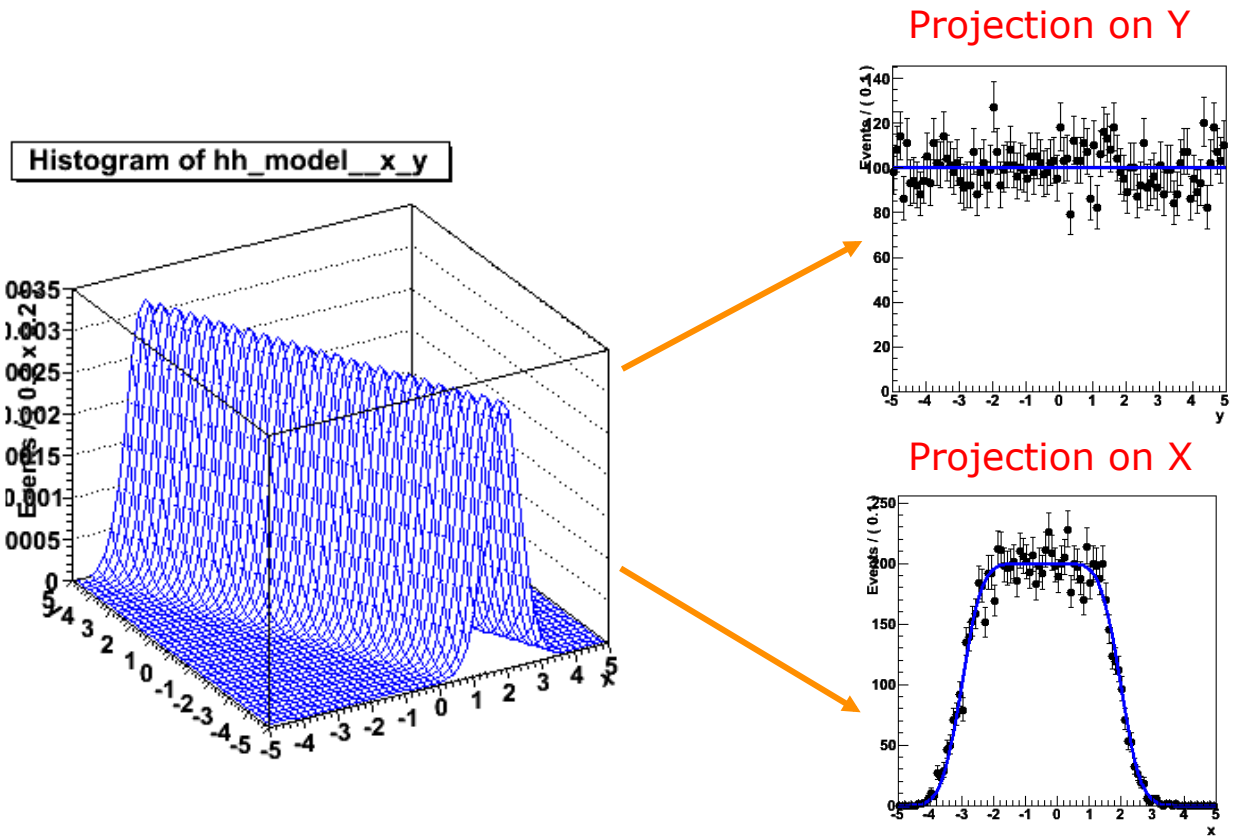
  - Example: Gaussian with shifting mean

```
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

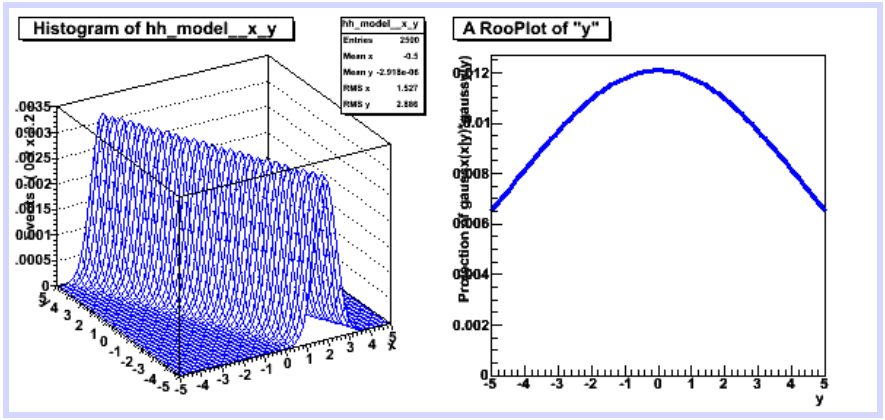
What does the example p.d.f look like?

- Use example model with x,y as observables

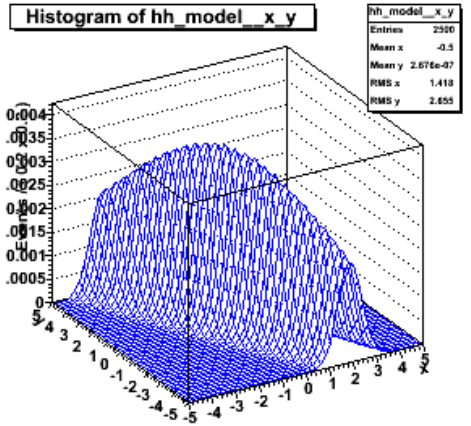


- Note flat distribution in y. Unlikely to describe data, solutions:
  1. Use as conditional p.d.f g(x|y,a,b)
  2. Use in conditional form multiplied by another pdf in y: g(x|y)*h(y)

Example with product of conditional and plain p.d.f.

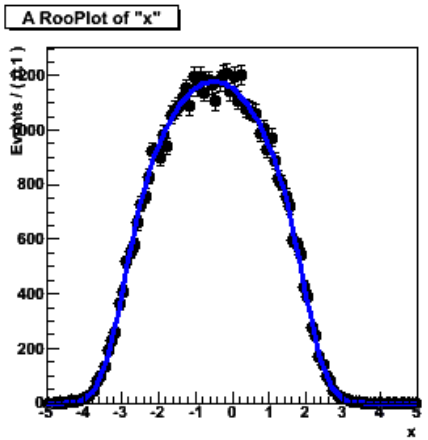


gx(x|y) * gy(y) =



model(x,y)

```
// I - Use g as conditional pdf g(x|y)
w::g.fitTo(data,ConditionalObservables(w::y)) ;

// II - Construct product with another pdf in y
w.factory("Gaussian::h(y,0,2)") ;
w.factory("PROD::gxy(g|y,h)") ;
```



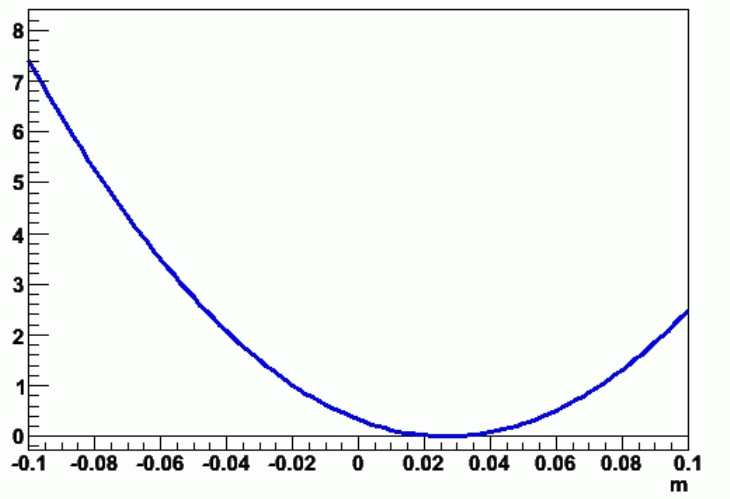
$$\int gx(x \mid y) g(y) dy$$

Limit setting 3 ways using RooFit models ‘by hand’

Constructing the likelihood

- So far focus on construction of pdfs, and basic use for fitting and toy event generation
- Can also explicitly construct the likelihood function of and pdf/data combination
  - Can use (plot, integrate) likelihood like any RooFit function object

```
RooAbsReal* nll = w::model.createNLL(data) ;

RooPlot* frame = w::param.frame() ;
nll->plotOn(frame,ShiftToZero()) ;
```



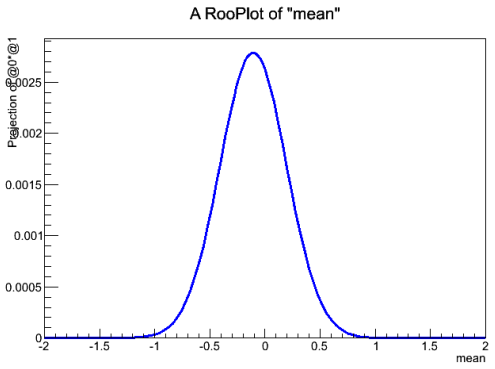
Minimizing the likelihood

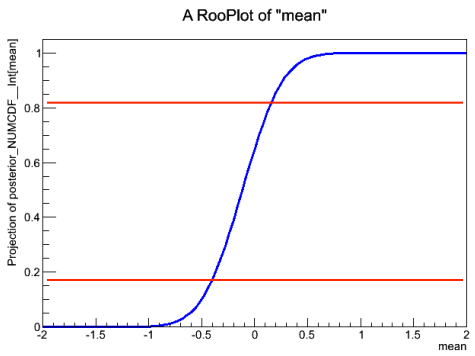
- Example – Manual MINUIT invocation
  - After each MINUIT command, result of operation are immediately propagated to RooFit variable objects (values and errors)

```
// Create likelihood (calculation parallelized on 8 cores)
RooAbsReal* nll = w::model.createNLL(data,NumCPU(8)) ;

RooMinuit m(*nll) ;  // Create MINUIT session
m.migrad() ;         // Call MIGRAD
m.hesse() ;          // Call HESSE
m.minos(w::param) ;  // Call MINOS for 'param'

RooFitResult* r = m.save() ; // Save status (cov matrix etc)
```

Constructing a Bayesian interval 'by hand'

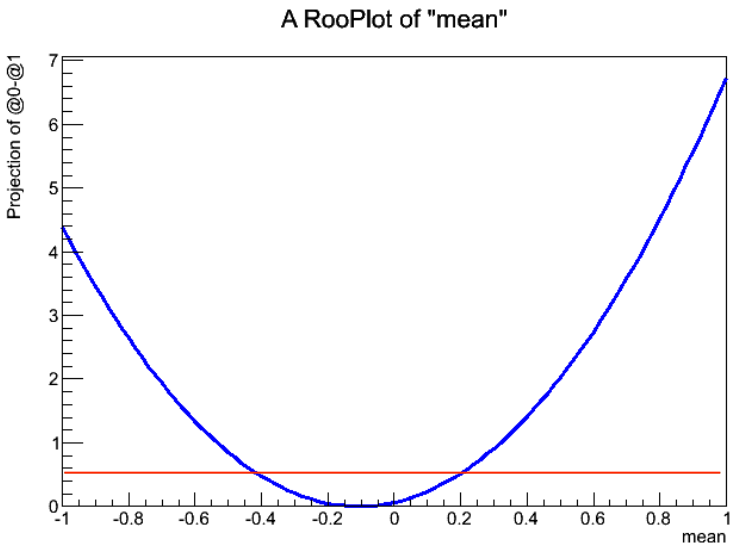
```
// Create pdf
RooWorkspace w("w",1) ;
w.factory("Gaussian::g(x[-10,10],mean[-10,10],sigma[3])") ;

// Create data
RooDataSet* d = w->pdf("g")->generate(*w->var("x"),100) ;

// Create L(mean|data)
RooAbsReal* nll = w->pdf("g")->createNLL(*d) ;
RooFormulaVar L("L","exp(-@0)",*nll) ;

// Create prior(mean)
w.factory("Uniform::prior(mean)") ;

// Create and plot posterior(mean)
RooGenericPdf posterior("posterior","@0*@1",
             RooArgSet(L,*w->pdf("prior"))) ;

RooPlot* frame = w->var("mean")->frame(-2,2) ;
posterior->plotOn(frame)
frame->Draw()

// Create and plot cdf of posterior
RooAbsReal* posterior_cdf = posterior->createCdf(*w->var("mean")) ;
RooPlot* frame = w->var("mean")->frame(-2,2) ;
posterior_cdf->plotOn(frame)
frame->Draw()
```





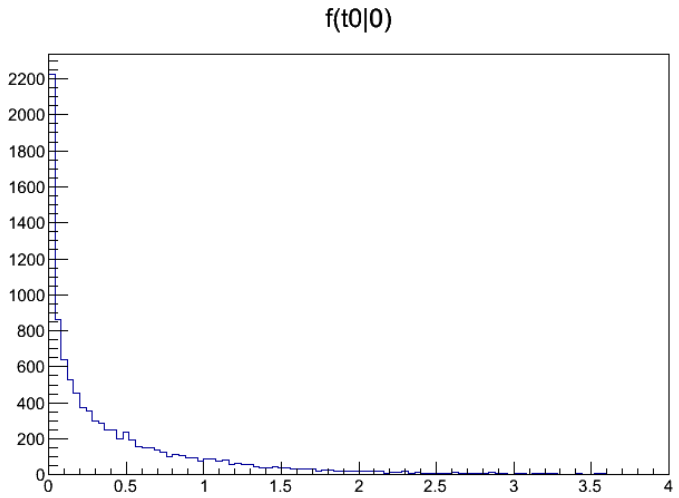
Construct a likelihood ratio interval

```
// Create pdf, data, -logL(mean) [identical to previous example]
RooWorkspace w("w",1) ;
w.factory("Gaussian::g(x[-10,10],mean[-10,10],sigma[3])") ;
RooDataSet* d = w->pdf("g")->generate(*w->var("x"),100) ;
RooAbsReal* nll = w->pdf("g")->createNLL(*d) ;

// Calculate L(mean-hat)
RooMinuit m(*nll) ;
m.migrad() ;
Double_t nll_hat = nll->getVal() ;

// Create log[ L(mean)/L(mean-hat) ]
RooFormulaVar lr("lr","@0-@1",
           RooArgSet(*nll,RooConst(nll_hat))) ;

// Plot likelihood ratio
RooPlot* frame = w->var("mean")->frame(-1,1) ;
lr.plotOn(frame) ;
frame->Draw() ;
```



Frequentist: calculate p-value for mean=0 hypothesis

```
TH1* h_t0 = new TH1D("h_t0","f(t0|0)",100,0,4) ;
for (int itoy=0 ; itoy<10000 ; itoy++) {

  // Generate dataset under hypothesis mean=0
  w->var("mean")->setVal(0) ;
  RooDataSet* toy = w->pdf("g")->generate(*w->var("x"),100) ;

  // Calculate -logL(mean_har), -logL(0) and ratio
  nll->setData(*toy) ;
  RooMinuit m(*nll) ;
  m.setPrintLevel(-1) ; m.migrad() ;
  Double_t logl_mean_hat = nll->getVal() ;

  w->var("mean")->setVal(0) ;
  Double_t logl_mean_zero = nll->getVal() ;
  h_t0->Fill(logl_mean_zero-logl_mean_hat) ;

  delete toy ;
}
h_t0->Draw() ;
```



Getting started – Online reference material

- RooFit class documentation (from code)
  - http://root.cern.ch/root/html/ROOFIT_ROOFITCORE_Index.html
  - http://root.cern.ch/root/html/ROOFIT_ROOFIT_Index.html
- RooFit home page at ROOT web site
  - http://root.cern.ch/drupal/content/roofit
  - Has links to manual and tutorial macros → The Quick Start guide has most of the factory syntax explained.

Exercise A – the basics

- Copy mod2/ex2A.C, look at it and run it
- This macro
  - Creates an exponential pdf in a workspace using the factory
  - Extracts pointers to pdf and variables in the workspace
  - Generates an unbinned dataset
  - Plots the data
  - Performs an unbinned ML fit to the data
  - Plot the fit result on data
- Change the formulation of the exponential
  - RooExponential implements f(x;a) = 1/a * exp (a*x)
  - Reformulate in it terms of a lifetime tau, so that a = -1/tau (use the factory method 'expr::funcname('tformula',args…)' to construct a formula expression that performs the appropriate substitution in the code

Exercise B – Persisting workspaces

- Create a new macro ex2B.C that uses the factory syntax to create a Gaussian pdf in a workspace w
  - Use class RooGaussian, name it "gaus", and with an observable x [0,10], a mean mu[0,10] with initial value 5 and and width s that is fixed to 1.
  - Create an unbinned dataset of 1000 events samples from this pdf
  - Import the pdf in to the workspace with the RooWorkspace::import() command. Give the dataset the name "obsData" in the workspace by adding the 'Rename("obsData")' argument to the import command.
    - If you get a syntax error on the Rename() function, please add a statement 'using namespace RooFit' at the beginning of your file
  - Write the workspace to a ROOT file using w.writeToFile(model.root);
- Write a second macro file ex2B_read.C that
  - Reads the file model.root (using Tfile* Tfile::Open()) and retrieves the workspace from the file
  - Access the pdf and variable from the workspace (as done in Ex2A), fits the pdf to data and plots both data and pdf.

Exercise C – Composite models

- Create a new macro ex2C.C that construct a composite model consisting of a Gaussian signal and an exponential background
  - You can recycle the factory lines from the preceeding macros, but in the Gaussian set mu=3 and s=0.3
  - Construct a composite model using the SUM::model(f*F,G) operator introducing a signal fraction fsig for the signal with range [0,1] and initial value 0.1
  - Generate 1000 events, fit that dataset and plot data and model
  - Also plot the background component of the model with a second plotOn() line that adds 'Components("expo")' and 'LineStyle(kDashed)' to the plotOn() command line
  - Modify the fitTo() command line: add a 'Save()' argument to the command line and save the return value as a 'RooFitResult *r'.
  - Print the fit Result (r->Print()).
  - Visualize the correlation matrix: r->correlationHist()->Draw("colz") ;
  - Visualize the fit uncertainty in the plot: before the first plotOn() call add a new plotOn() call identical to the first one, but with the additional argument 'VisualizeError(*r)'
    - In the end you have 3 plotOn() calls: the 1st one with VisualizeError(), 2nd one plain and the 3rd one with Components()

Exercise D – working with the likelihood function

- Copy ex2C.C in ex2D.C and keep the code up to (and including) the part where the dataset is generated
- Fix the tau parameter of the background model
  - Get a RooRealVar pointer out of the workspace and call method setConstant (kTRUE)
- Create the –logL(fsig) function
  - RooAbsReal* nll = pdf.createNLL(*data) ;
- Create a plot frame from parameter fsig and plot the –log(L) function in it.
  - Using ROOTs interactive zoom function find the approximate location of the minimum. Adjust the call to fsig.frame() to specificy a min and max that zoom in on the minumum: frame(xmin,xmax)
  - Add the utility command ShiftToZero() to the plotOn() call of the logL function to shift curve down so that the minimum is put at zero (NB: you are effectively plotting the log-likelihood ratio log(L(fsig)/L(fsig-hat)) now!)
- Measure the interval where the likelihood ratio curve intersects LLR = +0.5.
  - This is the likelihood ratio interval for fsig. Compare this to the fit error for fsig.

Exercise D - continued

- Now construct a Bayesian posterior from the likelihood and an (implicit) flat prior pi(fsig)
  - Create RooGenericPdf post("post","exp(-@0)",*nll)
  - This class construct the likelihood from –log(L) and normalizes the distribution [numerically] so that it becomes a pdf (the posterior)
  - Plot the posterior function on a plot frame from fsig
- Now we construct a central 68% Bayesian interval
  - A central interval is defined by having equal fractions of the distribution on either side, so for a 68% interval the boundaries are defined by the points that divide the probability in three regions with 16% : 68% : 16% of the probability.
  - To find this distribution, create the cumulative distribution F(fsig) from post(fsig) from the posterior: RooAbsReal* cdf = post.createCdf(fsig)
  - Plot the F(sig) in the same frame. Construct a central Bayesian 68% interval from the posterior by finding the intersection of the cumulative distribution with 16% and 16+68=84%.

Exercise E – Hypothesis tests

- Copy input file mod2/ex2E.C which
  - Constructs a Gaussian+Exponential pdf with all parameters fixed except fsig
  - Generates 1000 toy datasets with fsig=0 and evaluates for each of those q0 = L(0)/L(fsig-hat)
- Extend the macro as follows
  - Calculate q0 for the dataset 'data' (which was generated with fsig=0.1). To do so copy the relevant code from the toy loop to a place below the toy loop, while attaching the likelihood to dataset 'data' instead of one of the toy datasets. Can you give an estimate of the p-value for the background hypothesis for this dataset (using the histogram h_q0 as distribution for q0 for background-only datasets)
  - Calculate the distribution of q0 for data that has fsig=0.1. To so do, copy the complete toy generation loop and change the name of the TH1 from h_q0 to h_q0_prime and change the value of fsig from 0 to 0.1 when the data is generated. Plot the distribution of h_q0 and h_q0_prime on top of each other.