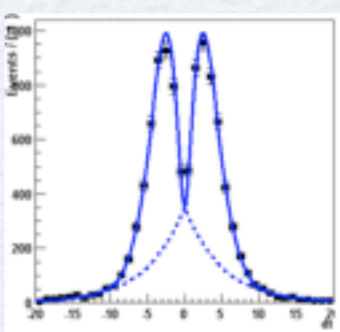
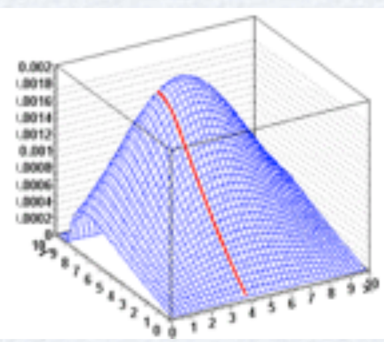
# Statistical Software Tools

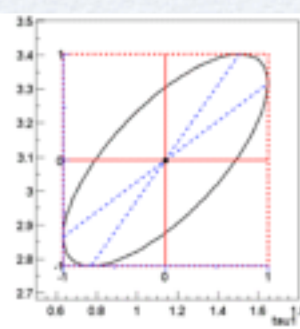
# RooFit/RooStats

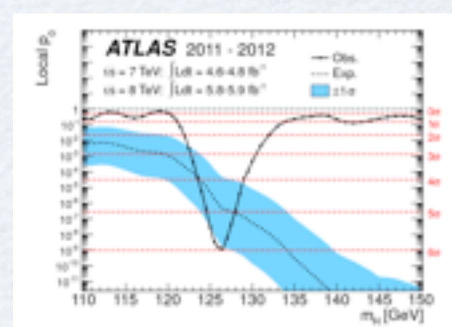
*Lorenzo Moneta (CERN)*

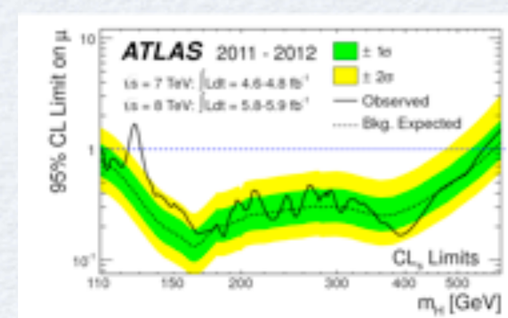
*INFN School of Statistics 2013, Vietri*









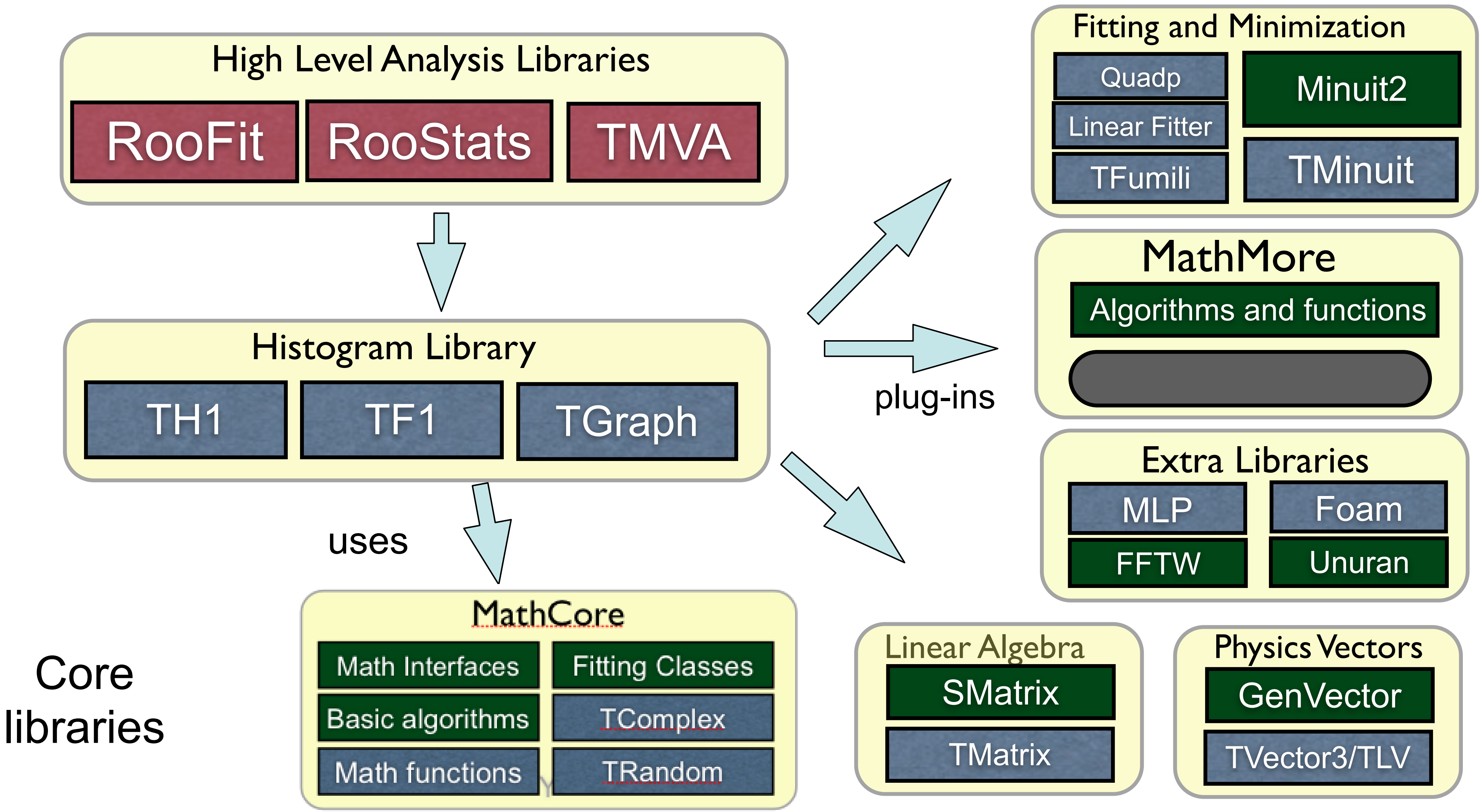



# Introduction

- We will cover only RooFit/RooStats
- Statistical tools for:
  - point estimation: determine the best estimate of a parameter
  - estimation of confidence (credible) intervals
    - lower/upper limits or multi-dimensional contours
  - hypothesis tests:
    - evaluation of p-value for one or multiple hypotheses (discovery significance)
- Model description and sharing of results
  - analysis combination

# ROOT Math/Stat Libraries

- Large set of mathematical libraries and tools needed for event reconstruction, simulation and statistical data analysis



# Function Minimization

- Common interface class (`ROOT::Math::Minimizer`)
- Existing implementations available as plug-ins:
  - **Minuit** (based on class `TMinuit`, direct translation from Fortran code)
    - with Migrad, Simplex, Minimize algorithms
  - **Minuit2** (new C++ implementation with OO design)
    - with Migrad, Simplex, Minimize and Fumili2
  - **Fumili** (only for least-square or log-likelihood minimizations)
  - **GSLMultiMin**: conjugate gradient minimization algorithm from GSL (Fletcher-Reeves, BFGS)
  - **GSLMultiFit**: Levenberg-Marquardt (for minimizing least square functions) from GSL
  - **Linear** for least square functions (direct solution, non-iterative method)
  - **GSLSimAn**: Simulated Annealing from GSL
  - **Genetic**: based on a genetic algorithm implemented in TMVA
- All these are available for ROOT fitting and in RooFit/RooStats
- Possible to combine them (e.g. use Minuit and Genetic)
- Easy to extend and add new implementations
  - e.g. minimizer based on NagC exists in the development branch (see here)

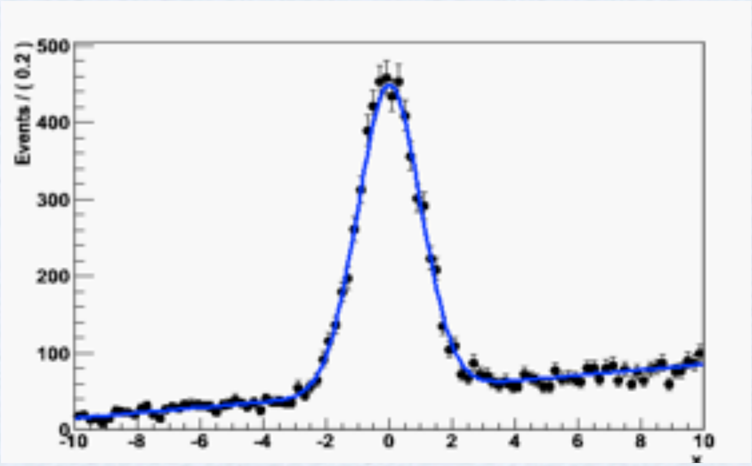
# Outline

- Introduction to RooFit
  - Basic functionality
  - Model building using the workspace
  - Composite models
- Exercises on RooFit:
  - building and fitting models
- Introduction to RooStats
  - Interval estimation tools (Likelihood/Bayesian)
  - Hypothesis tests
  - Frequentist interval/limit calculator (CLs)
- Exercises on interval/limit estimation and discovery significance (hypothesis test)

*Material based on slides from W. Verkerke (author of RooFit)*

# RooFit

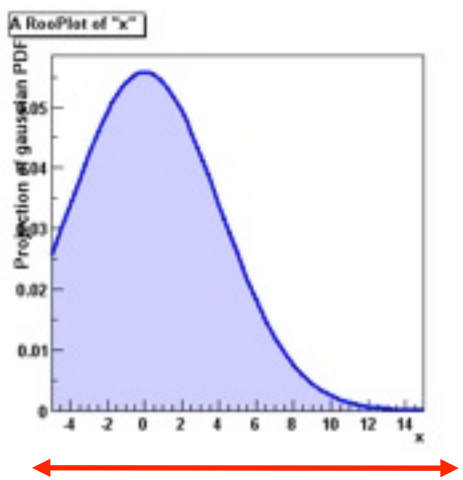
- Toolkit for data modeling
  - developed by *W. Verkerke and D. Kirkby*
- model distribution of observable ***x*** in terms of parameters ***p***
  - probability density function (pdf): $\mathcal{P}$`(x;p)`
- pdf are normalized over allowed range of observables ***x*** with respect to the parameters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



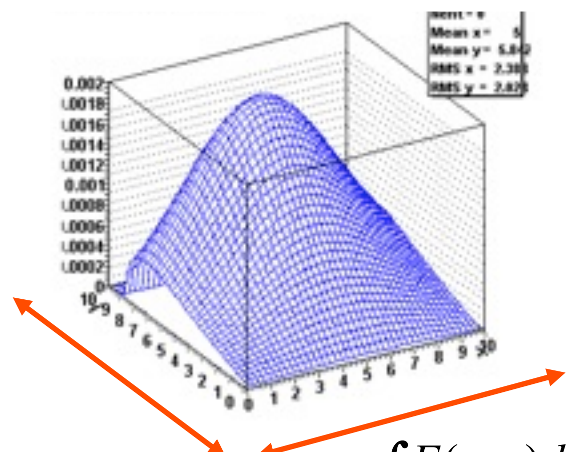
# Mathematic – Probability density functions

- 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

# Why RooFit ?

- ROOT function framework can handle complicated functions but difficult for users
  - require writing large amount of code
- Normalization of p.d.f. not always trivial
  - RooFit does automatically for user
- In complex fit, computation performance important
  - need to optimize code for acceptable performance
  - RooFit provides built-in optimization
    - evaluation only when needed
- Simultaneous fit to different data samples
- Provide full description of model for further use

# 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

# RooFit Modeling

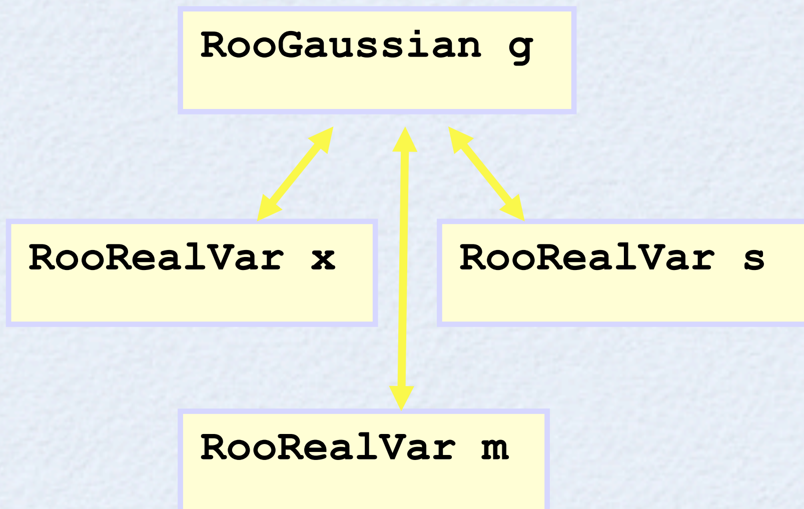
Mathematical concepts are represented as C++ objects

| 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:

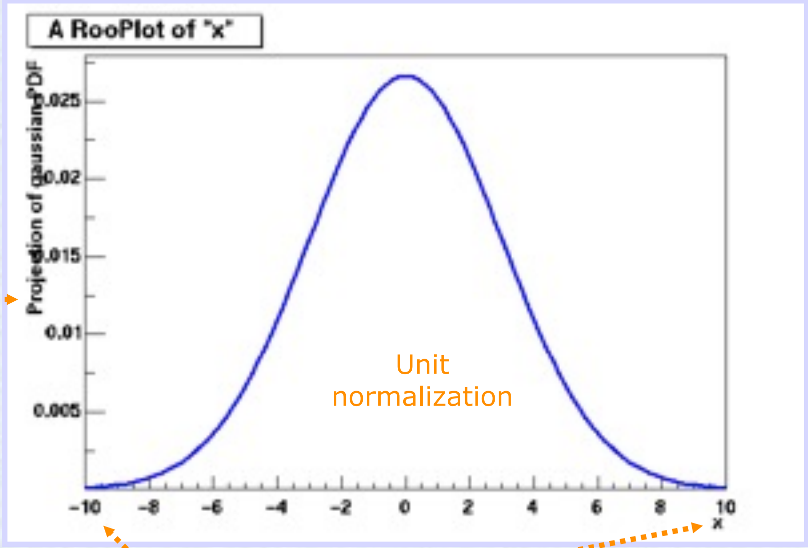
```
RooRealVar x(“x”,”x”,2,-10,10)
RooRealVar s(“s”,”s”,3) ;
RooRealVar m(“m”,”m”,0) ;
RooGaussian g(“g”,”g”,x,m,s)
```

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

# RooFit Functionality

- pdf visualization

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

# RooFit Functionality

- Toy MC generation from any pdf

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

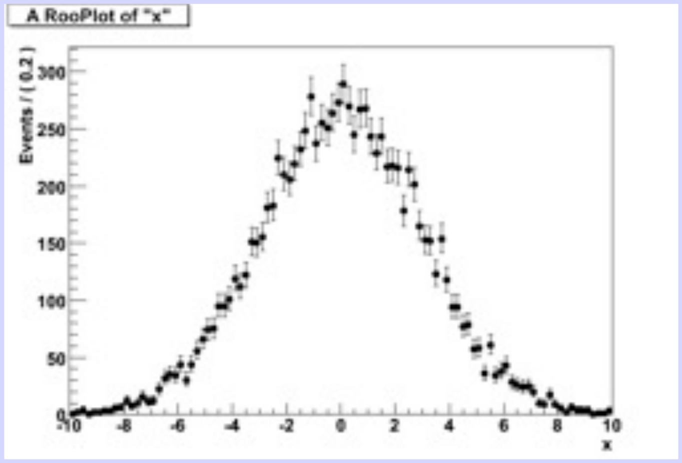
```
RooDataSet * data = pdf->generate(*x,10000);
```

- data visualization

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

Note that dataset is ***unbinned***
(vector of data points, x, values)

Binning into histogram is performed in `data->plotOn()` call



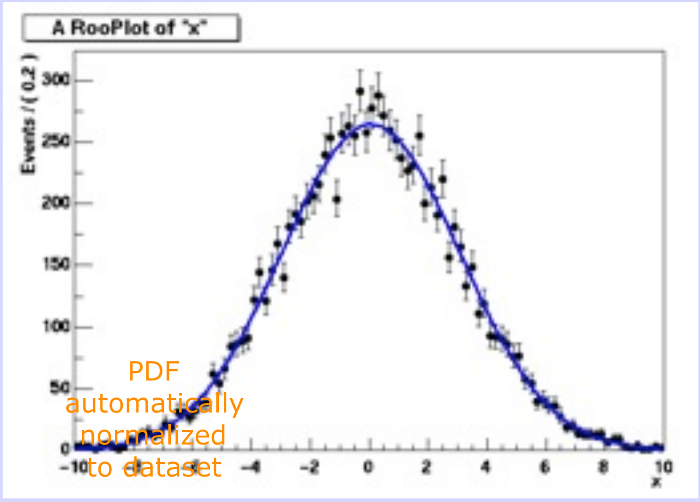
# RooFit Functionality

- Fit of model to data
  - e.g. unbinned maximum likelihood fit

```
pdf = pdf->fitTo(data);
```

- data and pdf visualization after fit

```
RooPlot * xframe = x->frame();
data->plotOn(xframe);
pdf->plotOn(xframe);
xframe->Draw();
```



# RooFit Workspace

- **`RooWorkspace`** class: container for all objected created:
  - 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("w");
workspace.import(*data);
workspace.import(*pdf);
workspace.writeToFile("myWorkspace.root")
```

# 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)
```

The workspace provides a factory method to auto-generates objects from a math-like language
(the p.d.f is made with 1 line of code instead of 4)

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

In the tutorial we will work using the workspace factory to build models

# 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(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

RooRealVar * x = w.var("x");

RooAbsPdf * f = w.pdf("f");
```

- 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

}
```

# 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)
```

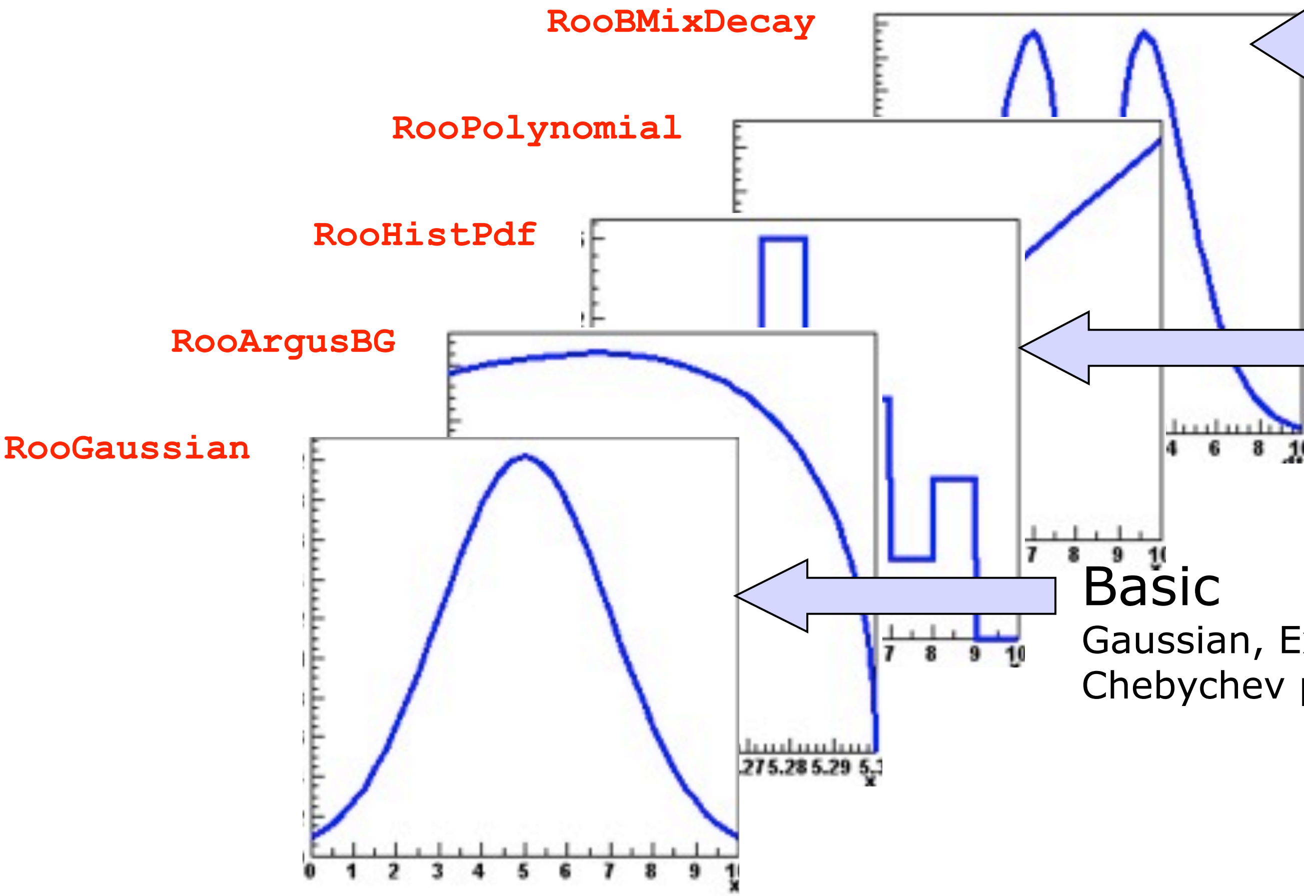
- Miscellaneous points
  - You can always use numeric literals where values or functions are expected
  - It is not required to give component objects a name, e.g.

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

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

# 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***

# Model building – (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$\otimes$G) | `Voigtian::v(x,mean,gamma,sigma)` |

# Factory syntax – using expressions

- Customized p.d.f from 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)") ;
```

- re-parametrization of variables (making functions)

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

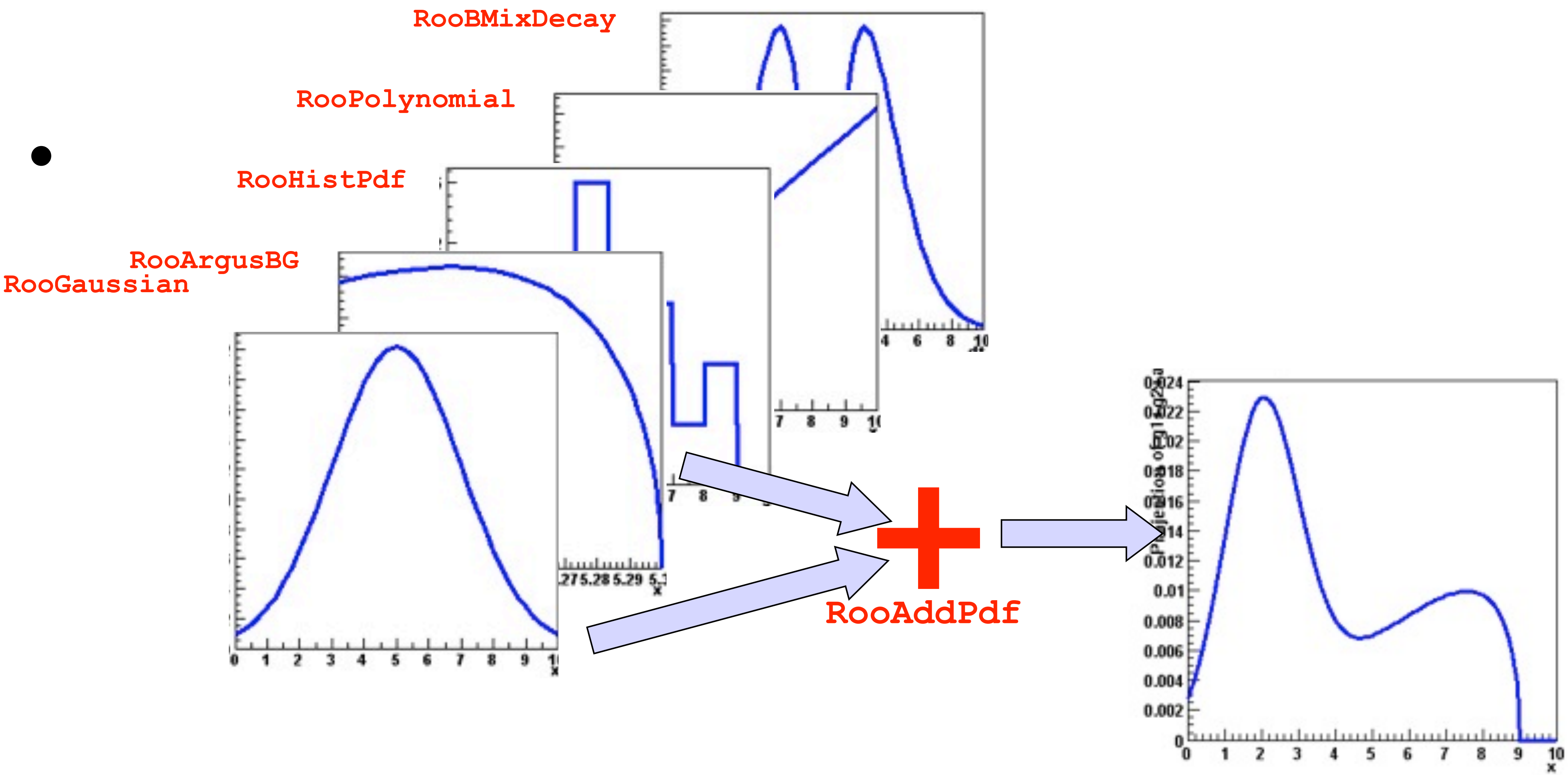
  - note using expr (builds a function, a RooAbsReal)
  - instead of EXPR (builds a pdf, a RooAbsPdf)

This usage of upper vs lower case applies also for other factory commands (SUM, PROD,.... )

# 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`**

•



# Factory syntax: Adding p.d.f.

- Additions of PDF (using fractions)

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

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

  - Note that last PDF does not have an associated fraction

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

- PDF additions (using expected events instead of fractions)

```
SUM::name(Nsig*SigPDF,Nbkg*BkgPDF)
```

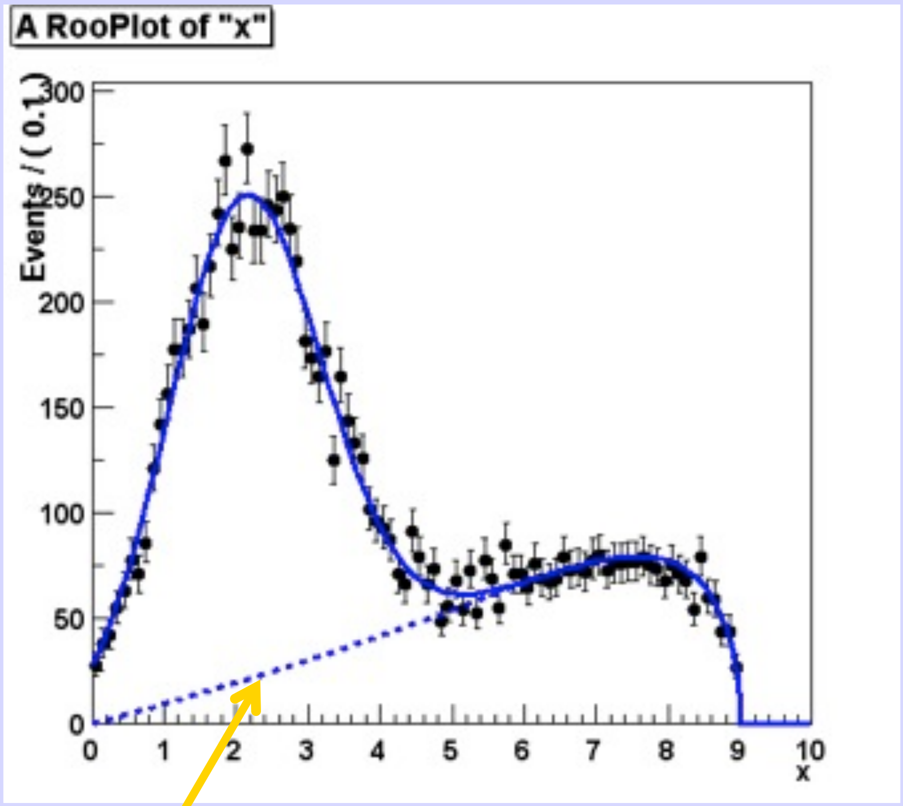
$$F(x) = \frac{N_S}{N_S + N_B} \times S(x) + \frac{N_B}{N_S + N_B} B(x) \quad ; \quad N_{\exp} = N_S + N_B$$

  - the resulting model will be extended
  - the likelihood will contain a Poisson term depending on the total number of expected events (Nsig+Nbkg)

$L\ (x \mid p) \to L(x|p)\mathrm{Poisson}(N_{obs},N_{exp})$

# Component plotting - Introduction

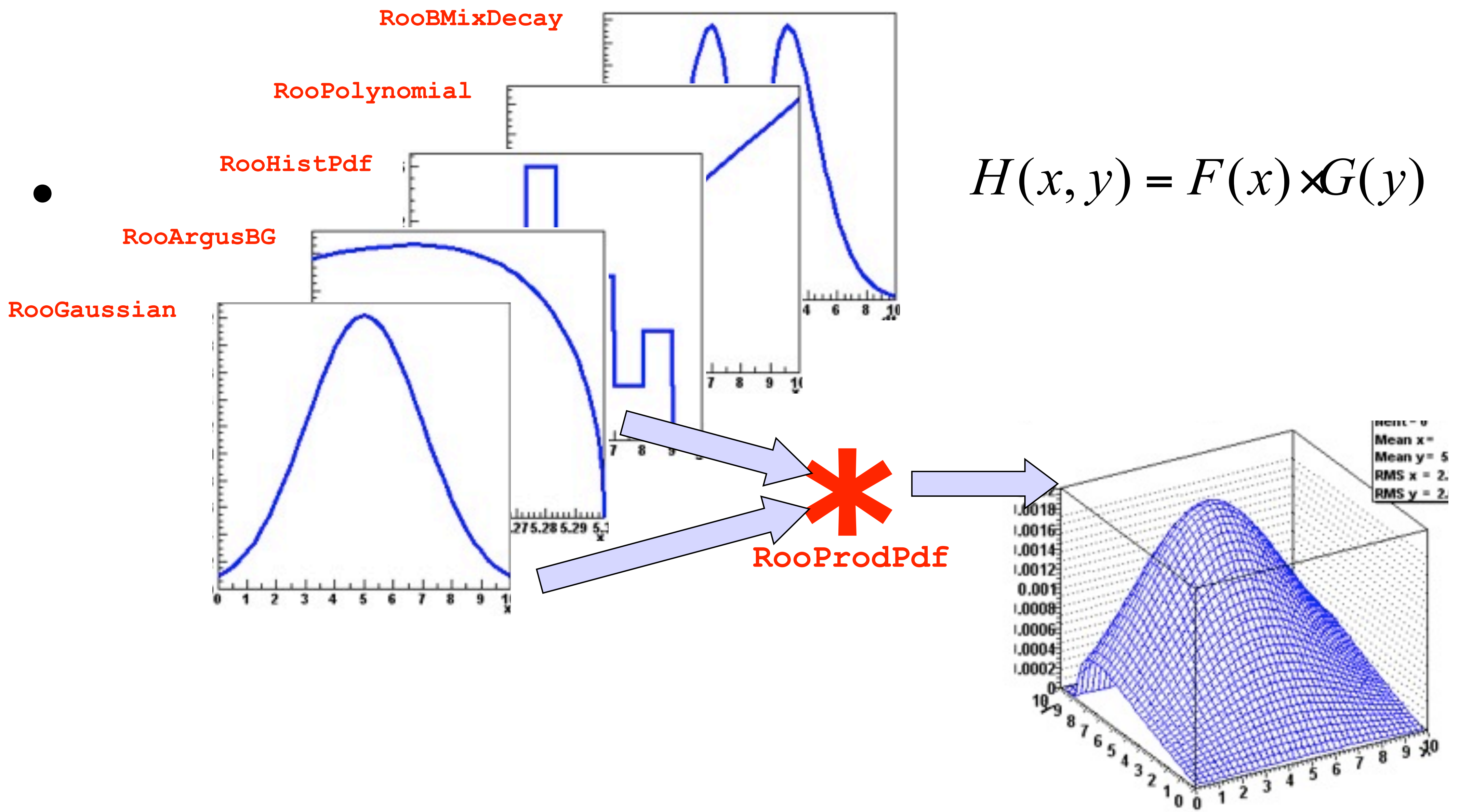
- Plotting, toy event generation and fitting works identically for composite p.d.f.s
  - Several optimizations applied behind the scenes that are specific to composite models (e.g. delegate event generation to components)
- Extra plotting functionality specific to composite pdfs
  - Component plotting




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

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

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

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



## Uncorrelated products – Mathematics and constructors

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

**2D**

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

**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)") ;
```

# 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])") ;
```

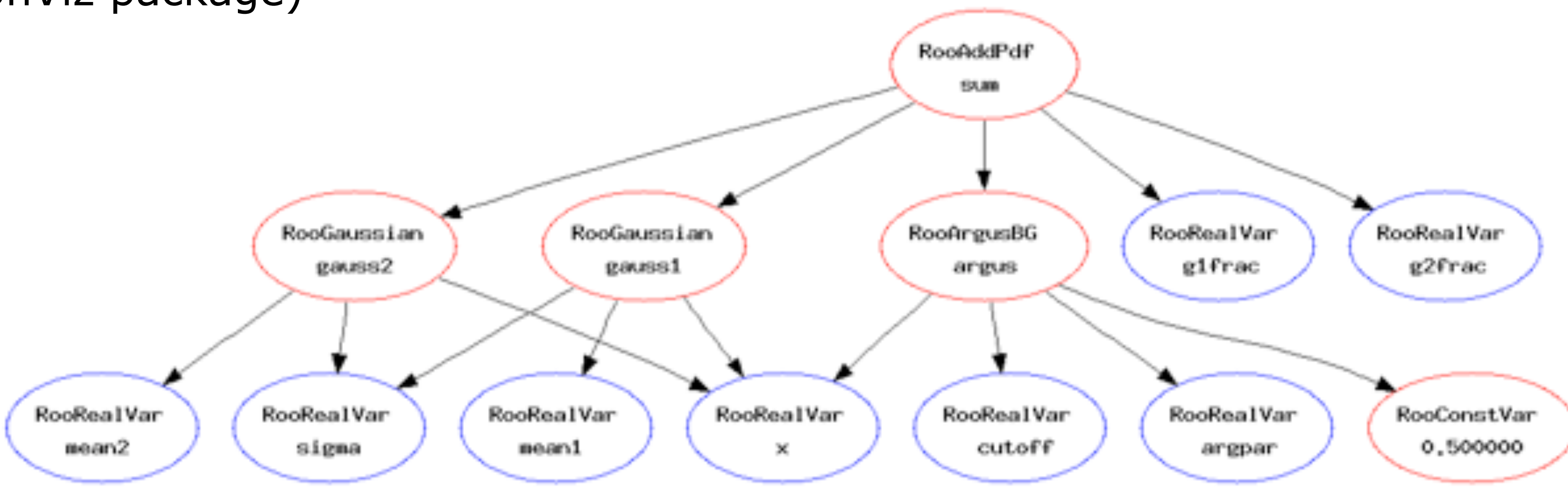
  - No assumption made in function on a,b,x,y being observables or parameters, any combination will work

# 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.pdf("sum")->graphVizTree("myfile.dot")**)
  - Graph output on ROOT Canvas in near future (pending ROOT integration of GraphViz package)



# Constructing joint pdfs (RooSimultaneous)

- Operator class SIMUL to construct joint models at the pdf level
  - need a discrete observable (category) to label the channels

```
// Pdfs for channels 'A' and 'B'
w.factory("Gaussian::pdfA(x[-10,10],mean[-10,10],sigma[3])") ;
w.factory("Uniform::pdfB(x)") ;

// Create discrete observable to label channels
w.factory("index[A,B]") ;

// Create joint pdf (RooSimultaneous)
w.factory("SIMUL::joint(index,A=pdfA,B=pdfB)") ;
```

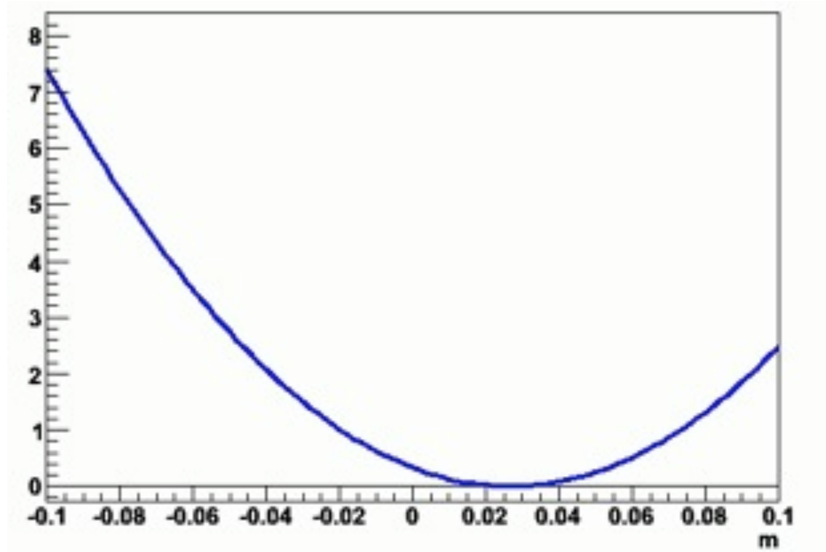
- Construct joint datasets
  - contains observables ("x") and category ("index")

```
RooDataSet *dataA, *dataB ;
RooDataSet dataAB("dataAB","dataAB",
                  RooArgSet(*w.var("x"),*w.cat("index")),
                  Index(*w.cat("index")),
                  Import("A",*dataA),Import("B",*dataB)) ;
```

# 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 = pdf->createNLL(data) ;

RooPlot* frame = parameter->frame() ;
nll->plotOn(frame,ShiftToZero()) ;
```



# Constructing the likelihood

- Example – Manual MIMIZATION using MINUIT
  - Result of minimization 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)) ;


RooMinimizer m(*nll) ;              // create Minimizer class
m.minimize("Minuit2","Migrad");    // minimize using Minuit2
m.hesse() ;                        // Call HESSE
m.minos(w::param) ;                // Call MINOS for 'param'

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

  - Also other minimizers (Minuit, GSL etc) supported

  - N.B. Different minimizer can also be used from RooAbsPdf::fitTo

```
//fit a pdf to a data set using Minuit2 as minimizer
pdf.fitTo(*data, RooFit::Minimizer("Minuit2","Migrad")) ;
```

# Minuit2

- Object-Oriented version of Minuit (re-written in C++)
  - same functionality with some improvements
    - single side parameter limits
    - better tools to debug minimization
    - capability to retrieve all information at each iteration
    - added Fumili algorithm
  - support parallelization in gradient calculation
- Used now for complex fits in RooFit/RooStats (e.g. Higgs discovery fits)
- Found to be more robust and able to converge faster (less iterations)

# Running Minuit2

- To use Minuit2 for fitting:
  - `pdf->fitTo(*data, RooFit::Minimizer(“Minuit2”,”Migrad”));`
- or when using ROOT fitting (TH1::Fit) or RooStats:
  - `ROOT::Math::MinimizerOptions::SetDefaultMinimizer(“Minuit2”)`
- Example of output log:

```
MnSeedGenerator: for initial parameters FCN = 0
MnSeedGenerator: Initial state:   - FCN =                0 Edm =      1303.17 NCalls =      9
VariableMetric: start iterating until Edm is < 0.001
VariableMetric: Initial state   - FCN =                0 Edm =      1303.17 NCalls =      9
VariableMetric: Iteration #   1 - FCN =  -1244.112454315 Edm =      47.6952 NCalls =     18
VariableMetric: Iteration #   2 - FCN =  -1477.322027873 Edm =     0.337079 NCalls =     31
VariableMetric: Iteration #   3 - FCN =  -1478.857831678 Edm =    0.0555537 NCalls =     37
VariableMetric: Iteration #   4 - FCN =  -1479.014254322 Edm =   0.00688715 NCalls =     43
VariableMetric: Iteration #   5 - FCN =  -1479.022055997 Edm =  3.78846e-08 NCalls =     49
VariableMetric: After Hessian   - FCN =  -1479.022055997 Edm =  4.12083e-08 NCalls =     59
Minuit2Minimizer : Valid minimum - status = 0
FVAL  = -1479.02205599658964
Edm   = 4.12082593076253669e-08
Nfcn  = 59
mu          = 1.05369 +/-   0.0656498       (limited)
sigma       = 2.07586 +/-   0.0464542       (limited)
```

# Adding parameter pdfs to the likelihood

- Systematic/external uncertainties can be modeled with regular RooFit pdf objects.
- To incorporate in likelihood, simply multiply with original pdf

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

w.factory(“PROD::gprime(f,Gaussian(mean,1.15,0.30))”) ;
```

$$-\log L(\mu,\sigma) = -\sum_{data} -\log(f(x_i;\mu,\sigma) - \log(Gauss(\mu,1.15,0.30))$$

– Any pdf can be supplied, e.g. a **`RooMultiVarGaussian`** from a **`RooFitResult`** (or one you construct yourself)

```
w.import(*fitRes->createHessePdf(w::mean,w::sigma),”parampdf”) ;
w.factory(“PROD::gprime(f,parampdf)”) ;
```

# RooFit Summary

- Overview of RooFit functionality
  - not everything covered
  - not discussed on how it works internally (optimizations, analytical deduction, etc..)
- Capable to handle complex model
  - scale to models with large number of parameters
  - being used for many analysis at LHC
- Workspace:
  - easy model creation using the factory syntax
  - tool for storing and sharing models (analysis combination)

# RooFit Documentation

- Starting point: http://root.cern.ch/drupal/content/roofit
- Users manual (134 pages ~ 1 year old)
- Quick Start Guide (20 pages, recent)
- Link to 84 tutorial macros (also in $ROOTSYS/tutorials/roofit)
- More than 200 slides from *W. Verkerke* documenting all features are available at the *French School of Statistics 2008*
    - http://indico.in2p3.fr/getFile.py/access?contribId=15&resId=0&materialId=slides&confId=750

# Time For Exercises !

Follow the RooFit exercises at the Twiki page:
https://twiki.cern.ch/twiki/bin/view/RooStats/RooStatsTutorialsJune2013

If you have network problem, you can download tar file from the agenda:

- unpack the tar file and open with your browser the page **RooStatsTutorialsJune2013.html**

# RooStats

## Lecture and Tutorials

# Outline

- Introduction to RooFit
  - Basic functionality
  - Model building using the workspace
  - Composite models
- Exercises on RooFit:
  - building and fitting models
- Introduction to RooStats
  - Interval estimation tools (Likelihood/Bayesian)
  - Hypothesis tests
  - Frequentist interval/limit calculator (CLs)
- Exercises on interval/limit estimation and discovery significance (hypothesis test)

# RooStats Project

- Collaborative project to provide and consolidate advanced statistical tools needed by LHC experiments
- Joint contribution from ATLAS, CMS, ROOT and RooFit
  - developments over-sighted by ATLAS and CMS statistics committees
  - initiated from previous code developed in ATLAS and CMS
  - used by both collaborations

# RooStats Goal

- Common framework for statistical calculations
  - work on arbitrary models and datasets
    - factorize modeling from statistical calculations
  - 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 facilitates the combinations of results

# Statistical Applications

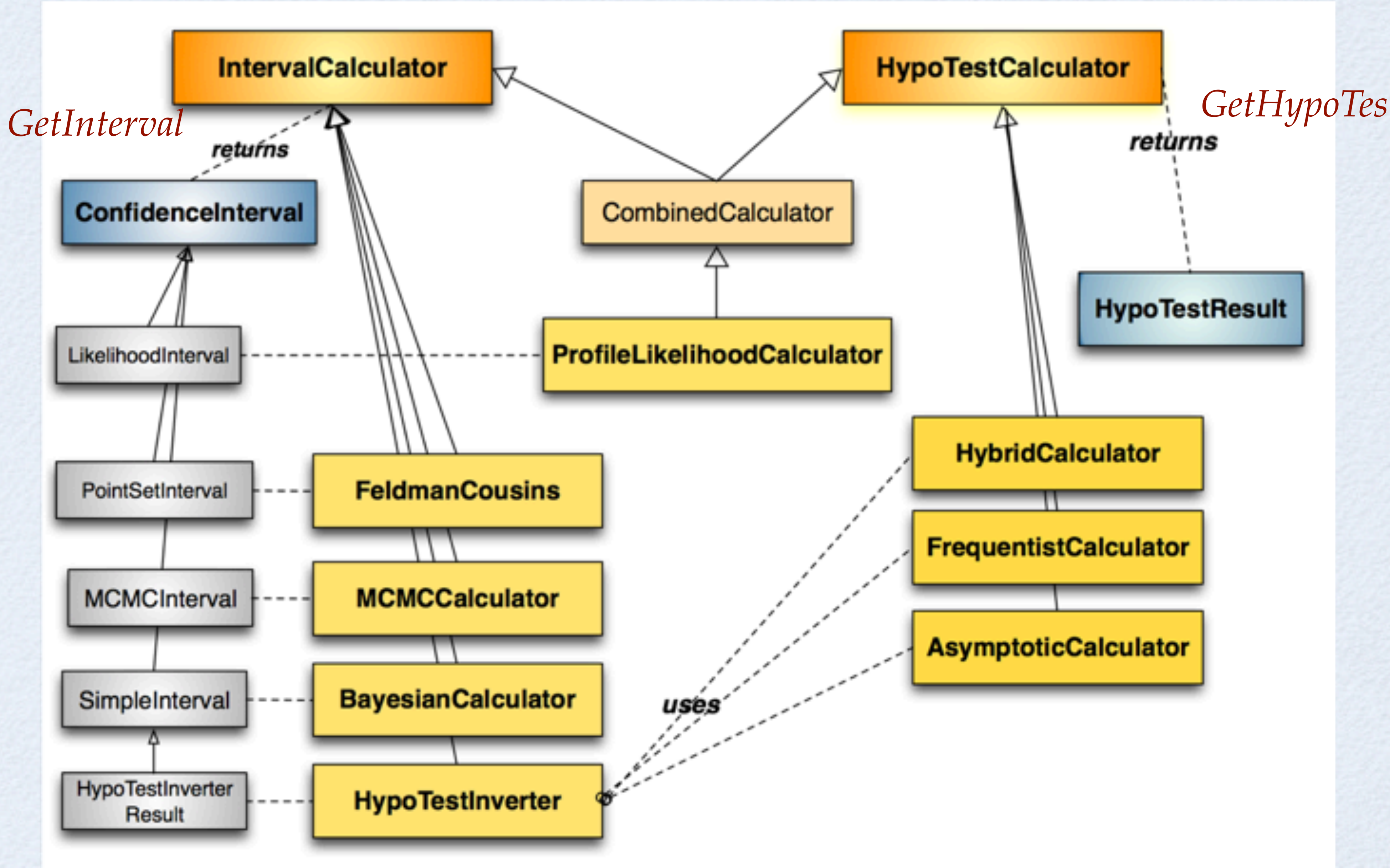
- Statistical problems:
  - point estimation (covered by RooFit)
  - estimation of confidence (credible) intervals
  - hypothesis tests
  - goodness of fit (not yet addressed)

# RooStats Technology

- Built on top of RooFit
  - generic and convenient description of models (probability density function or likelihood functions)
  - provides *workspace* (RooWorkspace)
    - container for model and data and can be written to disk
    - inputs to all RooStats statistical tools
    - convenient for sharing models (e.g digital publishing of results)
  - easily generation of models (workspace factory and HistFactory tool)
  - tools for combinations of model (e.g. simultaneous pdf)
- Use of ROOT core libraries:
  - minimization (e.g. Minuit), numerical integration, etc...
  - additional tools provided when needed (e.g. Markov-Chain MC)

# RooStats Design

- C++ interfaces and classes mapping to real statistical concepts



# RooStats Calculator classes

## Interval Calculators

- ProfileLikelihoodCalculator
  - interval estimation using asymptotic properties of the likelihood function
- BayesianCalculator
  - interval estimation based on Bayes theorem using adaptive numerical integration
- MCMCCalculator
  - Bayesian calculator using Markov-Chain Monte Carlo

- HypoTestInverter
  - invert hypothesis test results to estimate an interval
    - CLs limits, FC interval
- NeymanConstruction and FeldmanCousins
  - frequentist interval calculators

## HypoTest Calculators

- HybridCalculator, FrequentistCalculator
  - frequentist hypothesis test calculators using toy data (difference in treatment of nuisance parameters)
- AsymptoticCalculator
  - hypothesis tests using asymptotic properties of likelihood function

# ModelConfig Class

- `ModelConfig` class input to all Roostats calculators
  - contains a reference to the RooFit workspace class
  - provides the workspace meta information needed to run RooStats calculators
    - pdf of the model stored in the workspace
    - what are observables (needed for toy generations)
    - what are the parameters of interest and the nuisance parameters
    - global observables (from auxiliary measurements) for frequentist calculators
    - prior pdf for the Bayesian tools
  - ModelConfig can be imported in workspace for storage and later retrieval

# Building ModelConfig Class

- ModelConfig must be built after having the workspace
- Identify all the components which are present in the workspace

```
//specify components of model for statistical tools
ModelConfig  modelConfig("G(x|mu,1)");
modelConfig.SetWorkspace(workspace);
//set components using the name of ws objects
modelConfig.SetPdf( "normal");
modelConfig.SetParameterOfInterest("poi");
modelConfig.SetObservables("obs");
```

- Some tools (Bayesian) require to specify prior pdf

```
//Bayesian tools would also need a prior
modelConfig.SetPriorPdf( "prior");
```

- ModelConfig can be imported in workspace to be then stored in a file

```
//can import modelConfig into workspace too
workspace.import(*modelConfig);
```

# Profile Likelihood Calculator

- Method based on properties of the likelihood function
- Profile likelihood function:

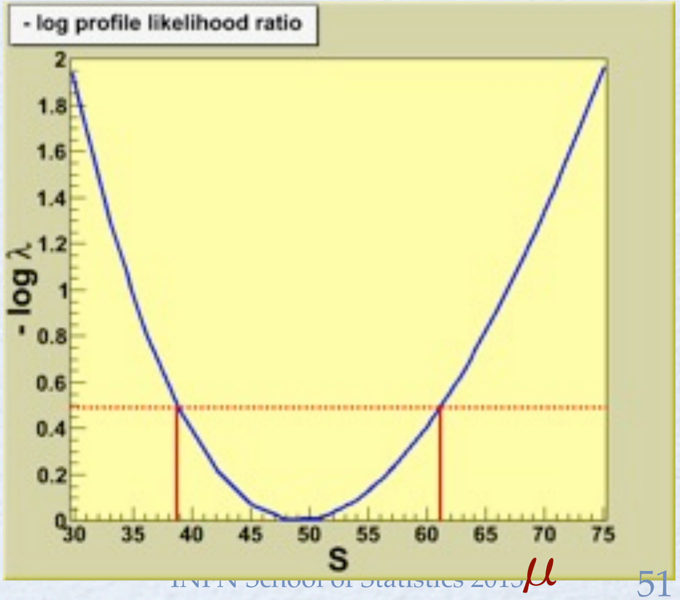
$$\lambda(\mu) = \frac{L(x|\mu, \hat{\hat{\nu}})}{L(x|\hat{\mu}, \hat{\nu})}$$

→ maximize w.r.t nuisance parameters $\nu$ and fix POI $\mu$

→ maximize w.r.t. all parameters

λ is a function of only the parameter of interest $\mu$

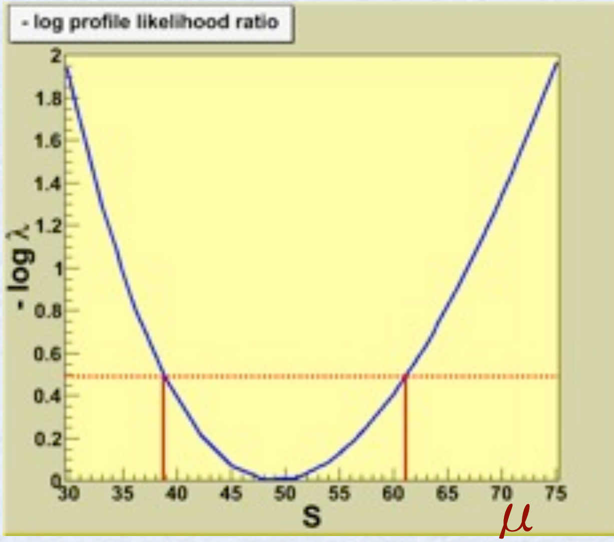
- Uses asymptotic properties of λ based on Wilks' theorem:
- from a Taylor expansion of logλ around the minimum:
  - ➔ -2logλ is a parabola (λ is a gaussian function)
  - ➔ interval on μ from logλ values




- Method of MINUIT/MINOS
  - lower/upper limits for 1D
  - contours for 2 parameters

# Using the Profile Likelihood Calculator

```
// create the class using data and model
ProfileLikelihoodCalculator plc(*data, *model);

// set the confidence level
plc.SetConfidenceLevel(0.683);

// compute the interval
LikelihoodInterval* interval = plc.GetInterval();
double lowerLimit = interval->LowerLimit(*mu);
double upperLimit = interval->UpperLimit(*mu);

// plot the interval
LikelihoodIntervalPlot plot(interval);
plot.Draw();
```




- For one-dimensional intervals:
  - 68% CL (1 σ) interval : $\Delta\log\boldsymbol{\lambda} = 0.5$
  - 95% CL interval : $\Delta\log\boldsymbol{\lambda} = 1.96$
- **`LikelihoodIntervalPlot`** can plot the 2D contours

# Bayesian Analysis in RooStats

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

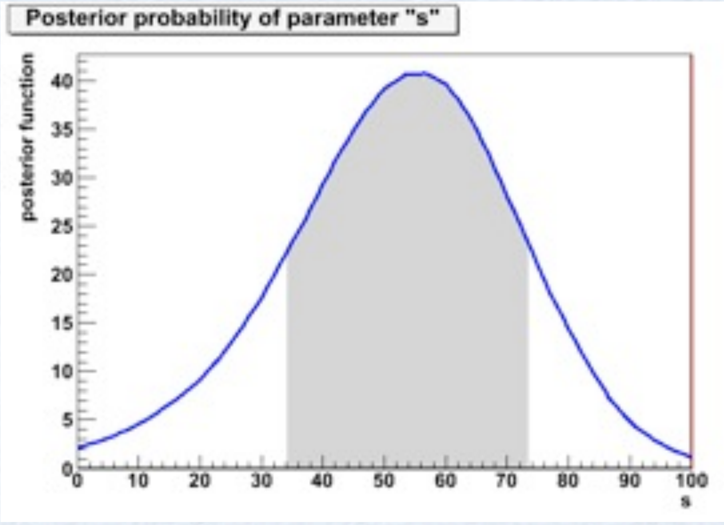
posterior probability

likelihood function — prior probability — nuisance parameters marginalization

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

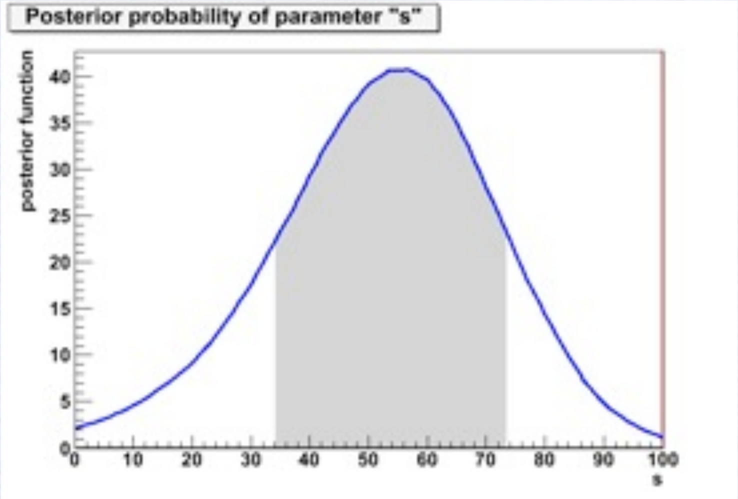
Bayesian Theorem

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



# Bayesian Classes

- **`BayesianCalculator`** class
  - posterior and interval estimation using numerical integration
  - working only for one parameter of interest but can integrate (marginalize) many nuisance parameters
  - support for different integration algorithms, using **`BayesianCalculator::SetIntegrationType`**
    - adaptive numerical (default type), working only for few nuisances (< 10)
    - Monte Carlo integration (PLAIN, MISER, VEGAS)
    - TOYMC : average from toys where the nuisance parameters are sampled from a given p.d.f. (nuisance pdf), but can work in model with many parameters
  - can compute:
    - central interval
    - one-sided interval (upper limit)
    - a shortest interval
  - provide plot of posterior and interval



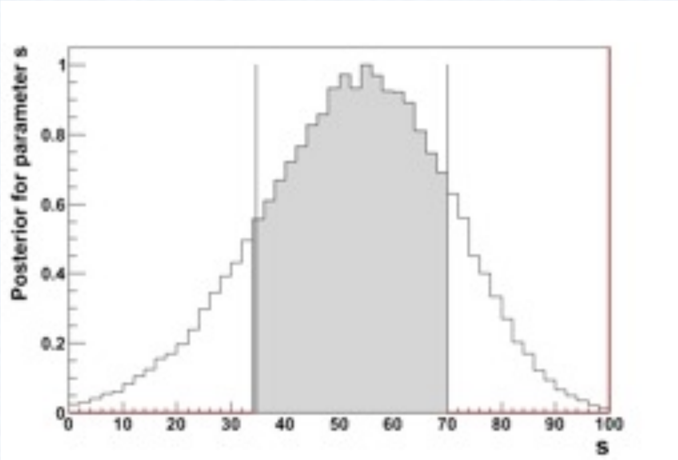

Example: 68% CL central interval

```
BayesianCalculator bc(data, model);
bc.SetConfidenceLevel(0.683);
bc.SetLeftSideTailFraction(0.5);
bc.SetIntegrationType("ADAPTIVE");
SimpleInterval* interval = bc.GetInterval();
double lowerLimit = interval->LowerLimit();
double upperLimit = interval->UpperLimit();
RooPlot * plot = bc.GetPosteriorPlot();
plot->Draw();
```

# MCMC Calculator

- **`MCMCCalculator`** class
  - integration using Markov-Chain Monte Carlo (Metropolis Hastings algorithm)
  - can deal with more than one parameter of interest
  - can work with many nuisance parameters
    - e.g. used in Higgs combination with more than 300 nuisances
  - possible to specify ProposalFunction
    - multivariate Gaussian from fit result
    - Sequential proposal
  - can visualize posterior and also the chain result

MCMCCalculator




```
MCMCCalculator mc(data, model);
mc.SetConfidenceLevel(0.683);
mc.SetLeftSideTailFraction(0.5);
SequentialProposal sp(0.1);
mc.SetProposalFunction(sp);
mc.SetNumIters(1000000);
mc.SetNumBurnInSteps(50);
MCInterval* interval = bc.GetInterval();
RooRealVar * s = (RooRealVar*)
model.GetParametersOfInterest()->find("s");
double lowerLimit = interval->LowerLimit(*s);
double upperLimit = interval->UpperLimit(*s);
MCMCIntervalPlot plot(*interval);
```

# Running RooStats

- RooStats provides standard tutorials taking all as input workspace, ModelConfig and data set names

- StandardProfileLikelihoodDemo.C

*run ProfileLikelihoodCalculator - get interval and produce plot*

```
root[]StandardProfileLikelihoodDemo("ws.root","w","ModelConfig","data")
```

- StandardBayesianNumericalDemo.C

*run Bayesiancalculator: get a credible interval and produce plot of posterior function*

```
root[]StandardBayesianNumericalDemo("ws.root","w","ModelConfig","data")
```

- StandardBayesianMCMCDemo.C

*run bayesian MCMCCalculator: get a credible interval and produce plot of posterior function*

```
root[]StandardBayesianMCMCDemo("ws.root","w","ModelConfig","data")
```

# RooStats
## Part2

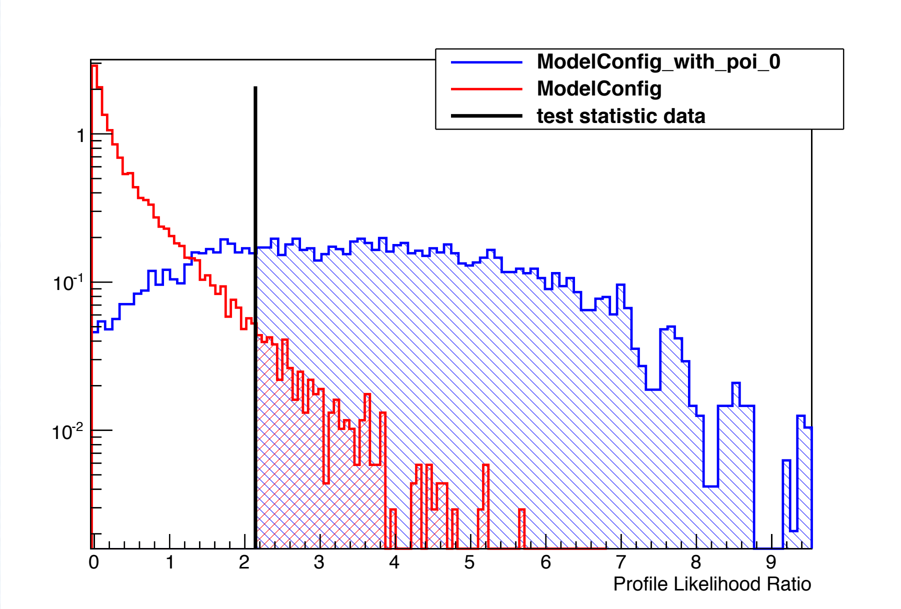
- Hypothesis tests in RooStats using toys and asymptotic formulae
- Hypothesis test inversion
  - Limit and interval calculators
    - CLs, Feldman-Cousins

# Frequentist Hypothesis Tests

- Ingredients:
  - Null Hypothesis: the hypothesis being tested (e.g. $\theta = \theta_0$ ), assumed to be true and one tries to reject it
  - Alternate Hypothesis: the competitive hypothesis (e.g. $\theta \neq \theta_0$ )
  - w is the critical region, a subspace of all possible data:
    - size of test : $\alpha = P( X \in w \mid H_0 )$
    - power of test : $1- \beta = P( X \in w \mid H_1 )$
  - Test statistics: a function of the data, t(X) ,used for defining the critical region in multidimensional data: $X \in w \rightarrow t(X) \in w_t$

# RooStats Hypothesis Test

- Define null and alternate model using ModelConfig
  - can use ModelConfig::SetSnapshot(const RooArgSet &) to define parameter values for the null in case of a common model (e.g. $\mu = 0$ for the B model)
- Select test statistics to use
- Select calculator
  - Use toys or asymptotic formula to get sampling distribution of test statistics
  - FrequentistCalculator or HybridCalculator have different treatment of nuisance parameters



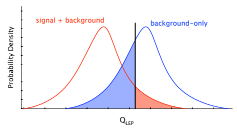
# Test Statistics

- Test statistics maps multidimensional space in one, in a way relevant to the hypothesis being tested

RooStats has the three common test statistics used in the field (and more)

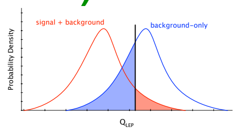
- simple likelihood ratio (used at LEP, nuisance parameters fixed)

$$Q_{LEP} = L_{s+b}(\mu = 1)/L_b(\mu = 0)$$



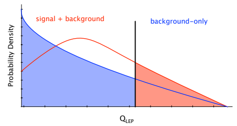

- ratio of profiled likelihoods (used commonly at Tevatron)

$$Q_{TEV} = L_{s+b}(\mu = 1, \hat{\hat{\nu}})/L_b(\mu = 0, \hat{\hat{\nu}}')$$




- profile likelihood ratio (related to Wilks's theorem)

$$\lambda(\mu) = L_{s+b}(\mu, \hat{\hat{\nu}})/L_{s+b}(\hat{\mu}, \hat{\nu})$$




- preferred choice is profile likelihood ratio which has known asymptotic distribution

# FrequentistCalculator

- Generate toys using nuisance parameter at their conditional ML estimate ( $\theta = \hat{\theta}_\mu$) by fitting them to the observed data
- Treat constraint terms in the likelihood (e.g. systematic errors) as auxiliary measurements
  - introduce global observables which will be varied (tossed) for each pseudo-experiment
  - $L = \text{Poisson}(\, n_{obs} \mid \mu + b)\ \text{Gaussian}(\, b_0 \mid b, \sigma_b)$
    - $b_0$ is a global observables, varied for each toys but it needs to be considered constant when fitting
    - $n_{obs}$ is the observable which is part of the data set
    - $\mu$ is the parameter of interest (poi)
    - b is the nuisance parameter

# HybridCalculator

- Nuisance parameters are integrated using their pdf (the constraint term) which is interpreted as a Bayesian prior
  - integration is done by generating for each toys different nuisance parameters values
  - need to have a pdf for the nuisance parameters (often it can be derived automatically from the model)

$$L = \text{Poisson}(\, n_{obs} \mid \mu + b)\ \text{Gaussian}(\, b \mid b_0, \sigma_b)$$

$$\Downarrow$$

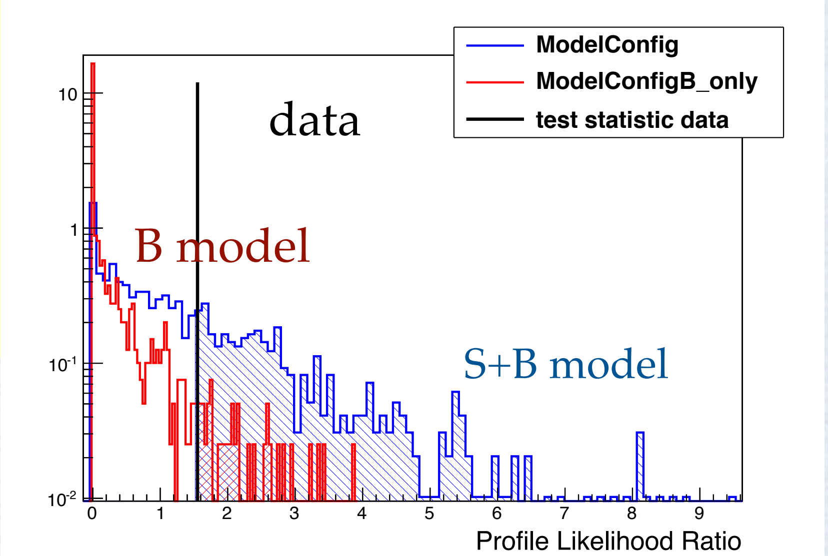
$$L = \int \text{Poisson}(\, n_{obs} \mid \mu + b)\ \text{Gaussian}(\, b \mid b_0, \sigma_b)\, db$$

# Example: FrequentistCalculator

- Define the models
  - N.B for discovery significance null is B model and alt is S

```
// create first HypoTest calculator (data, alt model , null model)
FrequentistCalculator fcalc(*data, *sbModel, *bModel);

// create the test statistics
ProfileLikelihoodTestStat profll(*sbModel->GetPdf());
// use one-sided profile likelihood for discovery tests
profll.SetOneSidedDiscovery(true);

// configure  ToyMCSampler and set the test statistics
ToyMCSampler *toymcs = (ToyMCSampler*)fcalc.GetTestStatSampler();
toymcs->SetTestStatistic(&profll);

fcalc.SetToys(1000,1000);  // set number of toys for (null, alt)

// run the test
HypoTestResult * r = fcalc.GetHypoTest();
r->Print();

// plot test statistic distributions
HypoTestPlot * plot = new HypoTestPlot(*r);
plot->Draw();
```

```
Results HypoTestCalculator_result:
 - Null p-value = 0.034 +/- 0.00573097
 - Significance = 1.82501 sigma
 - Number of Alt toys: 1000
 - Number of Null toys: 1000
```

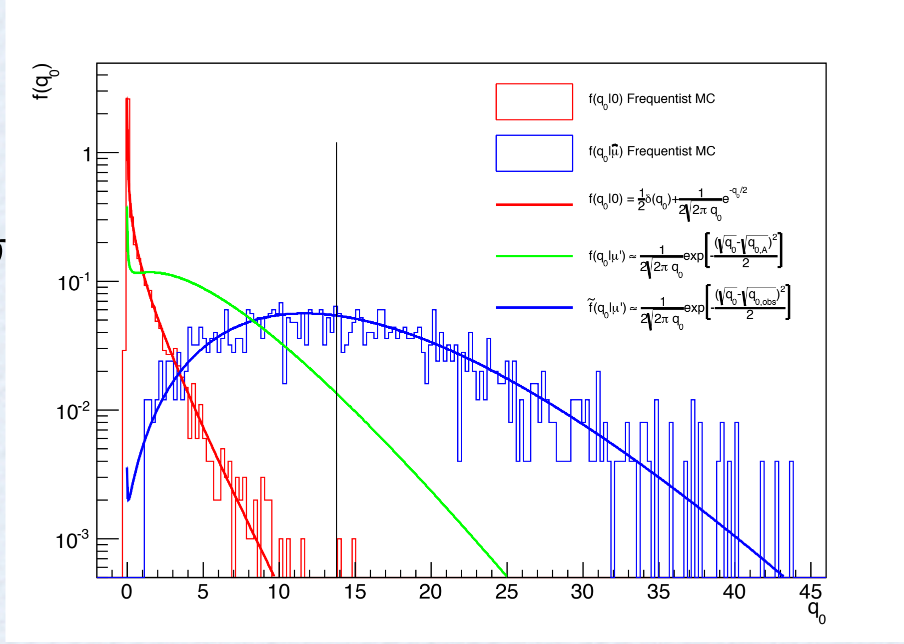


# AsymptoticCalculator

- Use the asymptotic formula for the test statistic distributions
- one-sided profile likelihood test statistic:
  - null model ($\mu = \mu_{TEST}$ )
    - half $X^2$ distribution
  - alt model ($\mu \neq \mu_{TEST}$ )
    - non-central $X^2$
    - use Asimov data to get the non centrality parameter $\Lambda = (\mu-\mu_{TEST})/\sigma$
- p-values for null and alternate can be obtained without generating toys

$$\lambda(\mu) = \frac{L(x|\mu,\hat{\hat{\nu}})}{L(x|\hat{\mu},\hat{\nu})}$$

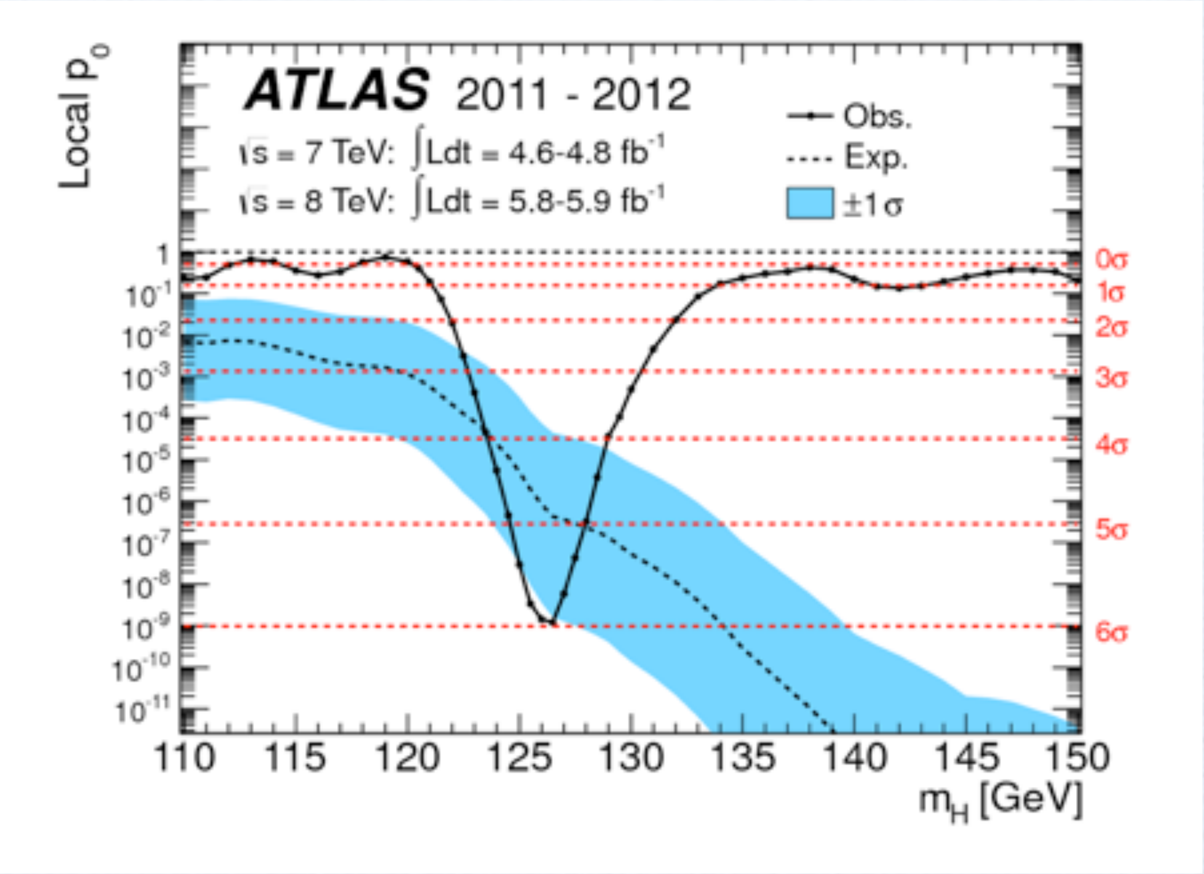
$\lambda(\mu) = 0$ for
$\hat{\mu} < 0$ (discovery)
$\hat{\mu} < \mu_{TEST}$ (limits)




➡ see Cowan, Cranmer, Gross, Vitells, arXiv:1007.1727,EPJC 71 (2011) 1-1

# Example: Discovery Significance

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



# Inversion of Hypothesis Tests

- one-to-one mapping between hypothesis tests and confidence intervals

**Table 20.1 Relationships between hypothesis testing and interval estimation**

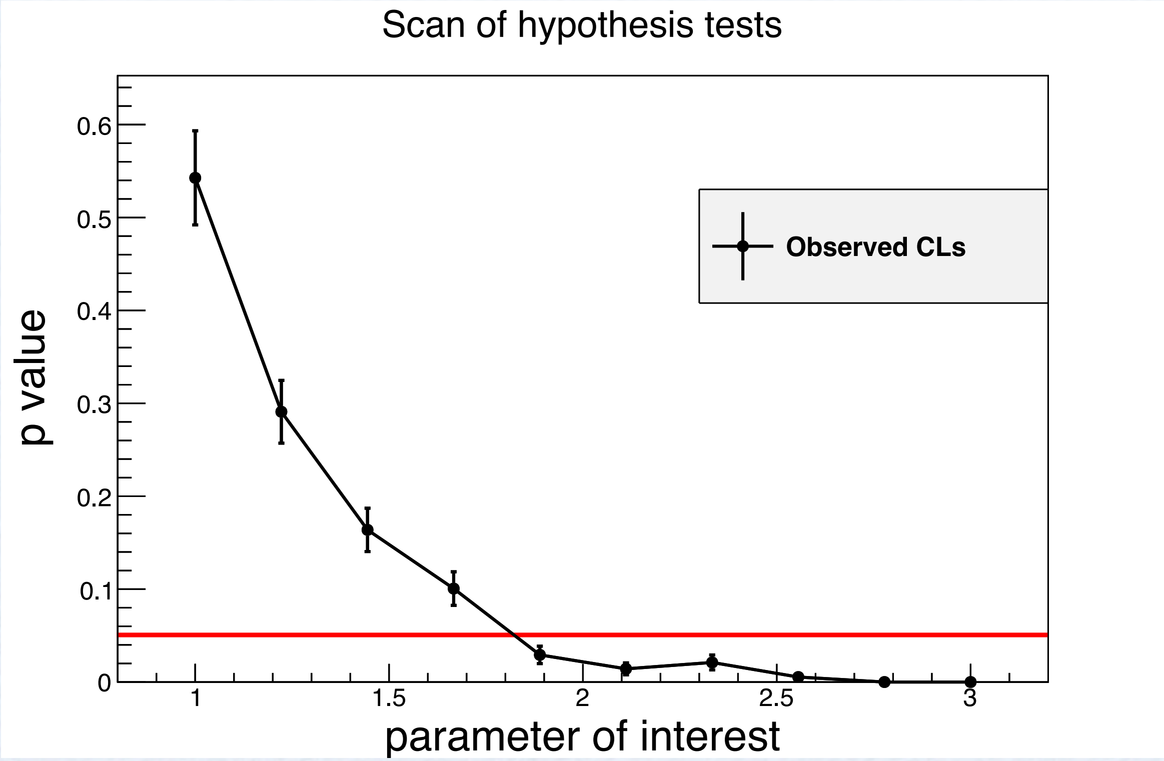
| Property of test | | Property of corresponding confidence interval |
|---|---|---|
| Size $= \alpha$ | | Confidence coefficient $= 1 - \alpha$ |
| Power $=$ probability of rejecting a false value of $\theta = 1 - \beta$ | | Probability of not covering a false value of $\theta = 1 - \beta$ |
| Most powerful | | Uniformly most accurate |
| $\longleftarrow$ | $\left\{ \begin{array}{c} \textit{Unbiased} \\ 1 - \beta \geq \alpha \end{array} \right\}$ | $\longrightarrow$ |
| Equal-tails test $\alpha_1 = \alpha_2 = \frac{1}{2}\alpha$ | | Central interval |

*from G. Feldman visiting Harvard statistics department*

They explained that in statistical theory there is a one-to-one correspondence between a hypothesis test and a confidence interval. (The confidence interval is a hypothesis test for each value in the interval.) The Neyman-Pearson Theorem states that the likelihood ratio gives the most powerful hypothesis test. Therefore, it must be the standard method of constructing a confidence interval.

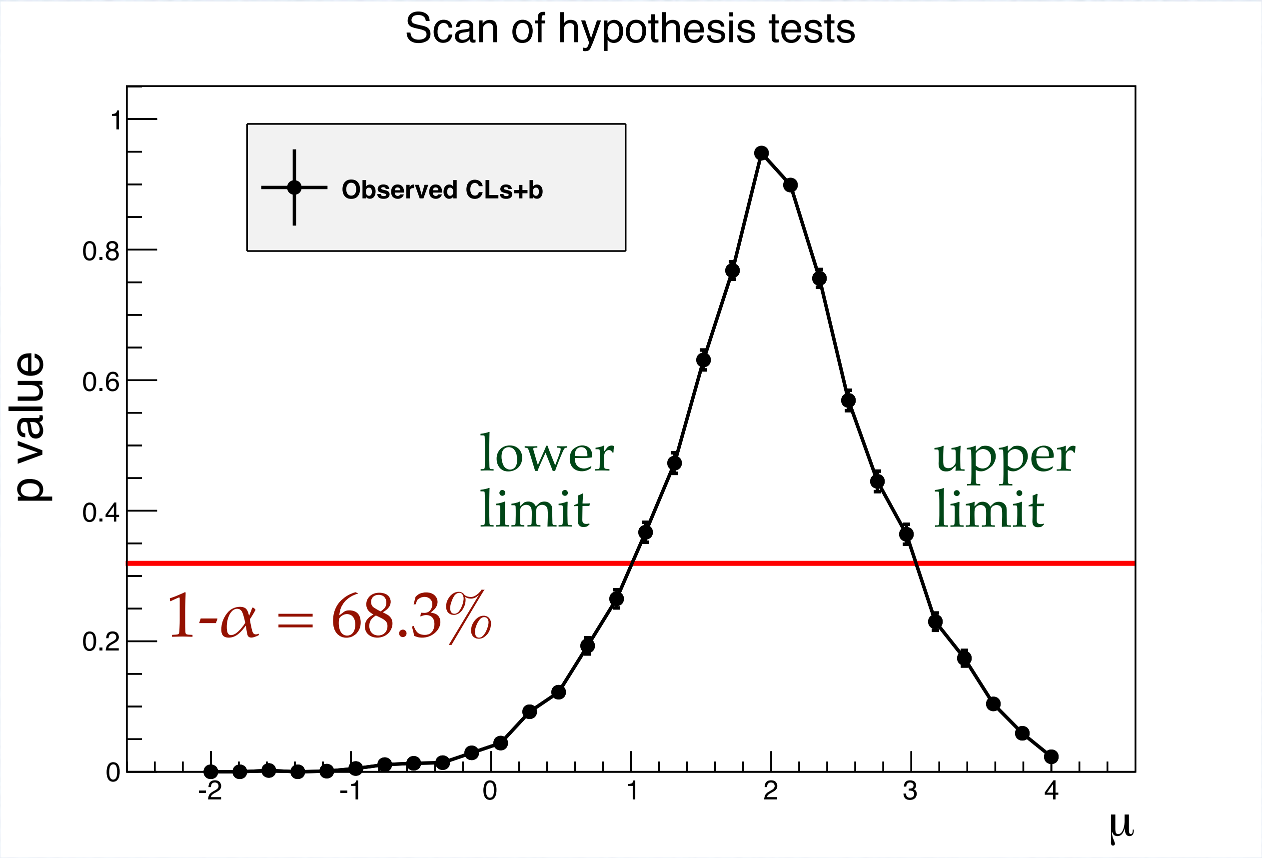
# Hypothesis Test Inversion

- Performing an hypothesis test at each value of the parameter
- Interval can be derived by inverting the p-value curve, function of the parameter of interest (μ)
  - value of μ which has p-value $\alpha$ (e.g. 0.05), is the upper limit of 1-$\alpha$ confidence interval (e.g. 95%)



# Hypothesis Test Inversion

- use one-sided test for upper limits (e.g. one-side profile likelihood test statistics)
- use two-sided test for a 2-sided interval




Example: 1-σ interval for a Gaussian measurement

# HypoTestInverter class

- Input is an Hypothesis Test calculator:
  - Frequentist/Hybrid/AsymptoticCalculator
  - possible to customize test statistic, number of toys, etc..
    - N.B: null model is S+B, alternate is B only model
- Compute an Interval (result is a **`ConfInterval`** object):
  - scan given interval of $\mu$ and perform hypothesis tests
  - compute upper/lower limit from scan result
    - can use $CL_s = CL_{s+b} / CL_b$ for the p-value
  - result (**`HypoTestInverterResult`**) contains all the hypothesis test results for each scanned $\mu$ value
  - can compute expected limits and bands

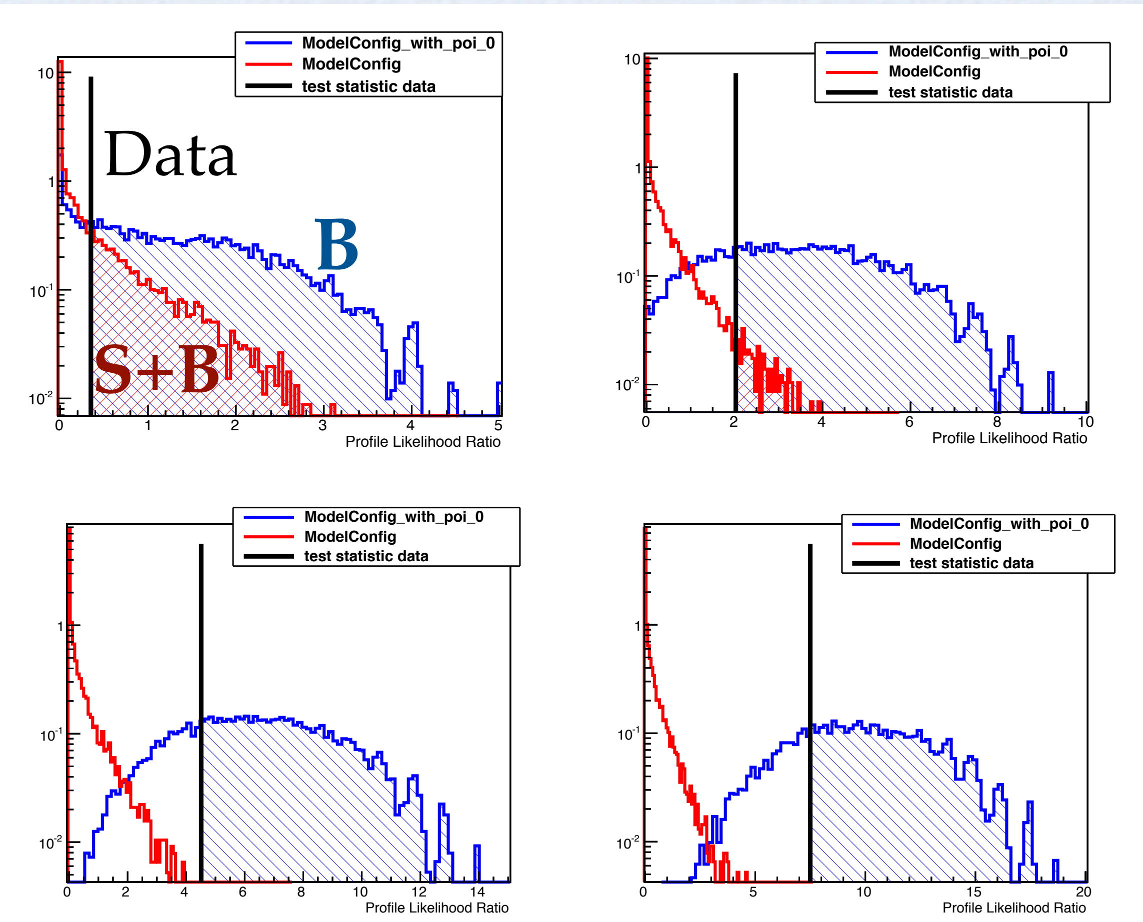
# HypoTestInverter

- **HypoTestInverter** class in RooStats

```
// create first HypoTest calculator (N.B null is s+b model)
FrequentistCalculator fc(*data, *bModel, *sbModel);

HypoTestInverter calc(*fc);
calc.UseCLs(true);

// configure  ToyMCSampler and set the test statistics
ToyMCSampler *toymcs = (ToyMCSampler*)fc.GetTestStatSampler();

ProfileLikelihoodTestStat profll(*sbModel->GetPdf());
// for CLs (bounded intervals) use one-sided profile likelihood
profll.SetOneSided(true);
toymcs->SetTestStatistic(&profll);

// configure and run the scan
calc.SetFixedScan(npoints,poimin,poimax);
HypoTestInverterResult * r = calc.GetInterval();

// get result and plot it
double upperLimit = r->UpperLimit();
double expectedLimit = r->GetExpectedUpperLimit(0);

HypoTestInverterPlot *plot = new HypoTestInverterPlot("hi","",r);
plot->Draw();
```

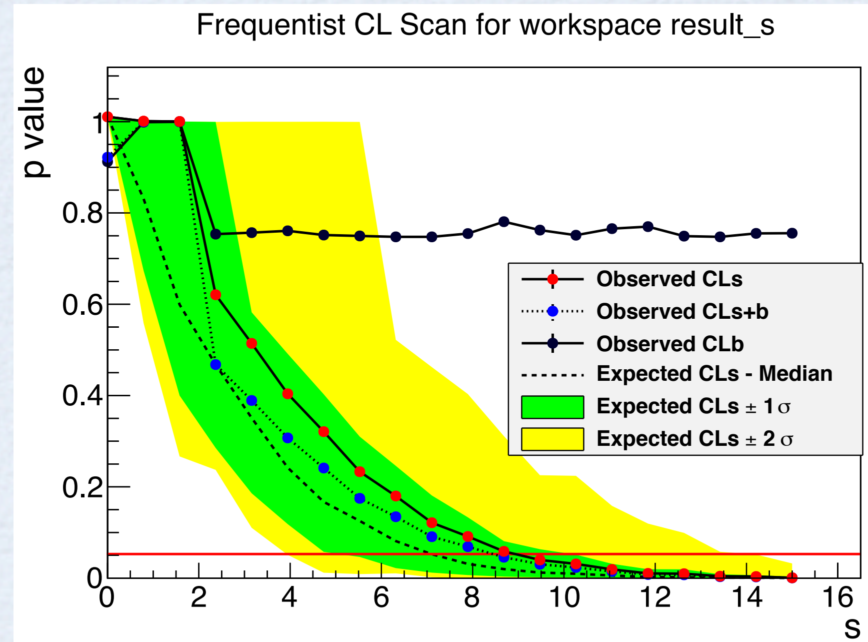
# Running the HypoTestInverter

Hypothesis test results for each scanned point




p-value, $CL_{s+b}$ (or $CL_b$) is integral of S+B (or B) test statistic distribution from data value

Scan result



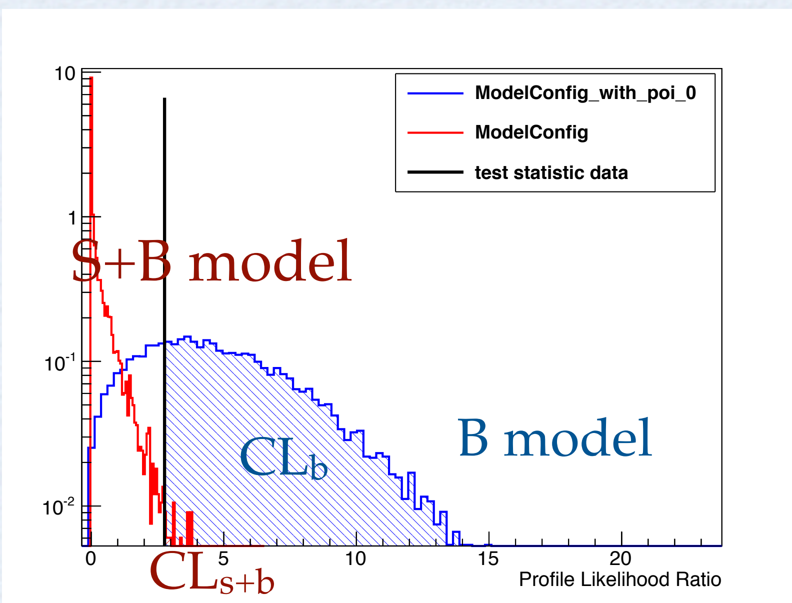

How expected limit and bands are obtained ?

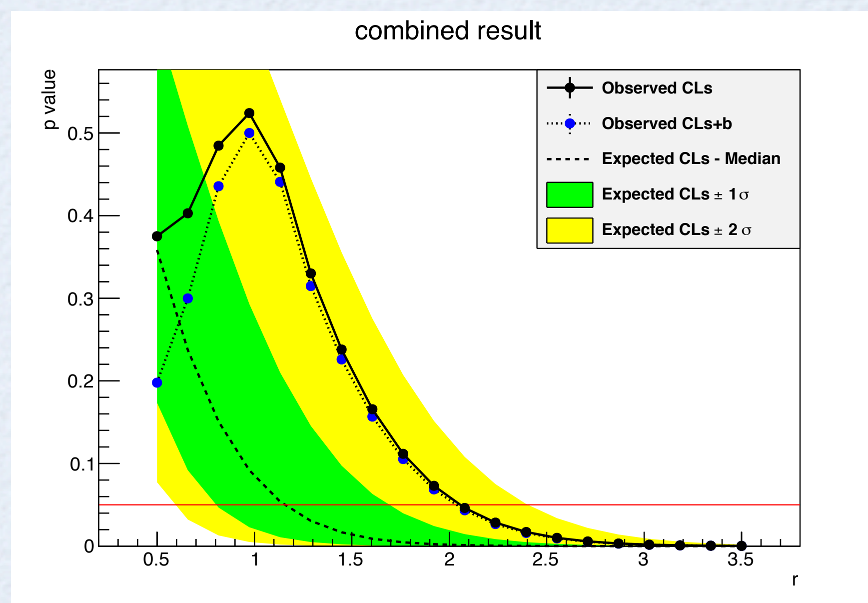
- compute p-value for quantiles (median, +/1,2 sigma) of the B model test statistic distribution (*i.e.* use quantile as the observed value)

# Asymptotic Limits

- **AsymptoticCalculator** class for HypoTestInverter
  - use the asymptotic formula for the test statistic distributions
    - $\chi^2$ approximation for the profile likelihood ratio
      - see G. Cowan *et al.*, arXiv:1007.1727,EPJC 71 (2011) 1-1
  - p-values $CL_{s+b}$ (null) and $CL_b$ (alt) obtained without generating toys
  - also expected limits from the alt distribution

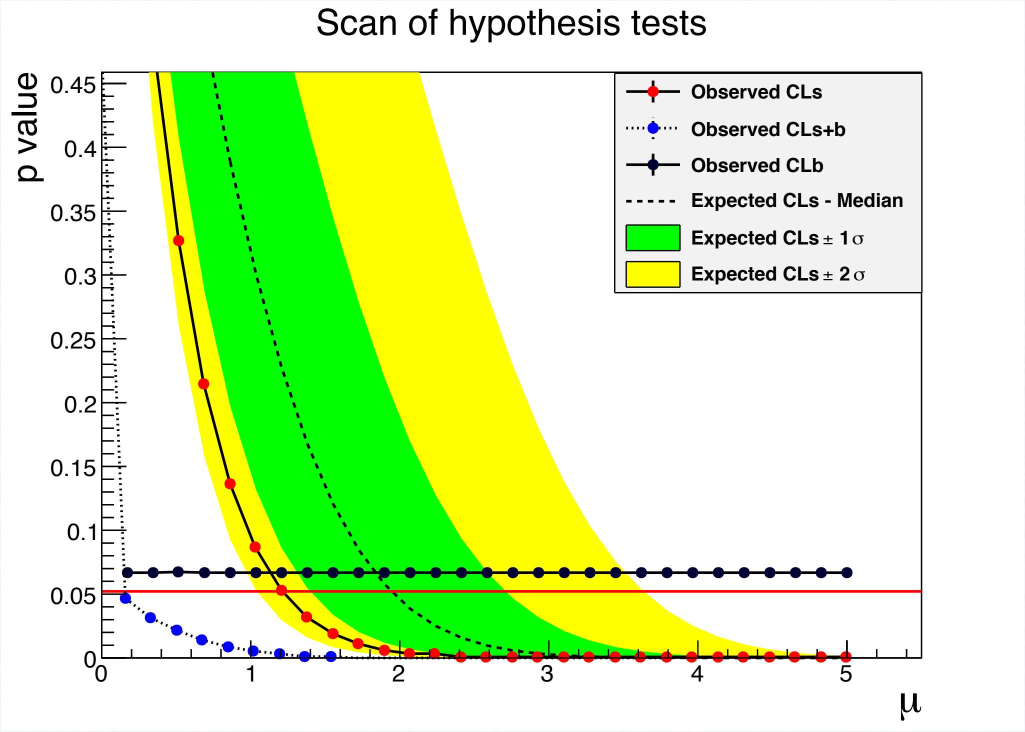
```
// create first HypoTest calculator (N.B null is s+b model)
AsymptoticCalculator ac(*data, *bModel, *sbModel);

HypoTestInverter calc(*ac);
// run inverter same as using other calculators
........
```

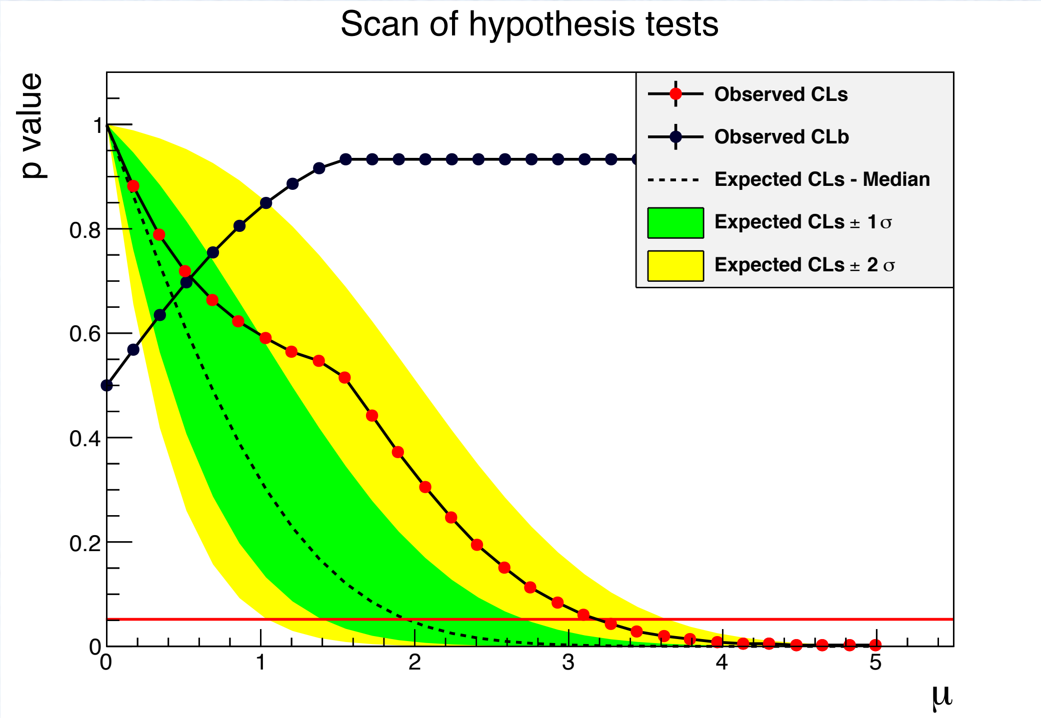


# Example of Scan

- 95% CL limit on a Gaussian measurement:
  - Gauss(x,μ,1), with μ≥0



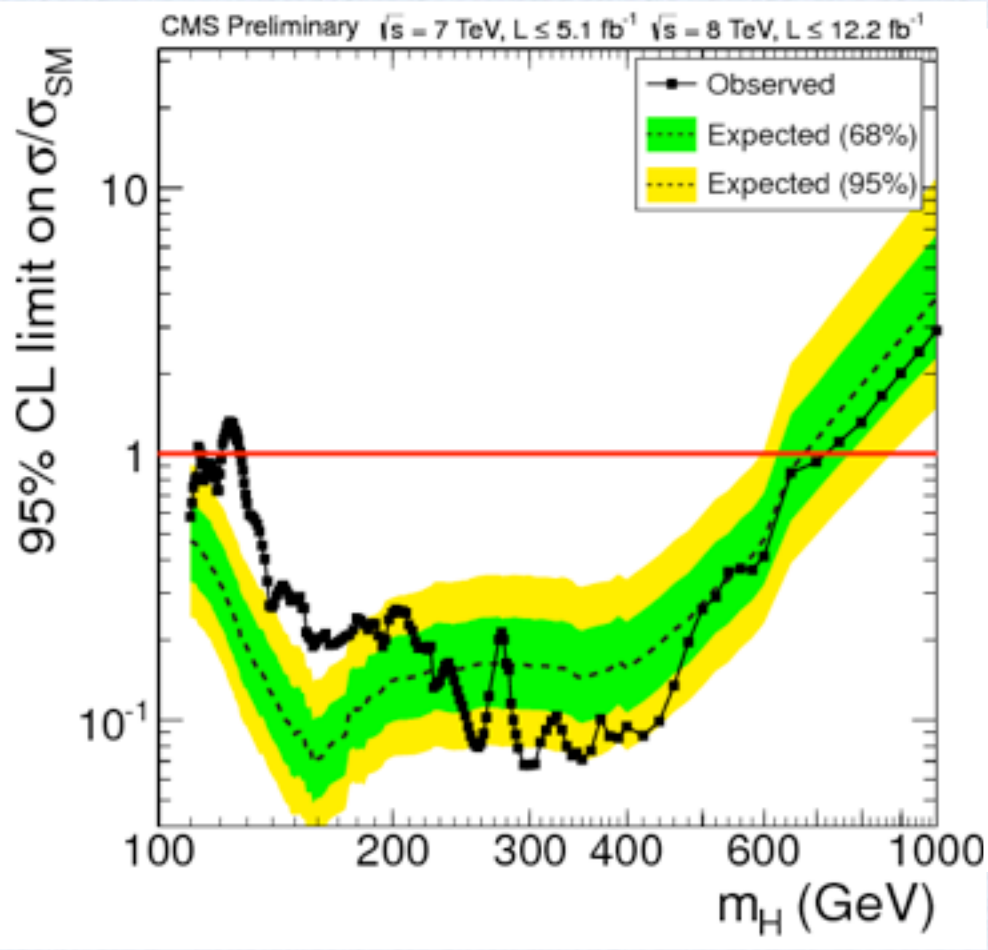

deficit, observation x = -1.5

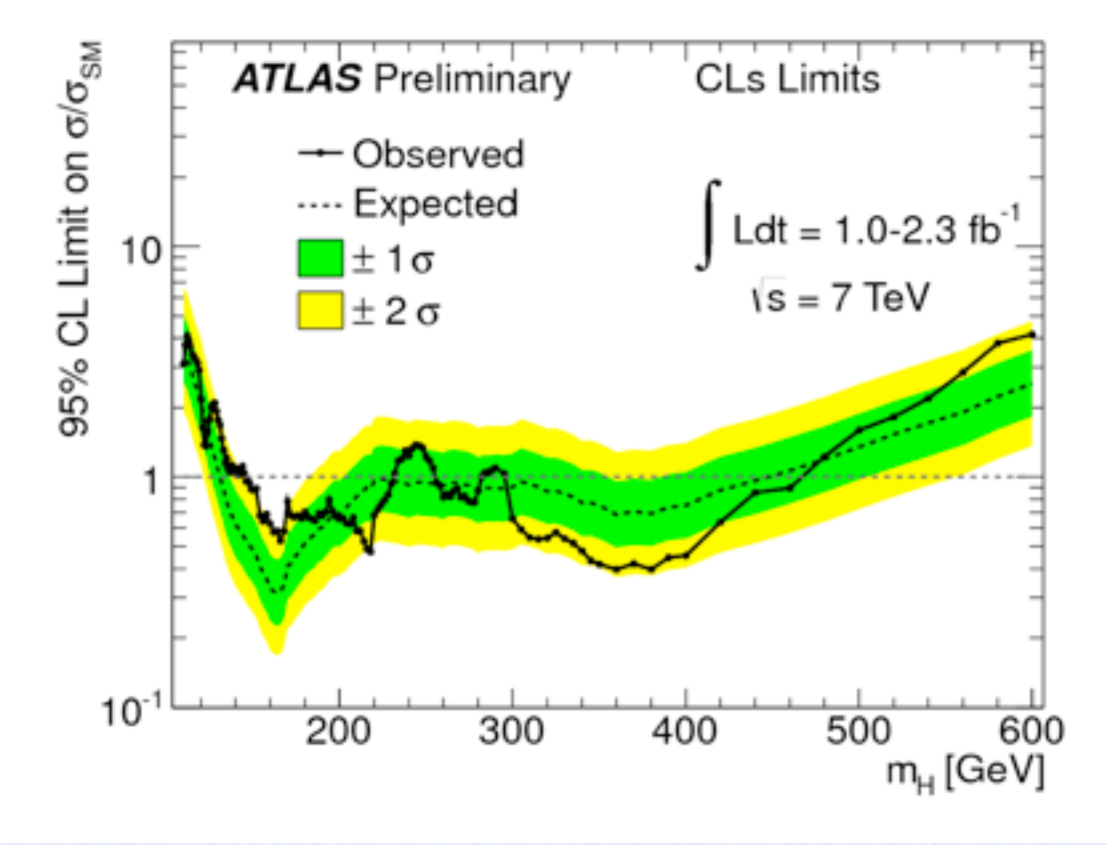
excess, observation x = 1.5

use $CL_s$ as p-value to avoid setting limits which are too good

# Example: Computing Limits

- By computing limits for different mass hypothesis:



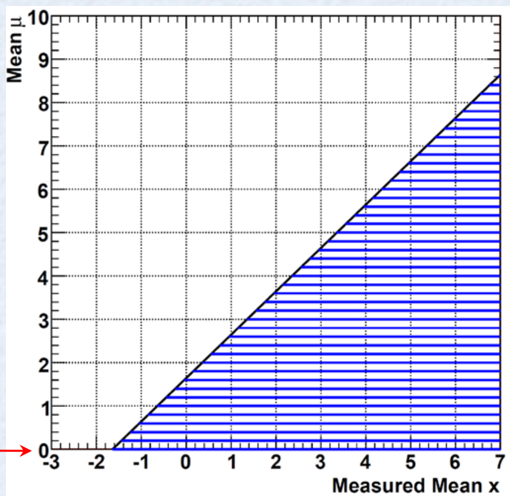
# Limits on bounded measurements

*from Bob Cousins:*

## Downward fluctuations in searches for excesses

**Classic example: Upper limit on mean $\mu$ of Gaussian based on measurement *x* (in units of $\sigma$).**

$\mu$=0

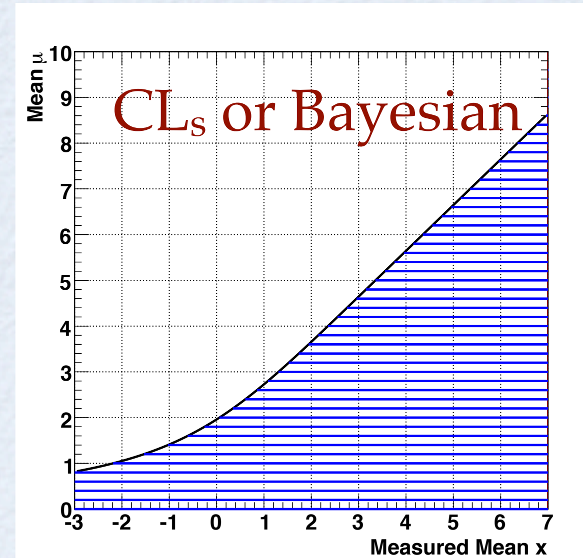


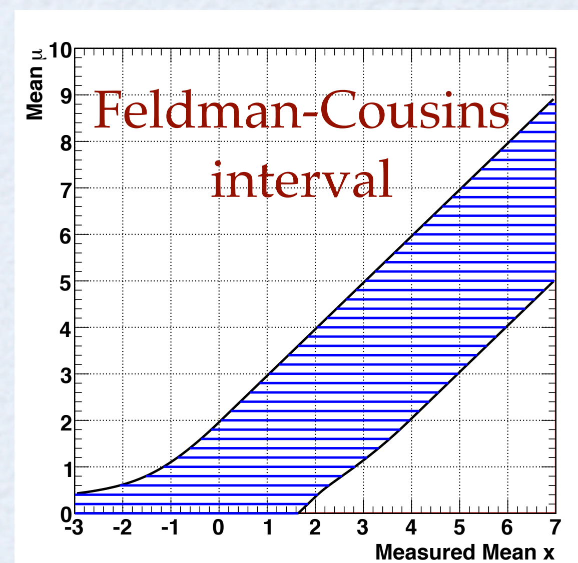

**Frequentist 1-sided 95% C.L. Upper Limits, based on $\alpha$ = 1 – C.L. = 5% (called $CL_{sb}$ at LEP).**

**For *x* < –1.64 $\sigma$ the confidence interval is the *null* set!**

**If $\mu \geq 0$ in model, as measured x becomes increasingly negative, standard classical upper limit becomes small and then null.**

**Issue acute 15-25 years ago in expts to measure $\nu_e$ mass in (tritium $\beta$ decay): several measured $m_\nu^2 < 0$.**



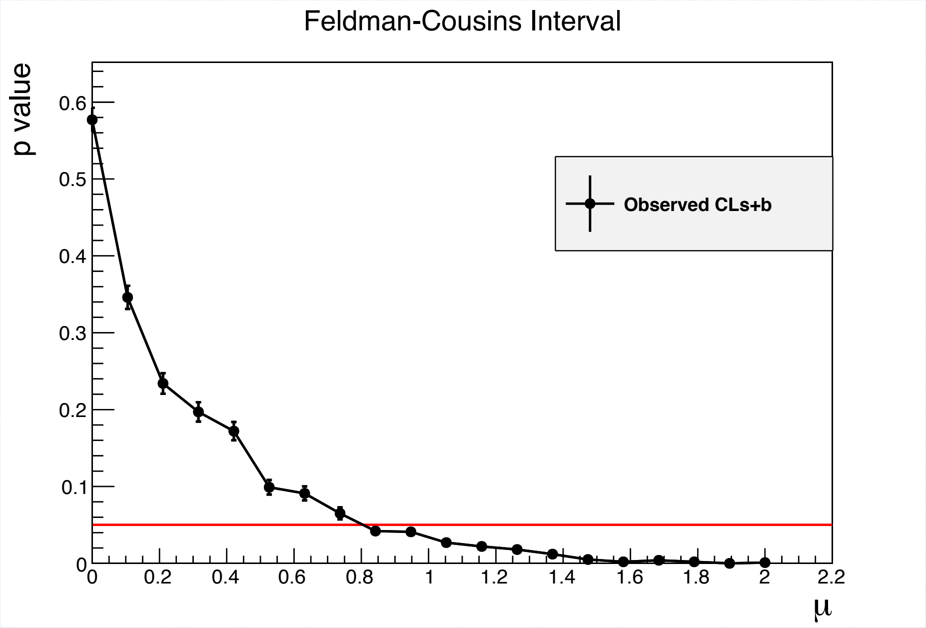




Bob Cousins, CMSDAS, 1/2012

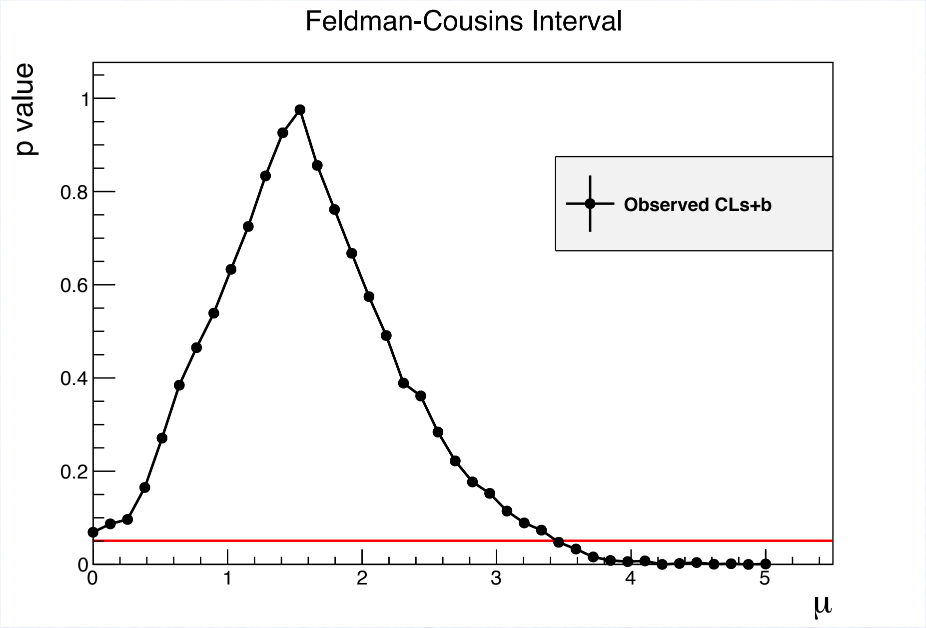
# Feldman-Cousins intervals

- HypoTestInverter class can compute also a Feldman-Cousins interval
  - need to use FrequentistCalculator and $CL_{s+b}$ as p-value
  - use the 2-sided profile likelihood test statistic

$$\lambda(\mu) = \frac{L(x|\mu, \hat{\hat{\nu}})}{L(x|\hat{\mu}, \hat{\nu})}$$




observation x = -1.5




observation x = 1.5

# Feldman-Cousins Interval

*from Kyle Cranmer:*

## A different way to picture Feldman-Cousins

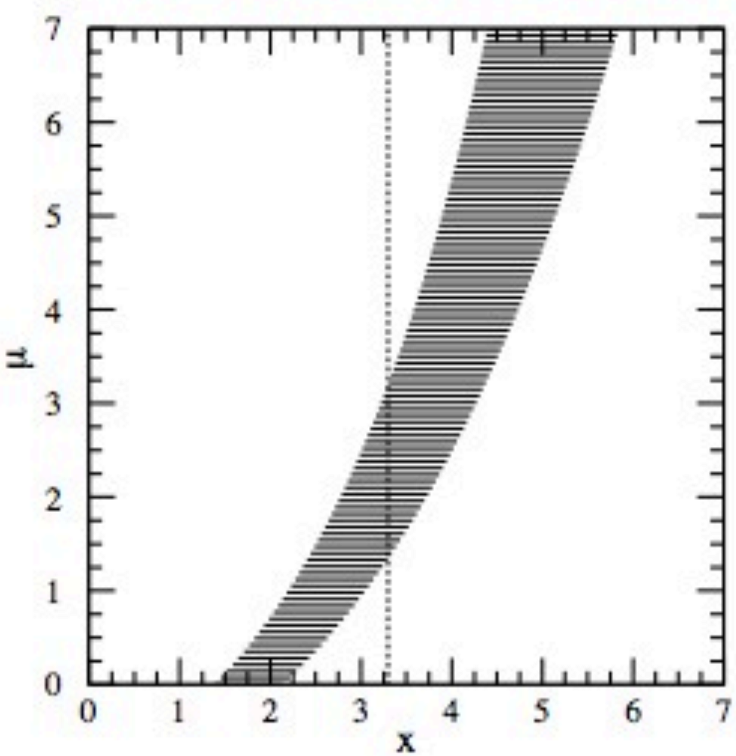
CENTER FOR COSMOLOGY AND PARTICLE PHYSICS

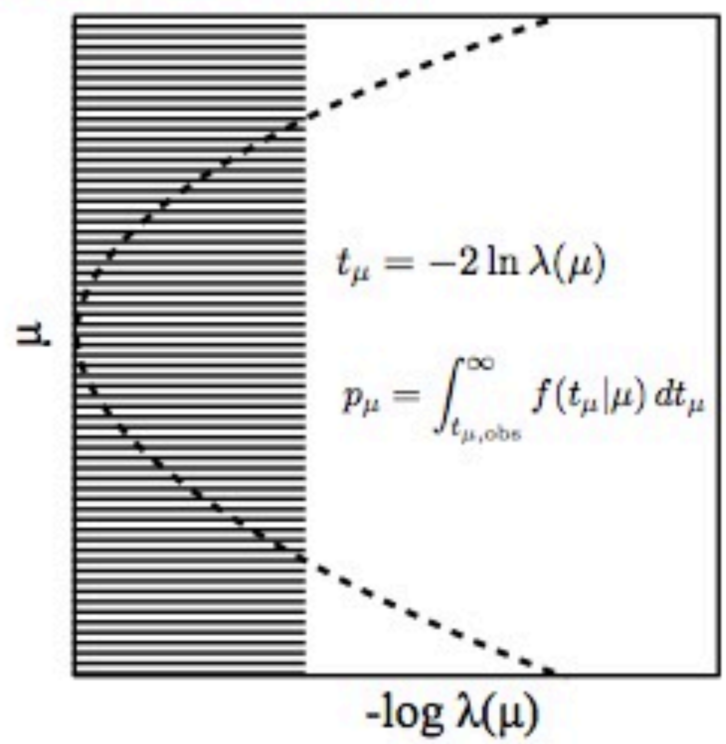
Most people think of plot on left when thinking of Feldman-Cousins

- bars are regions "ordered by" $R = P(n|\mu)/P(n|\mu_{\rm best})$, with $\int_{x_1}^{x_2} P(x|\mu)dx = \alpha.$

But this picture doesn't generalize well to many measured quantities.

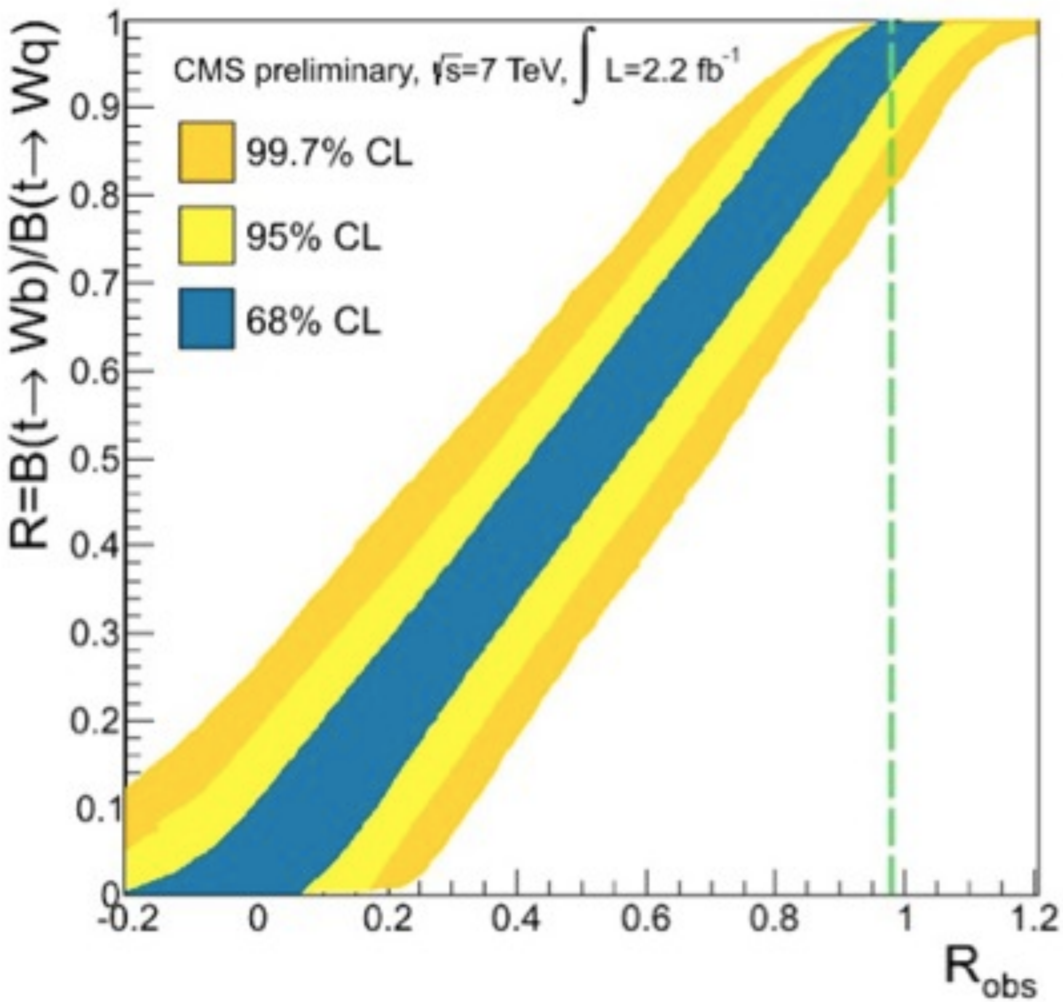
- Instead, just use R as the test statistic... and R is $\lambda(\mu)$



# Example: Feldman-Cousins interval

- Same RooStats code but with different configuration can compute also a Feldman-Cousins interval



# StandardHypoTestInvDemo.C

- Standard ROOT macro to run the Hypothesis Test inversion.
- Inputs to the macro:
  - workspace file, workspace name
  - name of S+B model (null) and for B model (alt)
    - if no B model is given, use S+B model with poi = 0
  - data set name
  - calculator type: frequentist (= 0), hybrid (=1), or asymptotic (=2)
  - test statistics
- options:
  - use $CL_s$ or $CL_{s+b}$ for computing limit
  - number of points to scan and min, max of interval

*load the macro after having created the workspace and saved in file SPlusBExpoModel.root*

```
root[] .L StandardHypoTestInvDemo.C
```

*run for CLs (with frequentist calculator (type = 0) and one-side PL test statistics (type = 3) scan 10 points in [0,100]*

```
root[] StandardHypoTestInvDemo("SPlusBExpoModel.root","w","ModelConfig","","data",0,3, true, 10, 0, 100)
```

*run for Asymptotic CLs (scan 20 points in [0,100])*

```
root[] StandardHypoTestInvDemo(SPlusBExpoModel.root","w","ModelConfig","","data",2,3, true, 20, 0, 100)
```

*run for Feldman-Cousins ( scan 10 points in [0,100])*

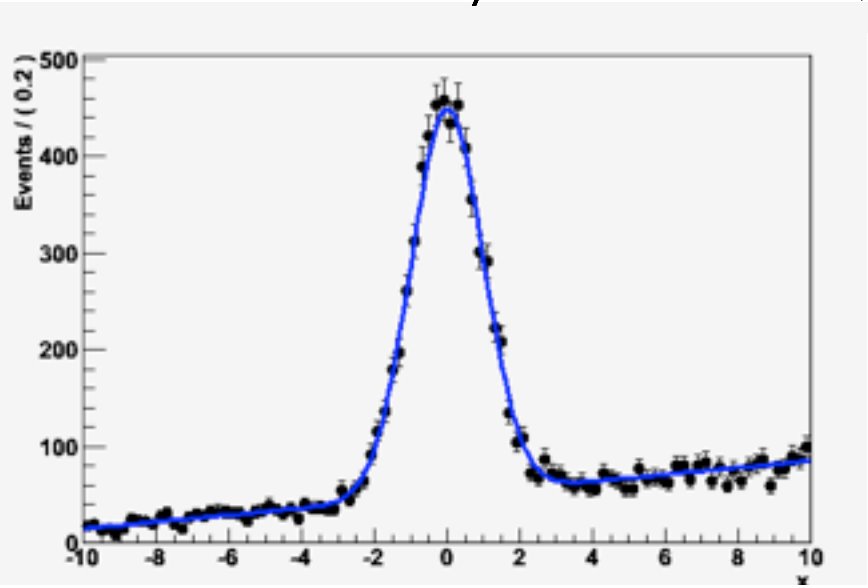
```
root[] StandardHypoTestInvDemo(SPlusBExpoModel.root","w","ModelConfig","","data",0,2, false, 10, 0, 15)
```

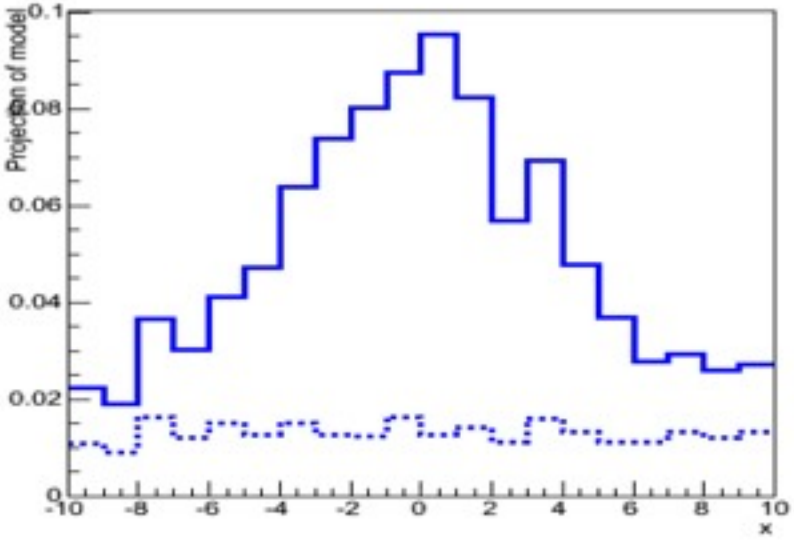
# HistFactory – a new class of pdfs

- Focus of RooFit traditionally on analytical models
  - Assumes you can formulate signal/background in an analytical form
  - Often possible in e+e- experiments,
    shapes for hadron colliders cumbersome → **rely on MC simulation**

**Analytical form:**
Gaussian+Polynomial

**Template form:**
Histogram (discrete)

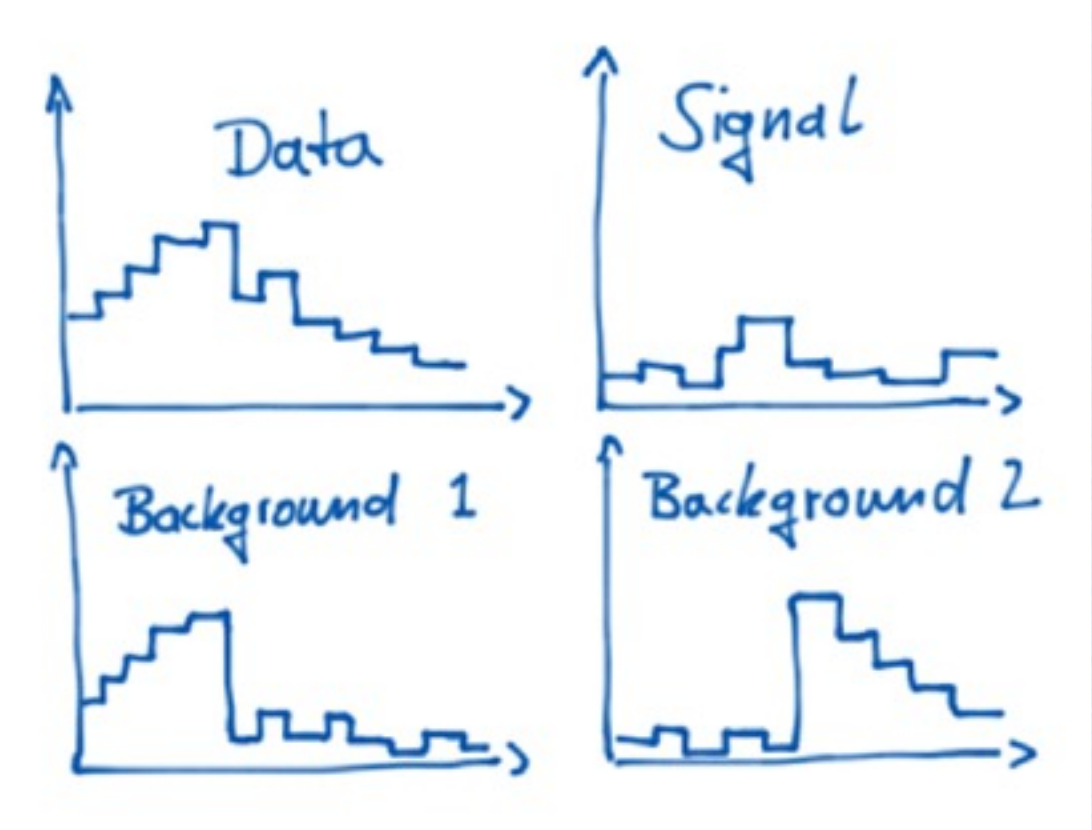


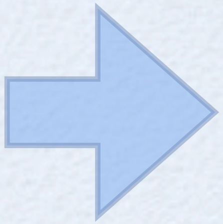




K. Cranmer, G. Lewis, L. Moneta, A. Shibata, and W. Verkerke, *HistFactory: A tool for creating statistical models for use with RooFit and RooStats*, CERN-OPEN-2012-016 (2012). http://cdsweb.cern.ch/record/1456844.

# Model Building with HistFactory

- Tool to build models from input histograms



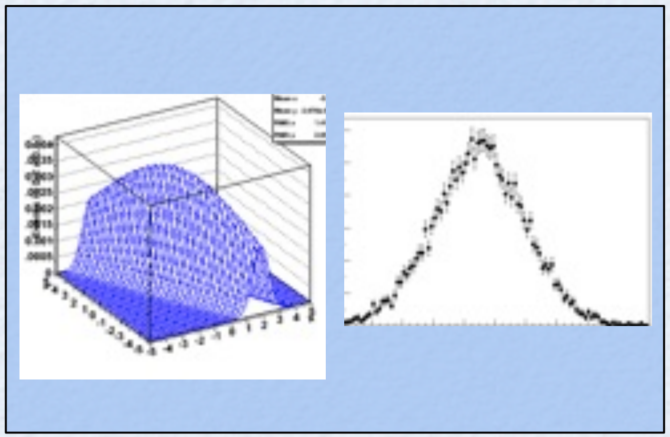



RooFit
Workspace

# RooFit/RooStats at LHC (Higgs analysis)

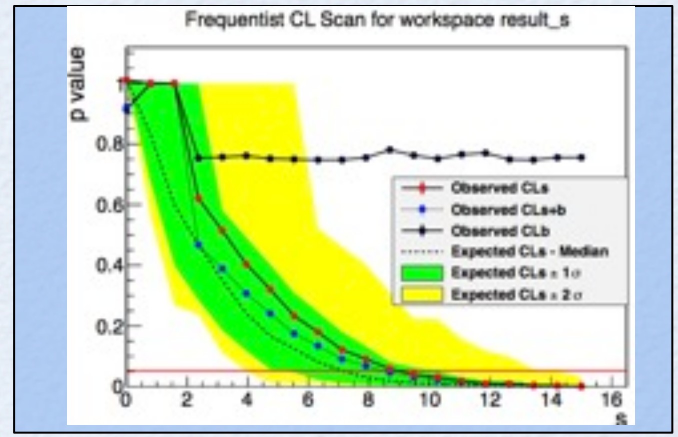
## Class RooWorkspace

*Simplify packaging and sharing of models*



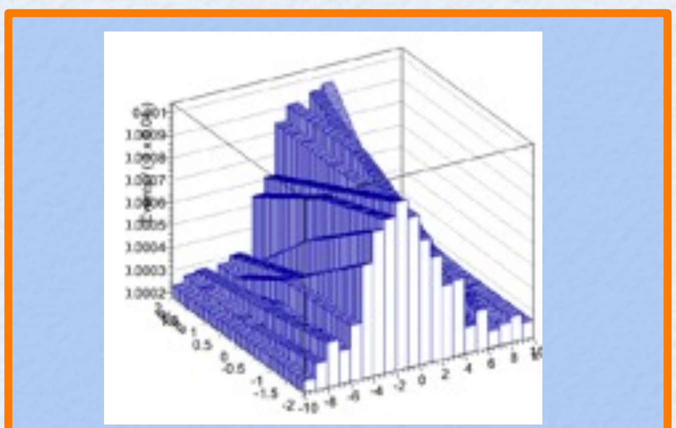
## RooStats toolkit

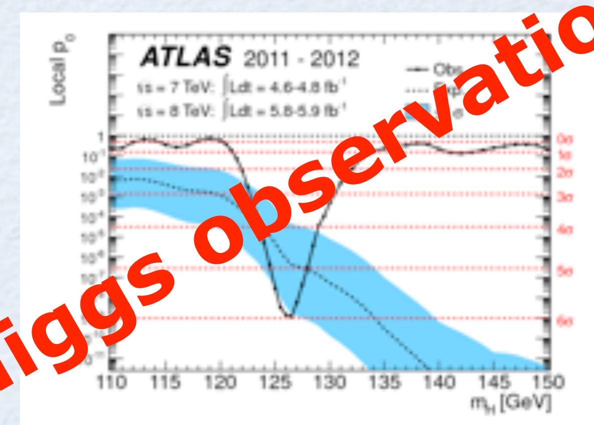
*Statistical tests based on likelihoods from RooFit models*




## HistFactory package

*Constructing models from Monte Carlo templates*

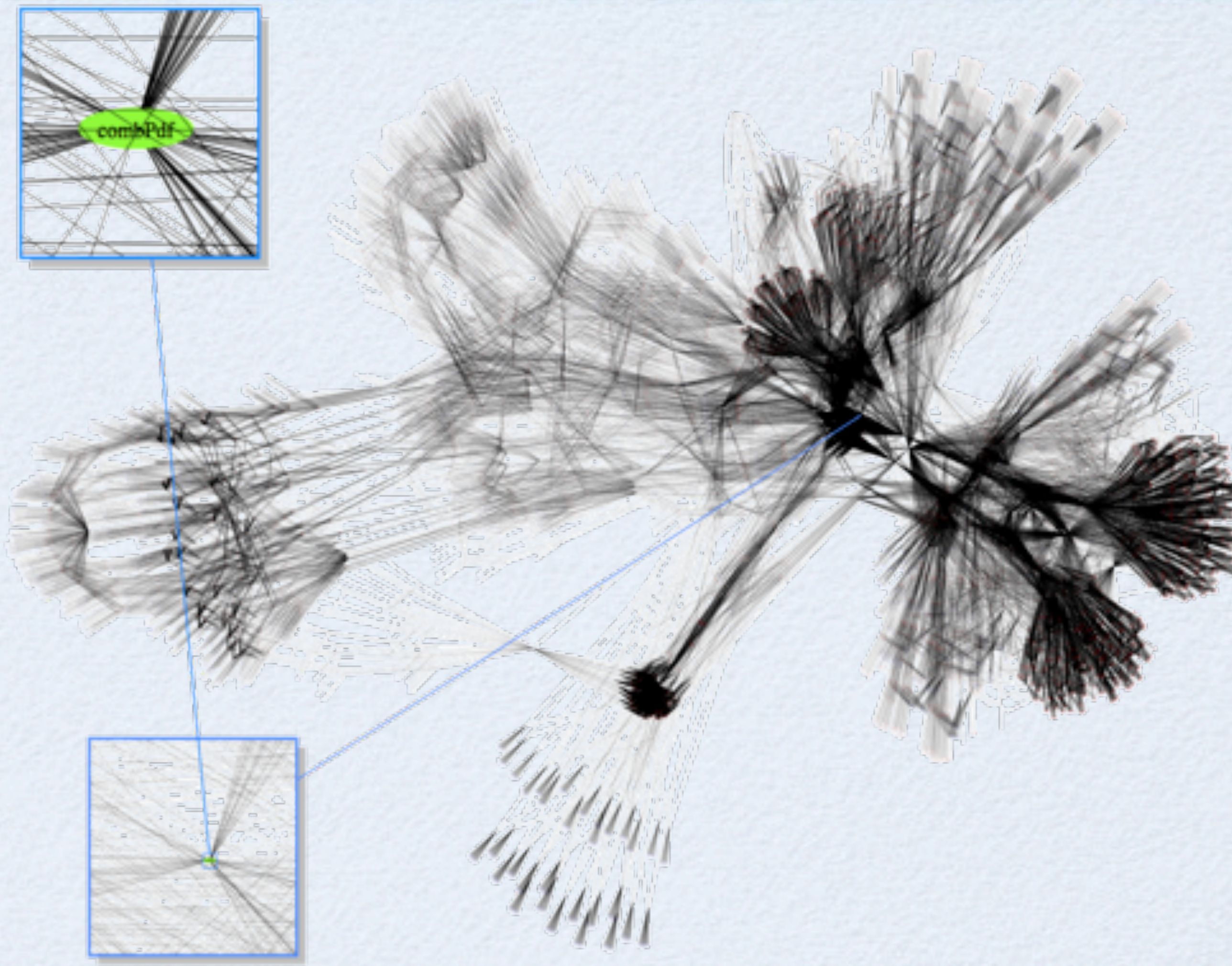





**Higgs observation**

# How well does it scale?

Graph of the full ATLAS Higgs combination model




Model has ~23.000 function objects, ~1600 parameters

Reading/writing of full model takes ~4 seconds
ROOT file with workspace is ~6 Mb

# Summary

- RooFit/RooStats allow you to perform advanced statistical data/analysis
  - LHC results (*e.g.* Higgs observation)
- Capable of using different tools and interpretations (Frequentist/Bayesian) on the same model
- Generic tools capable to deal with large variety of models
  - based on histograms or un-binned data
  - multi-dimensional observations
- Provide tools to facilitate complex model building
  - HistFactory for histogram based analysis

# Documentation

- RooStats TWiki: https://twiki.cern.ch/twiki/bin/view/RooStats/WebHome
- RooStats users guide (not really completed)
    - http://root.cern.ch/viewcvs/branches/dev/roostats/roofit/roostats/doc/usersguide/RooStats_UsersGuide.pdf
- For reference and citation: ACAT 2010 proceedings papers: http://arxiv.org/abs/1009.1003
- RooStats tutorial macros: http://root.cern.ch/root/html534/tutorials/roostats/index.html
- HistFactory document: https://cdsweb.cern.ch/record/1456844/files/CERN-OPEN-2012-016.pdf
- RooStats user support:
    - Request support via ROOT talk forum: http://root.cern.ch/phpBB2/viewforum.php?f=15
      (questions on statistical concepts accepted)
    - contact me directly (email: Lorenzo.Moneta at cern.ch )
- Contacts for statistical questions:
    - ATLAS statistics forum:
        - TWiki: https://twiki.cern.ch/twiki/bin/view/AtlasProtected/StatisticsTools
    - CMS statistics committee:
        - TWiki: https://twiki.cern.ch/twiki/bin/view/CMS/StatisticsCommittee

# Time For Exercises !

Follow the Twiki page at
https://twiki.cern.ch/twiki/bin/view/RooStats/RooStatsTutorialsJune2013#RooStats_Exercises

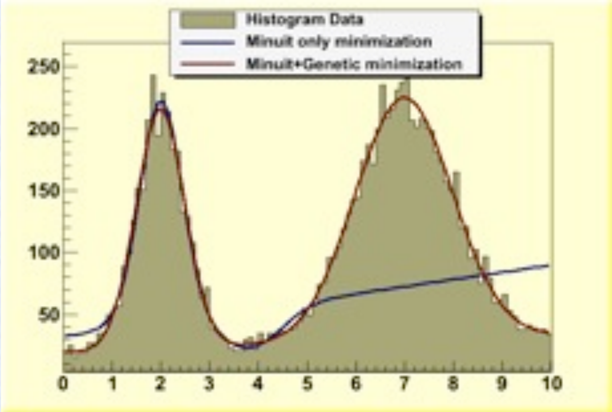
If you have network problem, you can download tar file from the agenda:

- unpack the tar file and open with your browser the page **RooStatsTutorialsJune2013.html**

# RooFit BackUp Slides

# Function Minimization

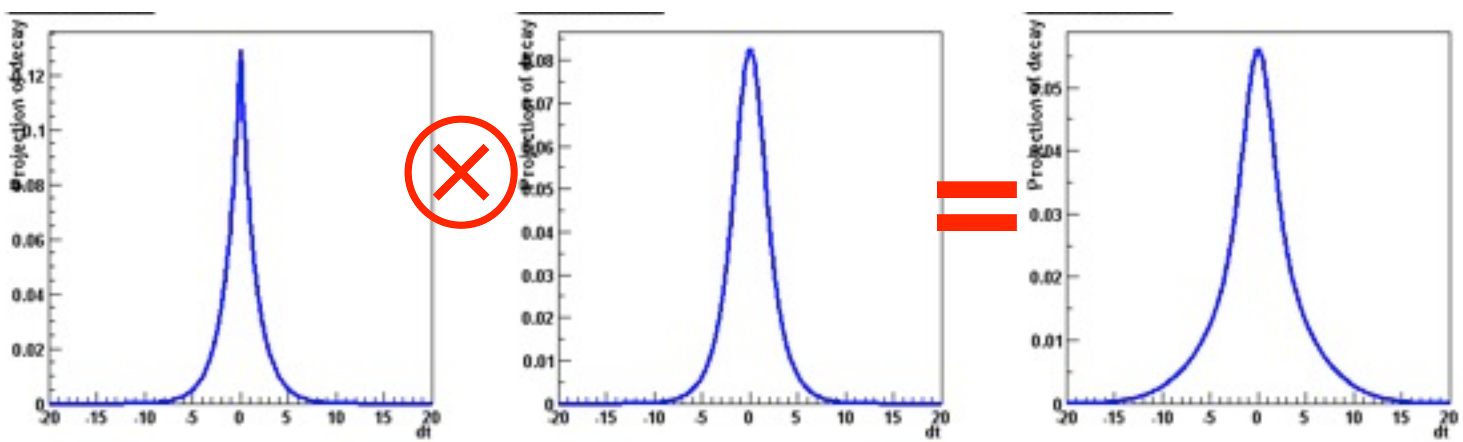
- `Minimizer` interface used for fitting in ROOT (by `ROOT::Fit::Fitter`) and also RooFit/RooStats (via class `RooMinimizer`)
- Control of minimization options and type of minimizer using the `ROOT::Math::MinimizerOptions` class
  - to change the minimizer for fitting:
    - `ROOT::Math::MinimizerOptions::SetDefaultMinimizer("Minuit2");`
  - e.g. to change the tolerance:
    - `ROOT::Math::MinimizerOptions::SetDefaultTolerance(1.E-6);`
  - several other options also available:
    (some specific to the minimizer)
- Possible to combine minimizers
  - e.g. use first Genetic and then Minuit to find the global minimum



# Convolution

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

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



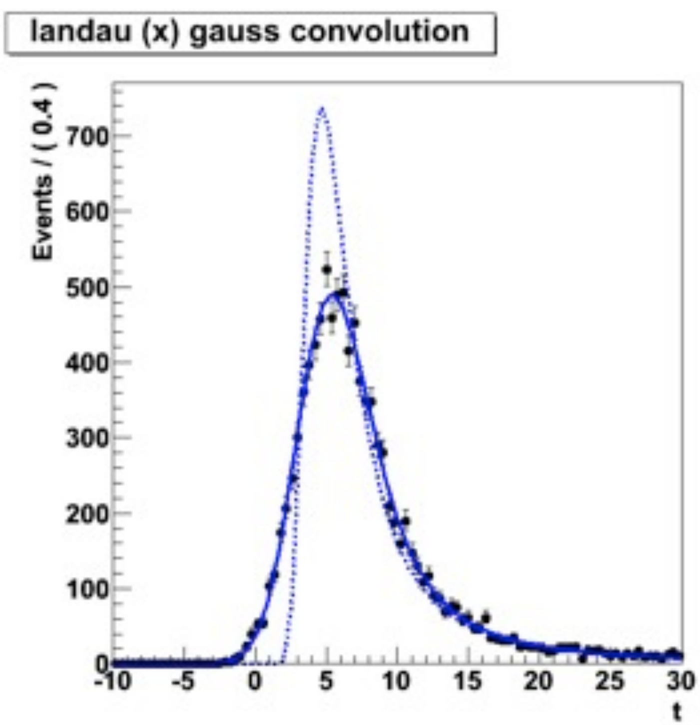

- But numeric calculation of convolution integral can be challenging. No one-size-fits-all solution, but 3 options available
  - Analytical convolution (BW⊗Gauss, various B physics decays)
  - Brute-force numeric calculation (slow)
  - FFT numeric convolution (fast, but some side effects)

# Convolution

- Example

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

w.var("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)



# HistFactory

*see also HistFactory doc* (https://cdsweb.cern.ch/record/1456844/files/CERN-OPEN-2012-016.pdf)

# HistFactory

- Tool available in ROOT (in roofit/histfactory) to build models based on histograms
  - generalization of number counting models

$$\mathcal{P}(n_b|\mu) = \mathrm{Pois}(n_{\mathrm{tot}}|\mu S + B)\left[\prod_{b\in\mathrm{bins}} \frac{\mu\nu_b^{\mathrm{sig}} + \nu_b^{\mathrm{bkg}}}{\mu S + B}\right]$$

where $n_b$ is the data histogram

in general HistFactory produces model of this form

$$\mathcal{P}(n_{cb}, a_p \mid \phi_p, \alpha_p, \gamma_b) = \prod_{c\in\mathrm{channels}} \prod_{b\in\mathrm{bins}} \mathrm{Pois}(n_{cb}|\nu_{cb}) \cdot G(L_0|\lambda, \Delta_L) \cdot \prod_{p\in\mathbb{S}+\mathbf{\Gamma}} P_p(a_p|\alpha_p)$$

luminosity constraint

parameter constraint

HistFactory can be configured with XML files or directly in C++/Python (New in 5.34)

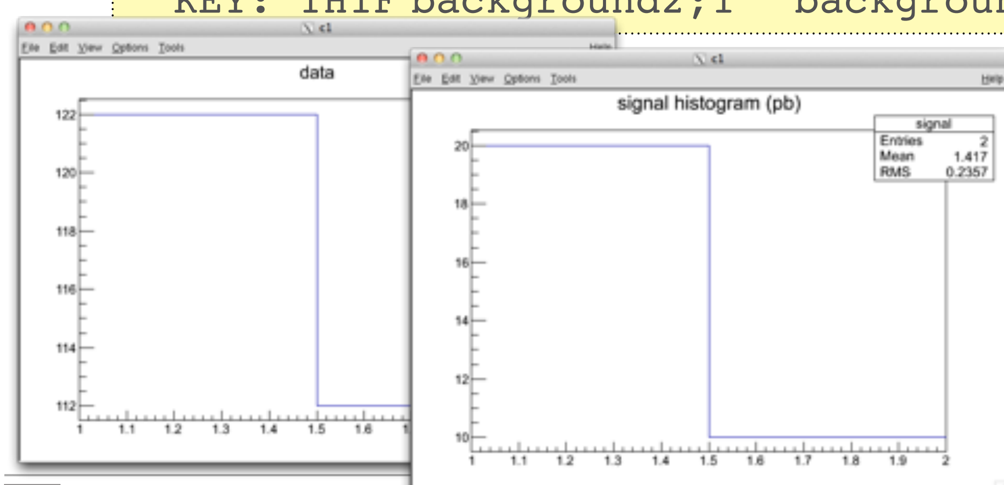
# Creating the Example

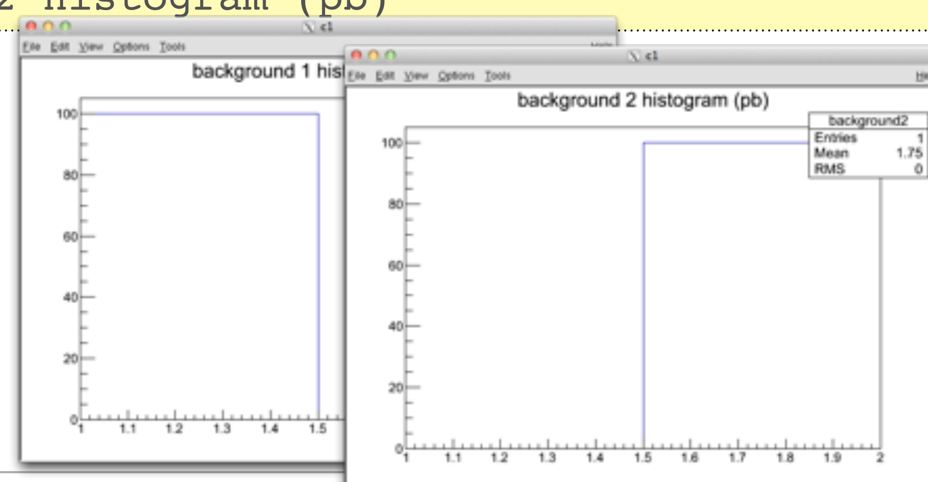
- go to an empty directory

```
[lxplus] ATLASDanalysis > prepareHistFactory

[lxplus] ATLASDanalysis > ls
config data result
```

- What is in data?

```
[lxplus] ATLASDanalysis > root -l data/example.root
root [0]
Attaching file example.root as _file0...
root [1] .ls
TFile**     example.root
 TFile*     example.root
  KEY: TH1F data;1  data
  KEY: TH1F signal;1    signal histogram (pb)
  KEY: TH1F background1;1   background 1 histogram (pb)
  KEY: TH1F background2;1   background 2 histogram (pb)
```



## HistFactory Models

Data: think of it as data points in a histogram

Model: looks the same (it is also a histogram), but one should think about it as a shape (a PDF) that is extended with the number of events in the histogram.

- ➡ $x_i$ events in bin $i$ really means: probability of an event in this bin is $x_i / \Sigma_j\, x_j$ and the PDF is extended with $\Sigma_j\, x_j$ (for bins with equal width).
- ➡ If there is only one bin, this reduces to “number counting form”.

From the HistFactory User Guide:

shapes (integral is one)

$$\mathcal{P}(\{x_1 \dots x_n\}|\mu) = \mathrm{Pois}(n|\mu S + B) \left[ \prod_{e=1}^{n} \frac{\mu S f_{\mathrm{S}}(x_e) + B f_{\mathrm{B}}(x_e)}{\mu S + B} \right]$$

total number of signal (S) and background (B) events including “signal strength modifier” $\mu$

# Example Channel

- config/example_channel.xml

zween — 112×33

ssh | bash | bash

```
<!--
    Single channel configuration exampple.
    The top level configuration XML is example.xml

    NormalizedByTheory should be "True" (not "TRUE" or "true") for all non-data-driven backgrounds.

    If you comment or remove the <Data> tag then it will use the expected data.

    Histogram inputs should be in pb and in top-level xml the lumi should be in 1/pb
    (The important thing is that they match... fb and 1/fb is also ok)

    Note: Config.dtd needs to be accessible.  It can be found in ROOT release area.
    The file system path is relative to location of this XML file, not the executable.
-->

<!DOCTYPE Channel  SYSTEM 'HistFactorySchema.dtd'>

  <Channel Name="channel1" InputFile="./data/example.root" HistoName="" >
    <Data HistoName="data" HistoPath="" />
    <Sample Name="signal" HistoPath="" HistoName="signal">
      <OverallSys Name="syst1" High="1.05" Low="0.95"/>
      <NormFactor Name="SigXsecOverSM" Val="1" Low="0." High="3." Const="True" />
    </Sample>
    <Sample Name="background1" HistoPath="" NormalizeByTheory="True" HistoName="background1">
      <OverallSys Name="syst2" Low="0.95" High="1.05"/>
    </Sample>
    <Sample Name="background2" HistoPath="" NormalizeByTheory="True" HistoName="background2">
      <OverallSys Name="syst3" Low="0.95" High="1.05"/>
      <!-- <HistoSys Name="syst4" HistoPathHigh="" HistoPathLow="histForSyst4"/>-->
    </Sample>
  </Channel>
~
"example_channel.xml" 31L, 1424C                                            12,5          All
```

file we just looked at

names of the histograms from previous page

# Example Model

- config/example.xml

zween — 117×46

ssh | bash | bash

```
<!--
//==============================================================================
// Name        : example.xml
//==============================================================================
-->

<!--
    Top-level configuration,  details for the example channel are in example_channel.xml.
    This is the input file to the executable.

    Note: Config.dtd needs to be accessible.  It can be found in ROOT release area.
    The file system path is relative to location of this XML file, not the executable.
-->

<!DOCTYPE Combination  SYSTEM 'HistFactorySchema.dtd'>

<Combination OutputFilePrefix="./results/example" Mode="comb" >

<Input>./config/example_channel.xml</Input>

  <Measurement Name="GaussExample" Lumi="1." LumiRelErr="0.1" BinLow="0" BinHigh="2" Mode="comb" >
    <POI>SigXsecOverSM</POI>
    <ParamSetting Const="True">Lumi alpha_syst1</ParamSetting>
    <!-- don't need <ConstraintTerm> default is Gaussian-->
  </Measurement>

  <Measurement Name="GammaExample" Lumi="1." LumiRelErr="0.1" BinLow="0" BinHigh="2" Mode="comb" >
    <POI>SigXsecOverSM</POI>
    <ParamSetting Const="True">Lumi alpha_syst1</ParamSetting>
    <ConstraintTerm Type="Gamma" RelativeUncertainty=".3">syst2</ConstraintTerm>
  </Measurement>

  <Measurement Name="LogNormExample" Lumi="1." LumiRelErr="0.1" BinLow="0" BinHigh="2" Mode="comb" >
    <POI>SigXsecOverSM</POI>
    <ParamSetting Const="True">Lumi alpha_syst1</ParamSetting>
    <ConstraintTerm Type="LogNormal" RelativeUncertainty=".3">syst2</ConstraintTerm>
  </Measurement>

  <Measurement Name="ConstExample" Lumi="1." LumiRelErr="0.1" BinLow="0" BinHigh="2" Mode="comb" ExportOnly="True">
    <POI>SigXsecOverSM</POI>
    <ParamSetting Const="True">Lumi alpha_syst1</ParamSetting>
  </Measurement>

</Combination>
```

1,1 All

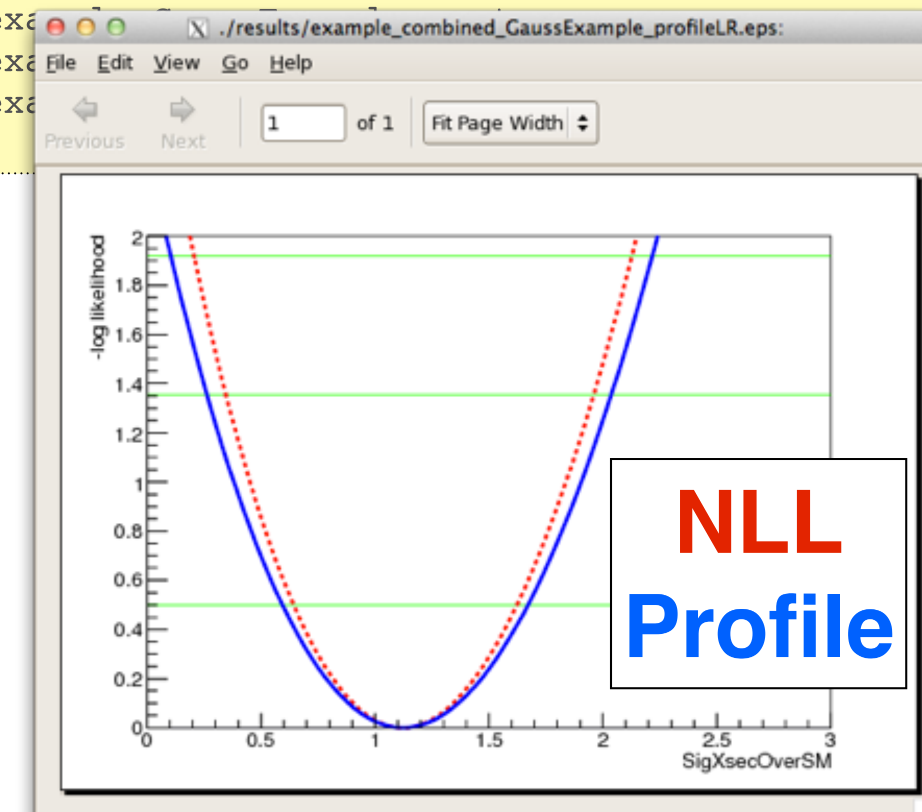
use that channel

do not run ProfileLikelihoodCalculator for this Measurement

# Running Example

- from the <u>main</u> directory:

```
[lxplus] ATLASDanalysis > hist2workspace config/example.xml
... producing a lot of output ...

[lxplus] ATLASDanalysis > ls results/
example_channel1_ConstExample_model.root        example_combined_GaussExample_model.root
example_channel1_GammaExample_model.root        example_combined_GaussExample_profileLR.eps
example_channel1_GammaExample_profileLR.eps     example_combined_LogNormExample_model.root
example_channel1_GaussExample_model.root        example_combined_LogNormExample_profileLR.eps
example_channel1_GaussExample_profileLR.eps     example_ConstExample.root
example_channel1_LogNormExample_model.root      example_GammaExample.root
example_channel1_LogNormExample_profileLR.eps   exa
example_combined_ConstExample_model.root        exa
example_combined_GammaExample_model.root        exa
example_combined_GammaExample_profileLR.eps
```




- this created many .root files and also .eps files
  - ➡ eps files are the outputs of the ProfileLikelihoodCalculator that was run automatically (use ExportOnly=”True” to switch that off)

# Look at result

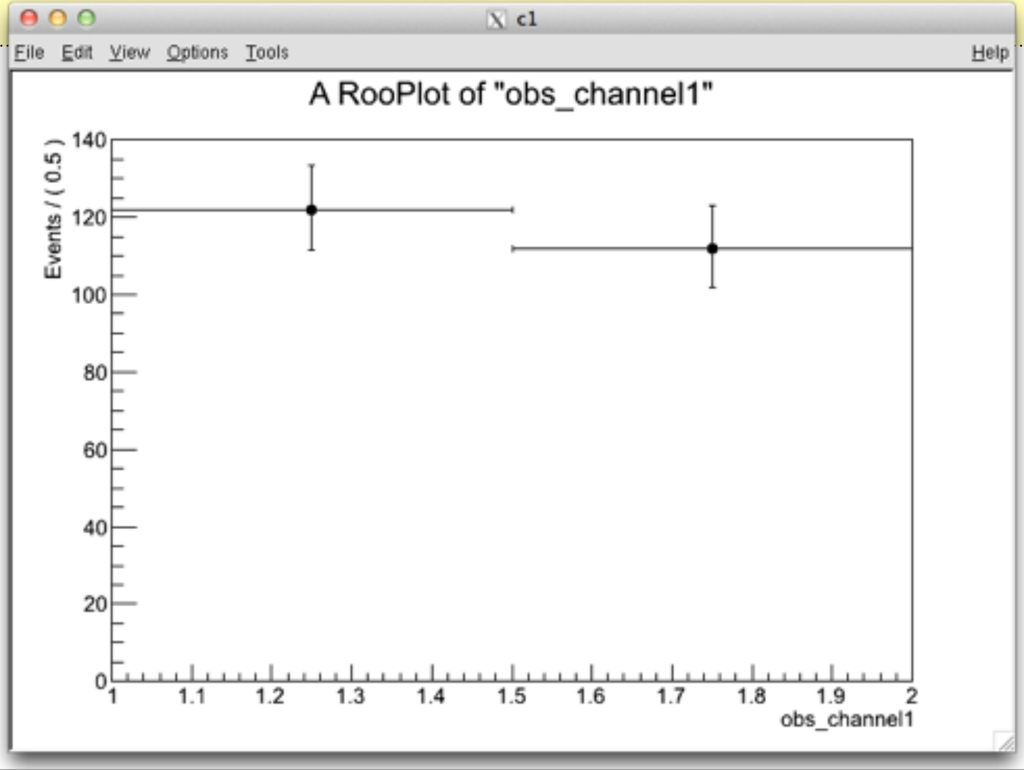
- Find out workspace name, model name and data name:

```
[lxplus] ATLASDanalysis > root -l results/
example_combined_GaussExample_model.root

root [1] .ls
TFile**      results/example_combined_GaussExample_model.root
 TFile*      results/example_combined_GaussExample_model.root
  KEY: RooWorkspace combined;1   combined
  KEY: TProcessID   ProcessID0;10c6e344e-2565-11e0-9717-ecd28a89beef


root [2] combined->Print()
RooWorkspace(combined) combined contents
... print out of variables, p.d.f.s, functions, named sets, and ...

datasets
--------
RooDataSet::asimovData(channelCat,obs_channel1)
RooDataSet::obsData(channelCat,obs_channel1)

generic objects
---------------
RooStats::ModelConfig::ModelConfig

root [3]
```

## Reading obsData

- in the standard form, the model is built using RooHistFuncs, which is more efficient than the number counting form
  - ➡ look at data like this:

```
root [2] f = combined->var("obs_channel1")->frame()
(const class RooPlot*)0x7fc173162800
root [3] combined->data("obsData")->plotOn(f)
(const class RooPlot*)0x7fc173162800
root [4] f->Draw()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [5]
```



# Using a HistFactory model

```
root [5] .x /afs/cern.ch/sw/lcg/app/releases/ROOT/5.30.01/x86_64-slc5-gcc43-opt/root/tutorials/
roostats/StandardProfileLikelihoodDemo.C("results/example_combined_GaussExample_model.root",
"combined", "ModelConfig", "obsData")

[#1] INFO:Minization --  Including the following contraint terms in minimization:
(alpha_syst2Constraint,alpha_syst3Constraint)
ProfileLikelihoodCalculator::DoGlobalFit - using Minuit / Migrad with strategy 1
[#1] INFO:Minization --  Including the following contraint terms in minimization:
(alpha_syst2Constraint,alpha_syst3Constraint)
[#1]                                   rLevel(nll_simPdf_simData_with_constr) Summation contains a RooNLLVar, using
its e
[#1]                                   eConst:
[#1] INFO:Fitting -- RooAbsTestStatistic::initSimMode
[#1] INFO:Minization -- RooMinimizer::optimizeConst:

  RooFitResult: minimized FCN value: 8.44132, estimat
                covariance matrix quality: Full, accu

    Floating Parameter    FinalValue +/-  Error
  --------------------  --------------------------
         SigXsecOverSM    1.1212e+00 +/-  5.26e-01
           alpha_syst2   -1.3646e-02 +/-  9.75e-01
           alpha_syst3    2.7826e-02 +/-  9.19e-01

[#1] INFO:Fitting -- RooAbsTestStatistic::initSimMode
[#1] INFO:Minization -- RooProfileLL::evaluate(nll_si
MINUIT
[#1] INFO:Fitting -- RooAddition::defaultErrorLevel(n
its error level
[#1] INFO:Minization -- RooProfileLL::evaluate(nll_si
likelihood for current configurations w.r.t all obser
[#1] INFO:Fitting -- RooAbsTestStatistic::initSimMode
[#1] INFO:Minization -- RooProfileLL::evaluate(nll_si
(SigXsecOverSM=1.12102)
.....................................................
.....................................................
default TCanvas with name c1

95% interval on SigXsecOverSM is : [0.102174, 2.21605]
root [6]
```

from previous pages