UNIVERSIDADE DO ESTADO DO RIO DE JANEIRO
UERJ
DFNAE
DEPARTAMENTO DE FÍSICA NUCLEAR E ALTAS ENERGIAS

RooFit

Aula 05

Outline

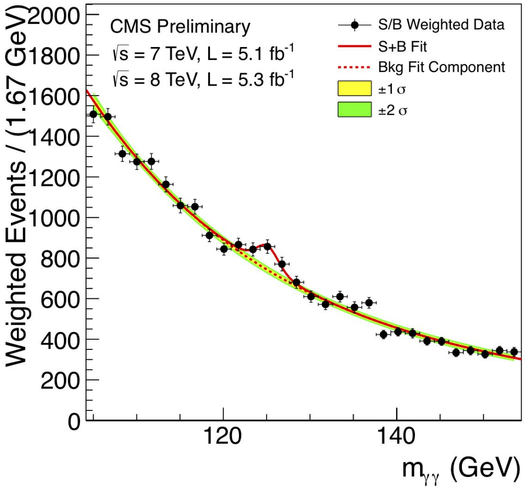
- Introduction to RooFit
  - –Basic functionality
  - –Model building using the workspace
  - –Composite models
- Exercises on RooFit:
  - –building and fitting model

Material based:

- on slides from W. Verkerke (author of RooFit)
- ROOT Tutorial at UERJ - 2015

What is Fitting ?

- Estimate parameters of a hypothetical distribution from the observed data distribution
  - $y = f(x \mid \theta)$ is the fit model function
- Find the best estimate of the parameters θ assuming $f(x \mid \theta)$
- Both Likelihood and Chi2 fitting are supported in ROOT



Example

Higgs ➞ γγ spectrum

We can fit for:

- the expected number of Higgs events
- the Higgs mass

Parameter Estimation

- Given a model for our observed data (Probability Density Function) we want to estimate the parameter of our model
- The model of the observed data is expressed using the Probability Density Function (PDF)
  - –the PDF is a differential probability $f(\vec{x},\theta)$
    - e.g. probability of observing event in an histogram bin $P_{bin} = \int_{bin} f(\vec{x},\theta)d\vec{x}$
  - –the PDF is normalised to 1 when integrated in all the sample space Ω $\int_{\Omega} f(\vec{x},\theta)d\vec{x} = 1$
- To estimate the parameter we use the **Likelihood Function** $L(\vec{x}_1,..,\vec{x}_N|\theta) = \prod_{i=1}^{N} f(\vec{x}_i,\theta)$
- More convenient to work with the log of the likelihood-function
- Use negative log-likelihood function and find global minimum $-\log L(\vec{x}_1,..,\vec{x}_N|\theta) = -\sum_i \log f(\vec{x}_i,|\theta)$
- Extend likelihood

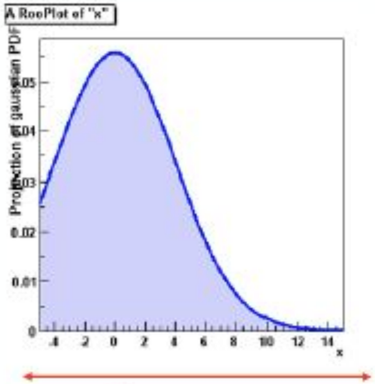
$$\log L(x|\theta) = \sum_{bin} \log e^{-\nu} \frac{\nu^N}{N!} f(x|\theta)$$

What is RooFit ?

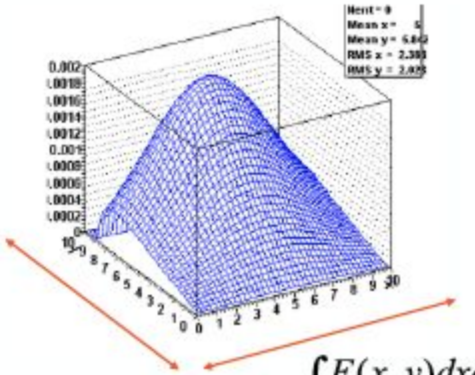
A toolkit distributed with ROOT and based on its core functionality.

• It is used to model distributions, which can be used for fitting and statistical data analysis.
–model distribution of observable **x** in terms of parameters **p**
• probability density function (p.d.f.): P(x;p)
• p.d.f. are normalized over allowed range of observables x with respect to the parameters **p**

$$\int_{\Omega} P(\vec{x};\vec{p})d\vec{x} = 1$$



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



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

Why RooFit ?

–ROOT can handle complicated functions but it might require writing large amount of code

–Normalization of p.d.f. not always trivial

- RooFit does it automatically

–In complex fit, computation performance important

- need to optimize code for acceptable performance
- built-in optimization available in RooFit
  - –evaluation of model parts only when needed

–Simultaneous fit to different data samples

–Provide full description of model for further use

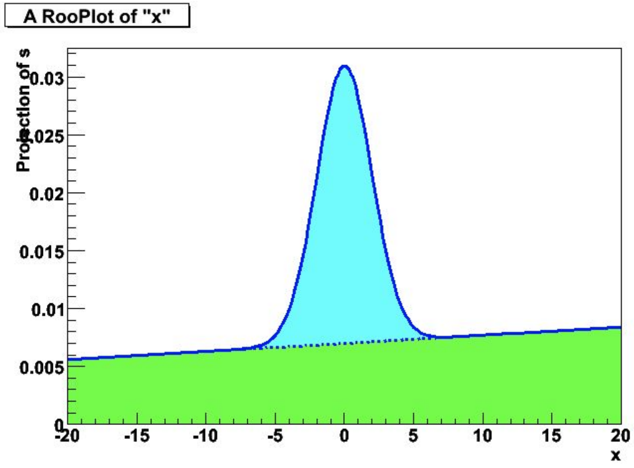
RooFit

- 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

Math - Functions vs probability density functions

• Why use probability density functions rather than ‘plain’ functions to model the data?

–Easier to interpret the models.
If Blue and Green pdf are each guaranteed to be normalized to 1,
then fractions of Blue,Green can be cleanly interpreted as #events
–Many statistical techniques only function properly with p.d.f. (e.g maximum likelihood fits)



• What is difficult with p.d.f ?
–The normalization can be hard to calculate
(e.g. it can be different for each set of parameter values p)
• In >1 dimension (numeric) integration can be particularly hard
–RooFit aims to simplify these tasks

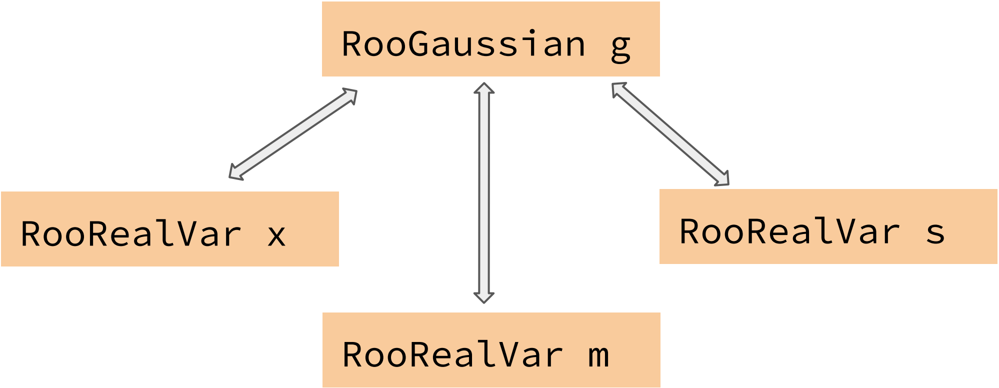
RooFit Modeling

Mathematical concepts are represented as C++ objects

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

RooFit Modeling

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



RooFit code:

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

The simplest possible example

Name of object — Title of object — initial range

objects representing a ‘real’ value:

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

initial range

PDF object:

```
RooGaussian model(“model”, “signal pdf”, x,mean,sigma);
```

References to variables

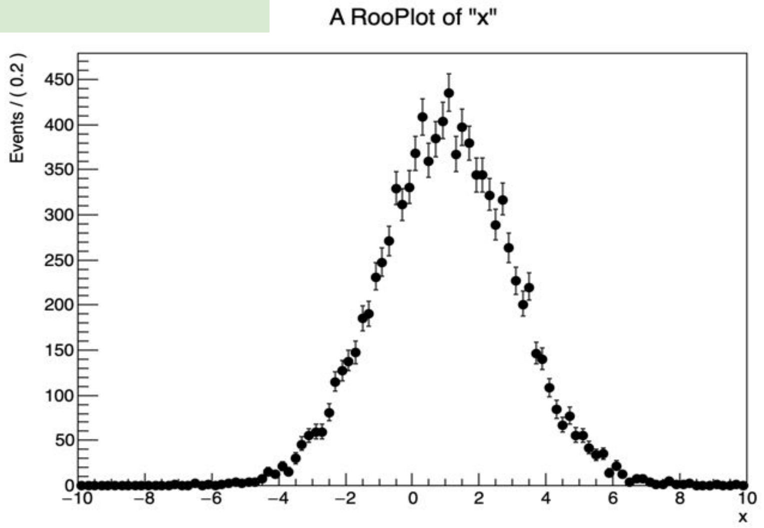
Basics - Generating toy MC events

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

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

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

Can generate both binned and unbinned datasets



Data visualization

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

Basics - Importing data

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

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

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

- Binned data can be imported from ROOT THx histograms

```
// Import binned data
RooDataHist data(“data”,”data”,x,Import(*myTH1));
```

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

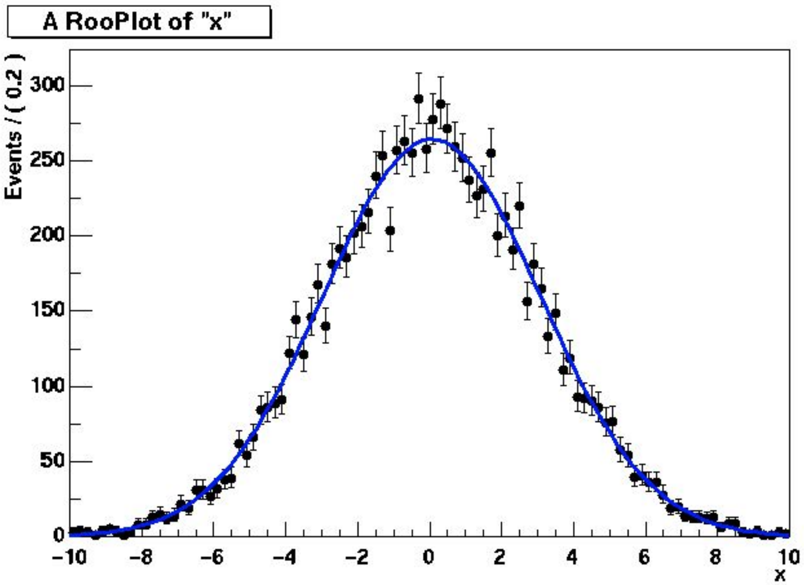
Basics - Fitting the data

- 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();
```



PDF automatically
normalized to dataset

Exercises working with RooFit

Exercise 1

- Create a Gaussian p.d.f, generate some toy data and fit it
- Extra:
  - –Play with some other p.d.f
    - e.g. Exponential pdf
- or some other p.d.f you want.
- You can find several pdf in roofit reference documentations
  - –http://root.cern.ch/root/html/ROOFIT_ROOFIT_Index.html
  - –(all class names in RooFit starts with “Roo”)

https://github.com/sandrofonseca/rootFitTutorial/blob/master/roofitUERJ/GausModelRooFit.ipynb

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
  - –all information is available for further analysis
- Combination of results joining workspaces in a single one
  - –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)
```

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

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

We will work in the examples 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 functions into a workspace
  - When importing a function, all its components (variables) are automatically imported too

```
RooRealVar x(“x”,”x”,-10,10);
RooRealVar mean(“mean”,”mean”,5);
RooRealVar sigma(“sigma”,”sigma”,3);
RooGaussian f(“f”,”f”,x,mean,sigma);
// imports f,x,mean and sigma
w.import(f);
```

Using the workspace

- Looking into a workspace

```
w.Print() ;
variables
---------
(mean,sigma,x)
p.d.f.s
-------
RooGaussian::f[ x=x mean=mean sigma=sigma ] =
0.249352
```

- Getting variables and functions out of a workspace

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

Using the workspace

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

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

- Organizing your code – Separate construction and use of models

```
void driver() {
RooWorkspace w("w") ;
makeModel(w) ;
useModel(w) ;
}
void makeModel(RooWorkspace& w) {
// Construct model here
}
void useModel(RooWorkspace& w) {
// Make fit, plots etc here
}
```

Factoring Syntax

- Rule #1 – Create a variable

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

- Rule #2 – Create a function or pdf object

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

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

```
Gaussian::g(x,mean,sigma)
// equivalent to RooGaussian(“g”,”g”,x,mean,sigma)
Polynomial::p(x,{a0,a1})
// equivalent to RooPolynomial(“p”,”p”,x”,RooArgList(a0,a1));
```

Factoring 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

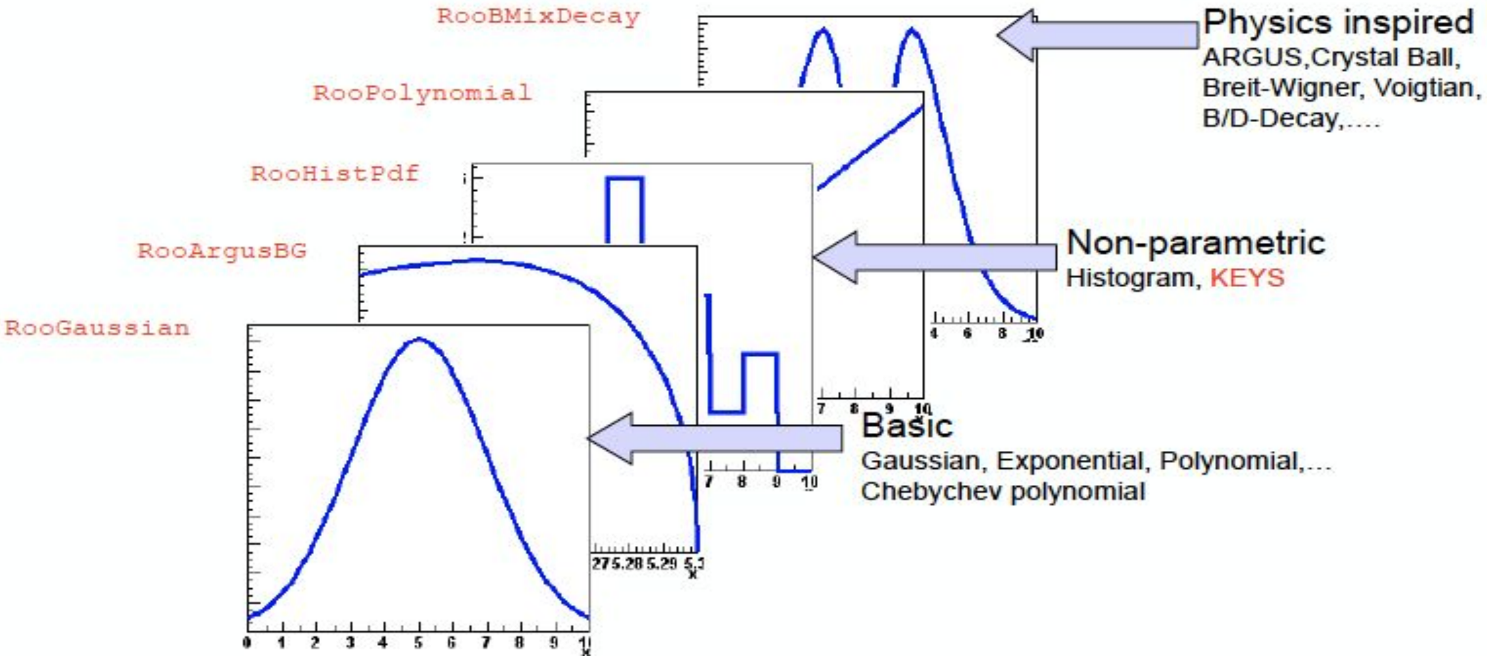
```
Gaussian::g(x[-10,10],0,3)
```

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

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

Model Building

- RooFit provides a collection of compiled standard PDF classes



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

(Re)using standard components

- List of most frequently used pdfs and their factory spec

  Gaussian `Gaussian::g(x,mean,sigma)`

  Breit-Wigner `BreitWigner::bw(x,mean,gamma)`

  Landau `Landau::l(x,mean,sigma)`

  Exponential `Exponential::e(x,alpha)`

  Polynomial `Polynomial::p(x,{a0,a1,a2})`

  Chebychev `Chebychev::p(x,{a0,a1,a2})`

  Kernel Estimation `KeysPdf::k(x,dataSet)`

  Poisson `Poisson::p(x,mu)`

  Voigtian `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)”);
```

- re-parametrization of variables (making functions)

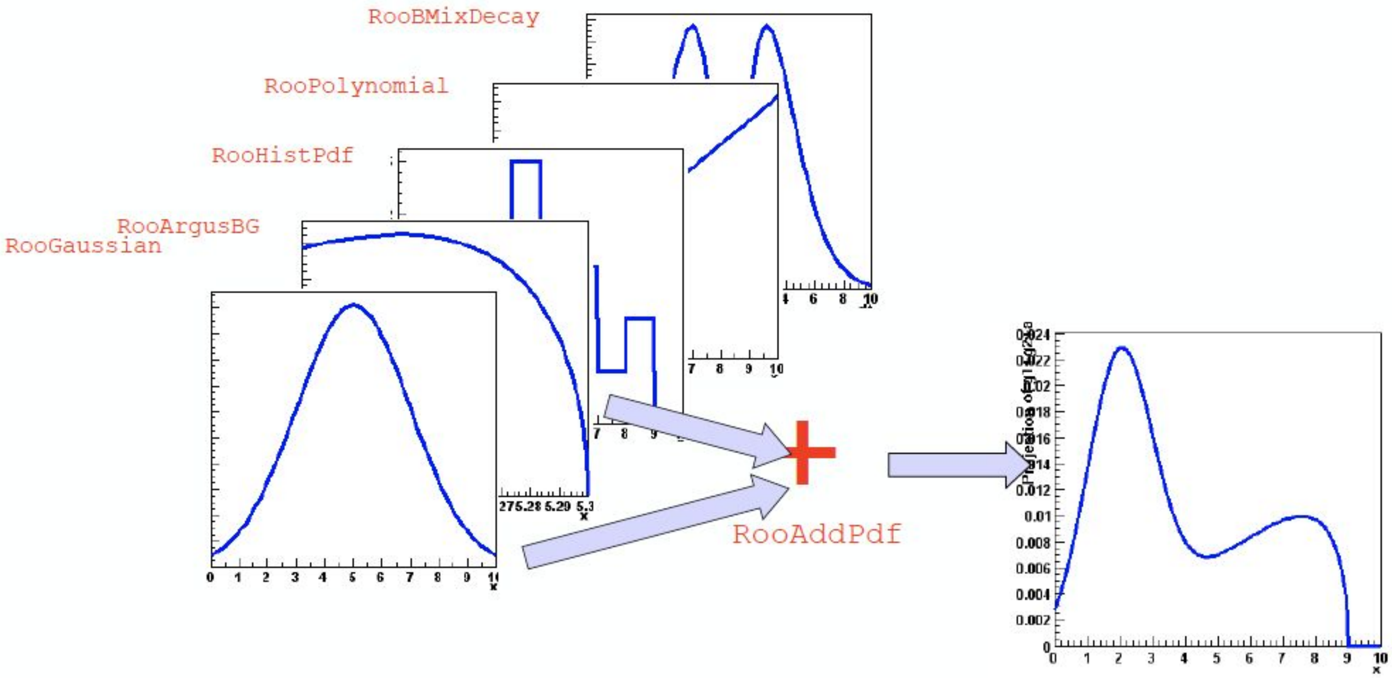
```
w.factory(“expr::w(‘(1-D)/2’,D[0,1])”);
```

  – 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`



Adding p.d.f.s - Factory syntax

- Additions created through a SUM expression

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

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

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

–Note that last PDF does not have an associated fraction in case of floating overall normalization

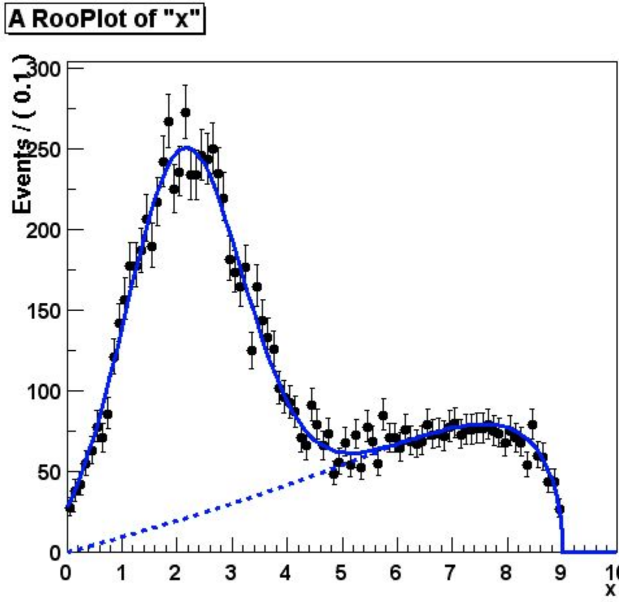
  - when the normalization is fitted from the observed events

- Complete example

```
w.factory("Gaussian::gauss1(x[0,10],mean1[2],sigma[1]") ;
w.factory("Gaussian::gauss2(x,mean2[3],sigma)") ;
w.factory("ArgusBG::argus(x,k[-1],9.0)") ;
w.factory("SUM::sum(g1frac[0.5]*gauss1, g2frac[0.1]*gauss2,argus)");
```

Plotting Components of a p.d.f

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




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

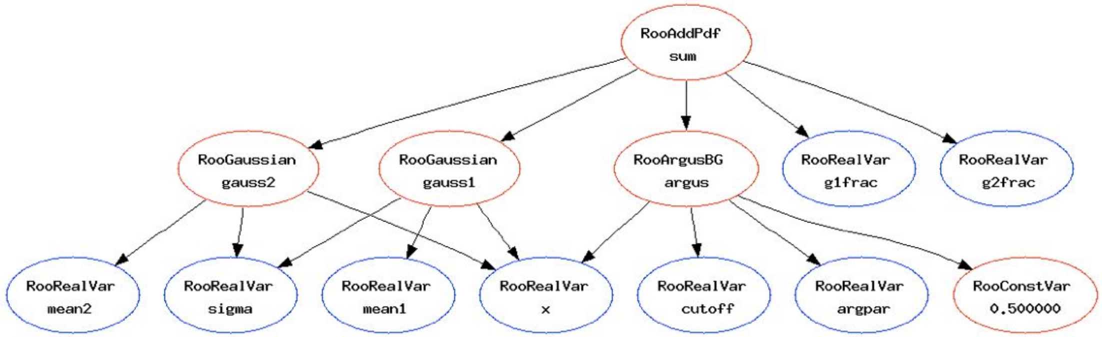
Operations on specific to composite pdfs

- Tree printing mode of workspace reveals component structure

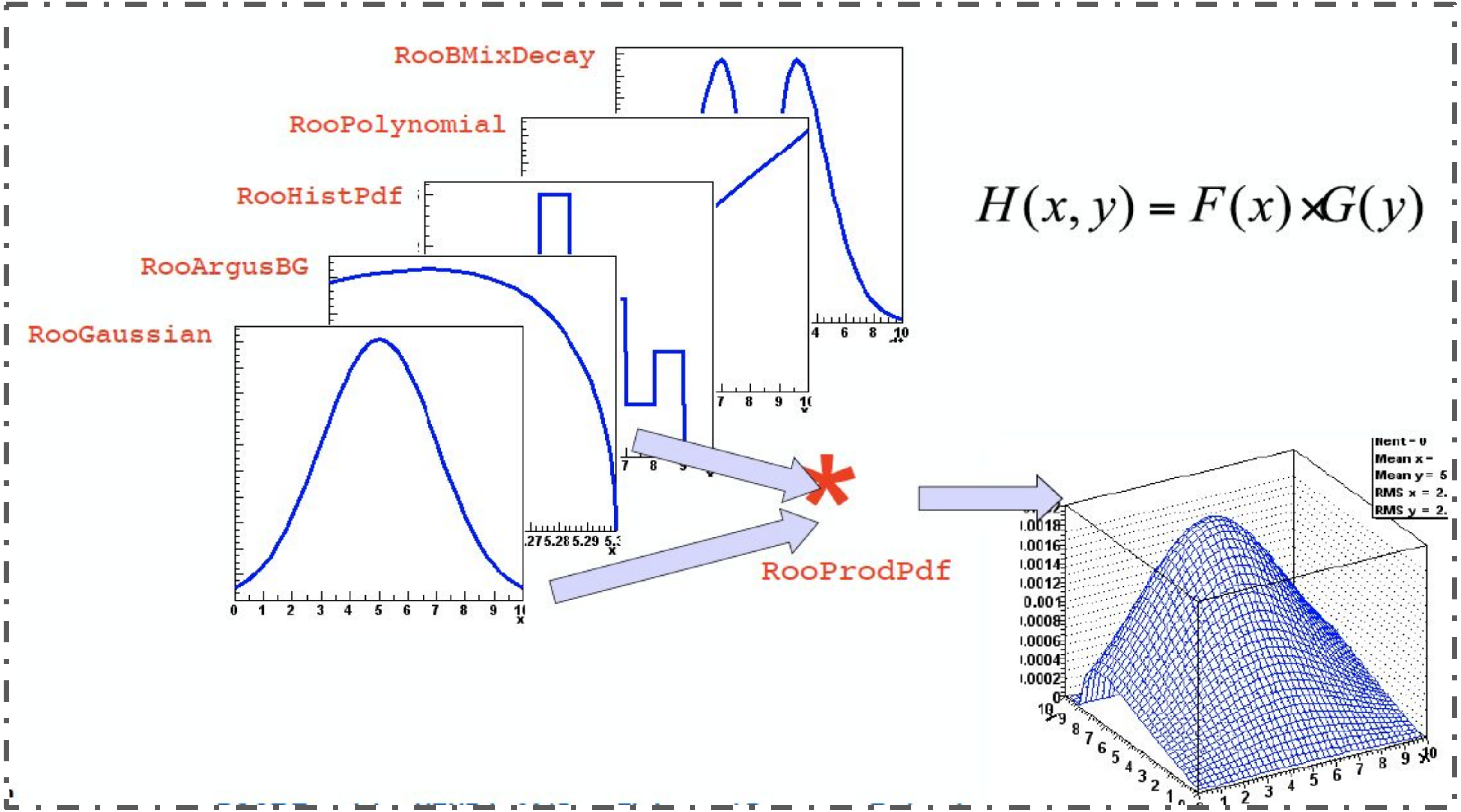
```
w.pdf("sum")->Print("t");
  RooAddPdf::sum[ g1frac * g1 + g2frac * g2 + [%] * argus ] = 0.0687785
    RooGaussian::g1[ x=x mean=mean1 sigma=sigma ] = 0.135335
    RooGaussian::g2[ x=x mean=mean2 sigma=sigma ] = 0.011109
    RooArgusBG::argus[ m=x m0=k c=9 p=0.5 ] = 0
```

  - Can also make input files for GraphViz visualization

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



Products of uncorrelated p.d.f.s



Uncorrelated products - Mathematics and constructors

2D

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

nD

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

– 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)");
```

Constructing joint p.d.f.s (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)”);
```

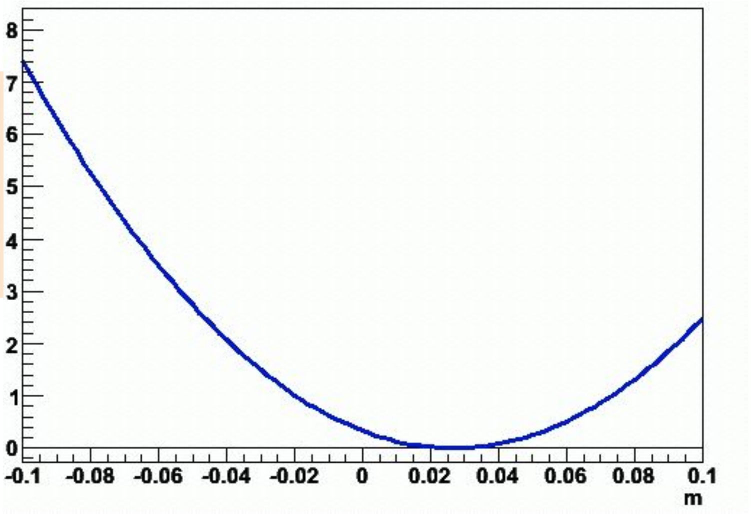
- Construct joint datasets
  - contains observables (“x”) and category (“index”)

```
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 MINIZATION using MINUIT
  - Result of minimization are immediately propagated to RooFit variable objects (values and errors)

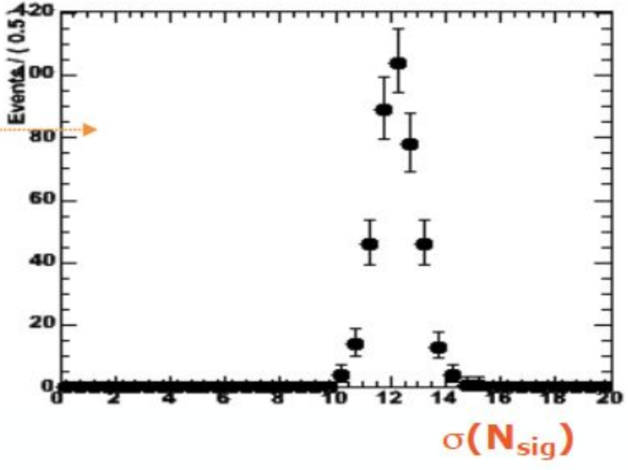
```
// Create likelihood (calculation parallelized on 8 cores)
RooAbsReal* nll = w::model.createNLL(data,NumCPU(8)) ;
RooMinimizer m(*nll) ; // create Minimizer class
m.minimize("Minuit2","Migrad"); // minimize using Minuit2
m.hesse() ; // Call HESSE
m.minos(w::param) ; // Call MINOS for 'param'
RooFitResult* 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")) ;
```

Fit Validation Study - The pull distribution

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



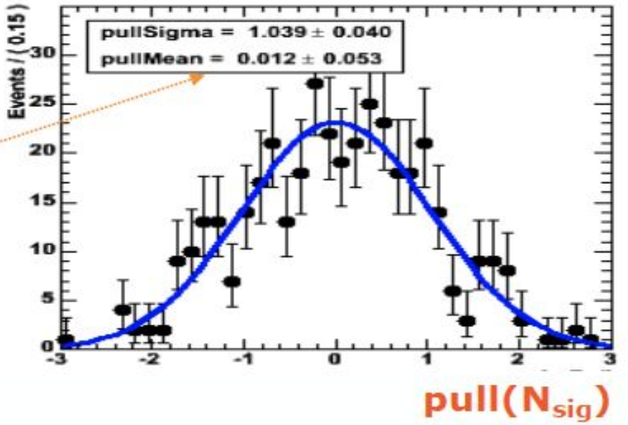
- Solution: look at the ***pull distribution***

Definition:

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

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

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



Exercise 2

This exercise involves RooFit only

- Construct a J/ψ and ψ(2S) + background PDF
  - J/ψ with a Crystal Ball function
  - ψ(2S) with a similar(spoiler!) Crystal Ball
  - Background with a polynomial

- For now, the ψ(2S) will involve a very small amount of signal events
- Fit it, plot it, save it

Input file is here: https://cernbox.cern.ch/index.php/s/mccq4dW7qlYWOHx

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

- Pull : http://physics.rockefeller.edu/luc/technical_reports/cdf5776_pulls.pdf

https://github.com/sandrofonseca/rootFitTutorial/tree/master/roofitUERJ

Backup

Composition of p.d.f.s

RooFit pdf building blocks do not require variables as input, just real-valued functions

– Can substitute any variable with a function expression in parameters and/or observables

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

– Example: Gaussian with shifting mean

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

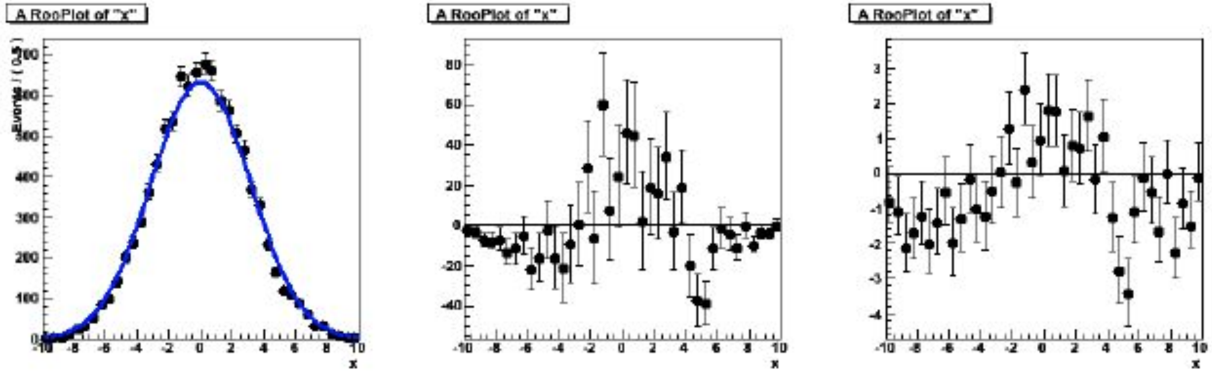
– No assumption made in function on a,b,x,y being observables or parameters, any combination will work

How do you know if your fit was “good”?

- Goodness-of-fit broad issue in statistics (we will see maybe later)

- For one-dimensional fits, a χ^2 is usually the right thing to do
  - –Some tools implemented in RooPlot to be able to calculate χ^2/ndf of curve w.r.t data

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



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

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