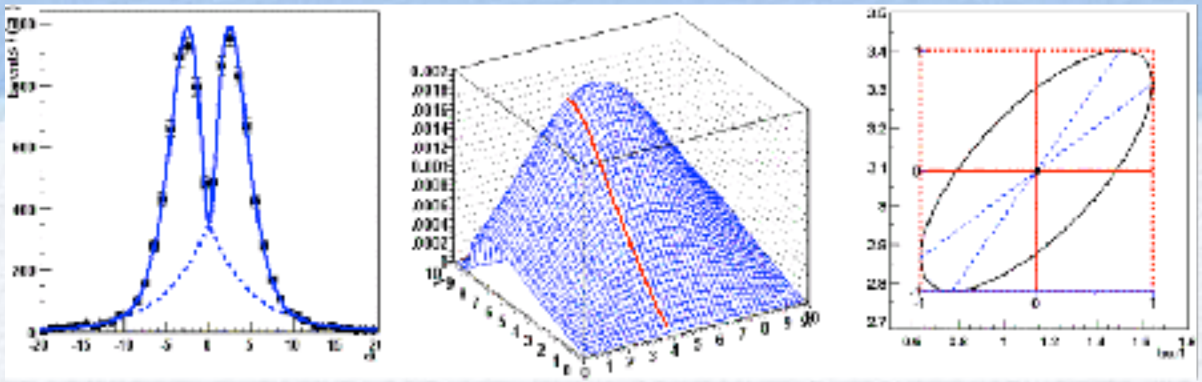
# RooFit



# Outline

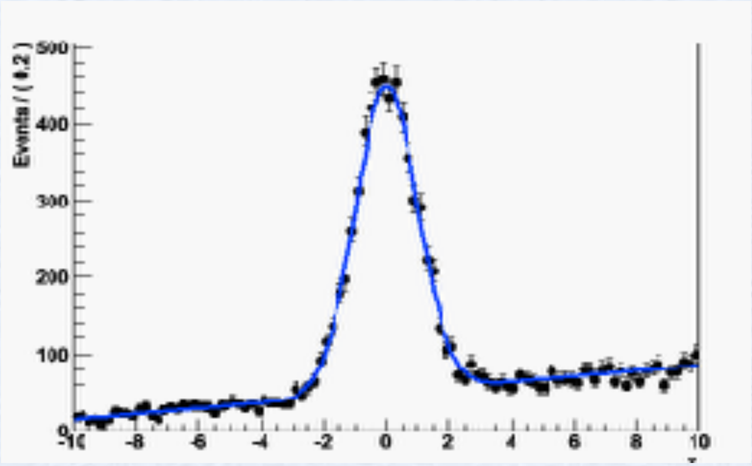
- Introduction to RooFit
  - Basic functionality
  - Model building using the workspace
  - Composite models

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

- Exercises on RooFit:
  - building and fitting model

# RooFit

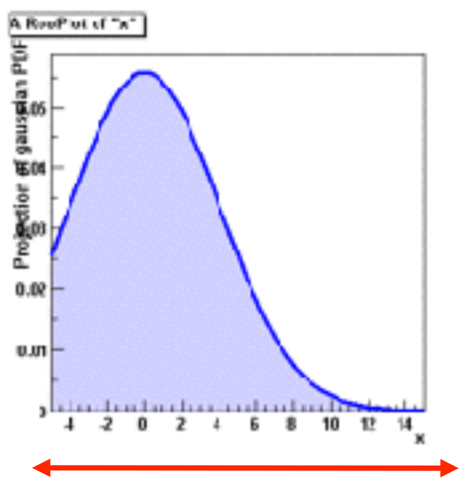
- Toolkit for data modeling
  - developed by *W. Verkerke and D. Kirkby*
- model distribution of observable ***x*** in terms of parameters ***p***
  - probability density function (pdf): $\mathcal{P}$`(x;p)`
- pdf are normalized over allowed range of observables ***x*** with respect to the parameters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



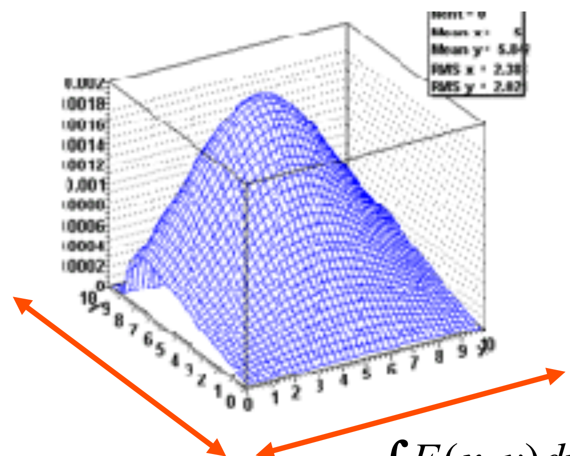
# Mathematic – Probability density functions

- 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

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



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



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

- 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 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 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

# RooFit Modeling

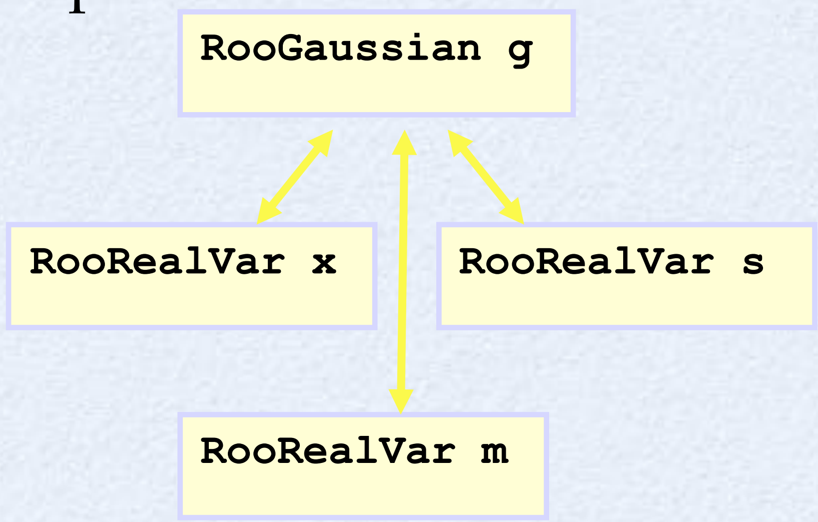
Mathematical concepts are represented as C++ objects

| 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

*Gaus(x,m,s)*

Example: Gaussian pdf




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

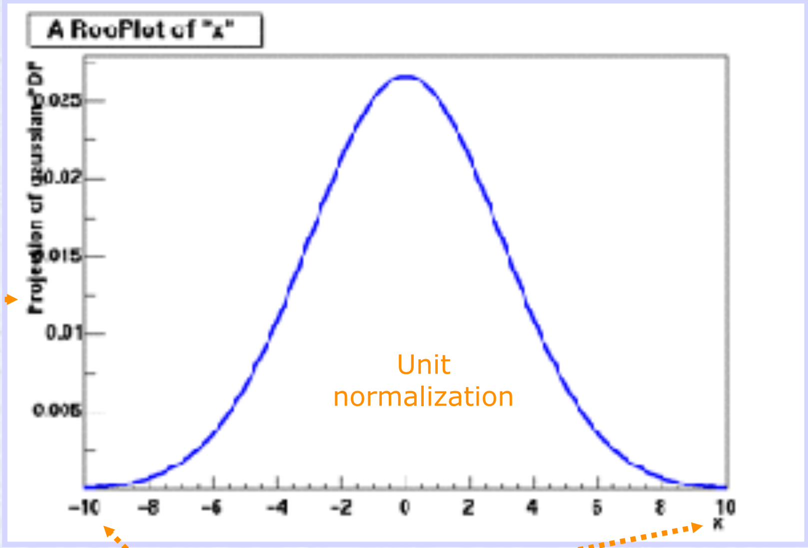
# RooFit Functionality

- 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

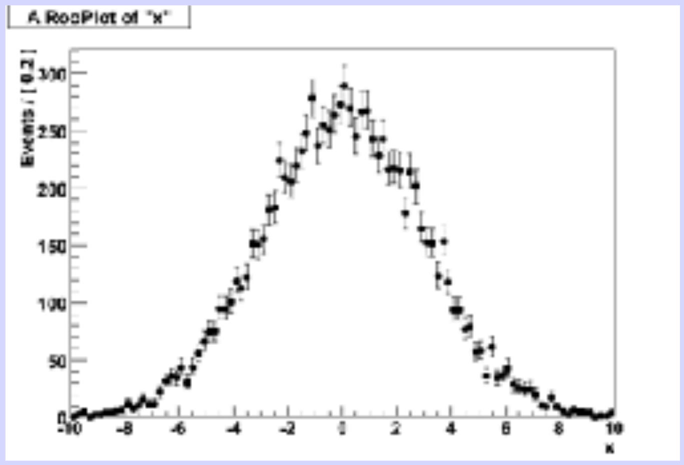
```
RooDataSet * data = pdf->generate(*x,10000);
```

- 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



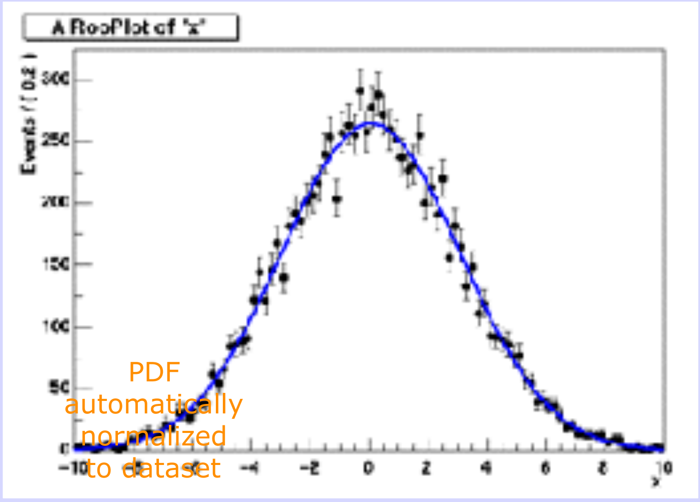
# RooFit Functionality

- 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
  - 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)
```

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") ;
```

# 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)) ;
```

# 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** internally
  - Specify a **RooArgSet** to import multiple observables

- Import from a text file of variables (separated by white spaces)

```
// Import unbinned data from a text file
RooDataSet * data = RooDataSet::read("data.txt",RooArgList(x,y)) ;
```

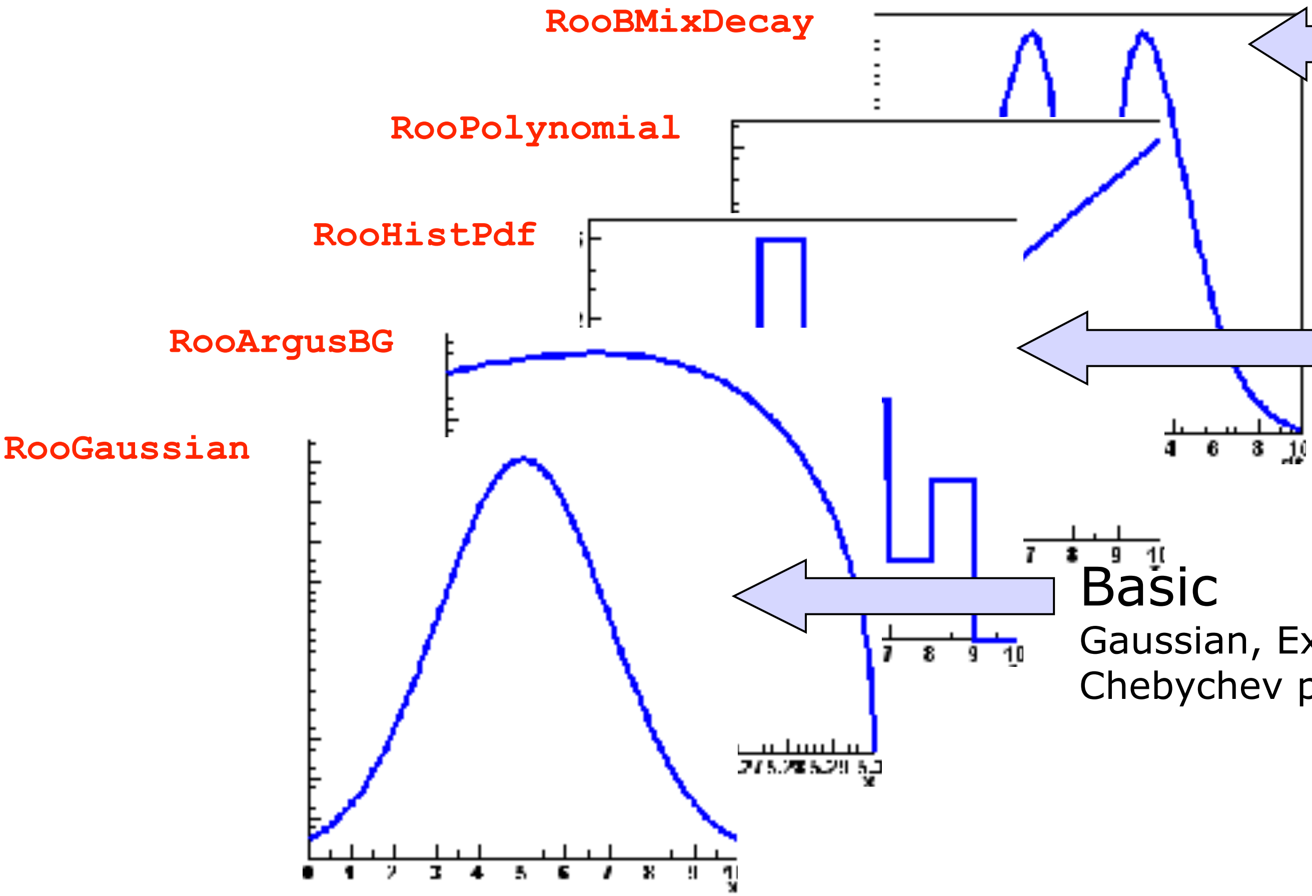
- Binned data can be imported from ROOT **THx** histograms

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

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

## 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)

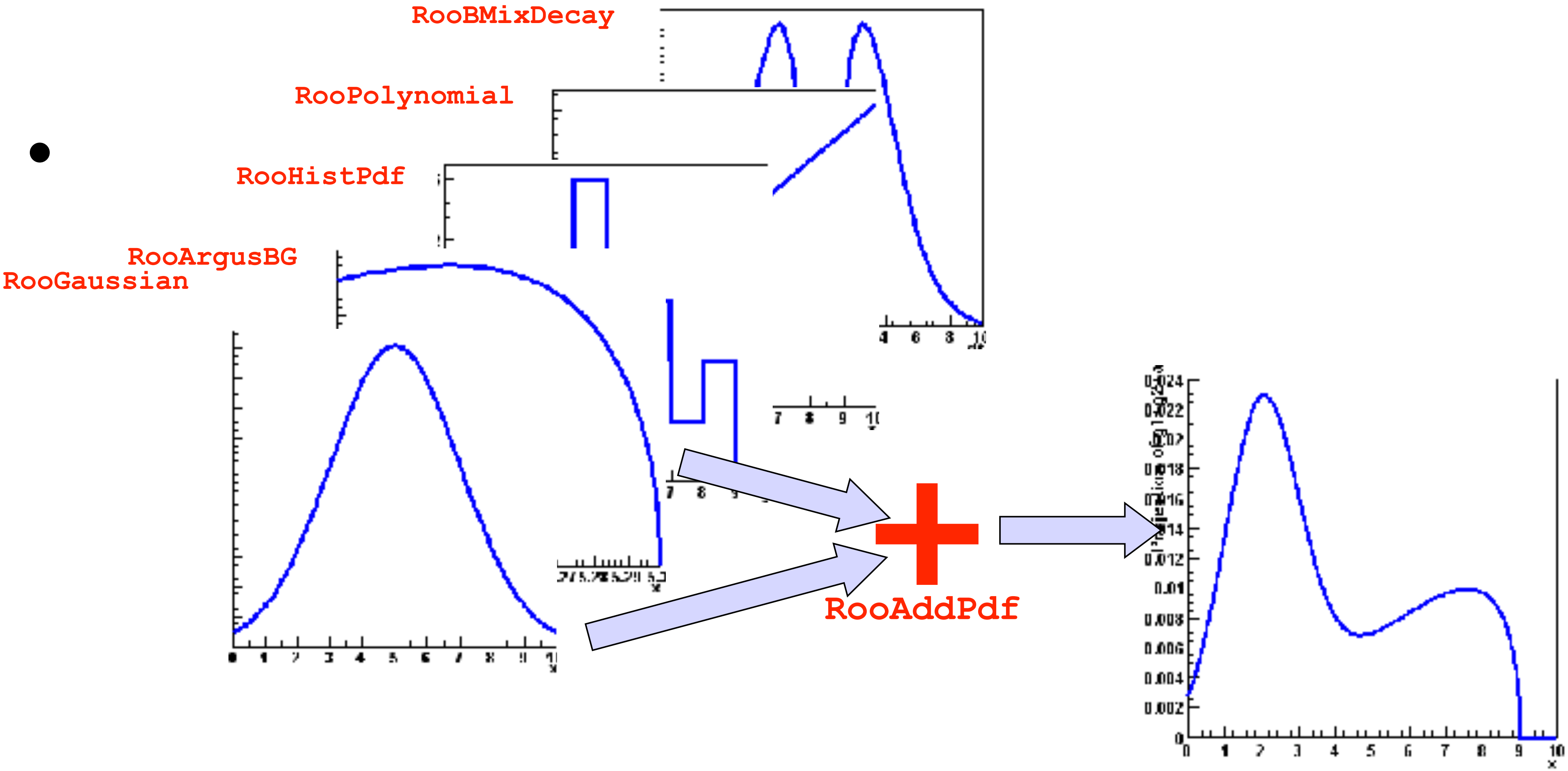
```
w.factory("expr::w('(1-D)/2',D[0,1])") ;
```

  - 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)
```

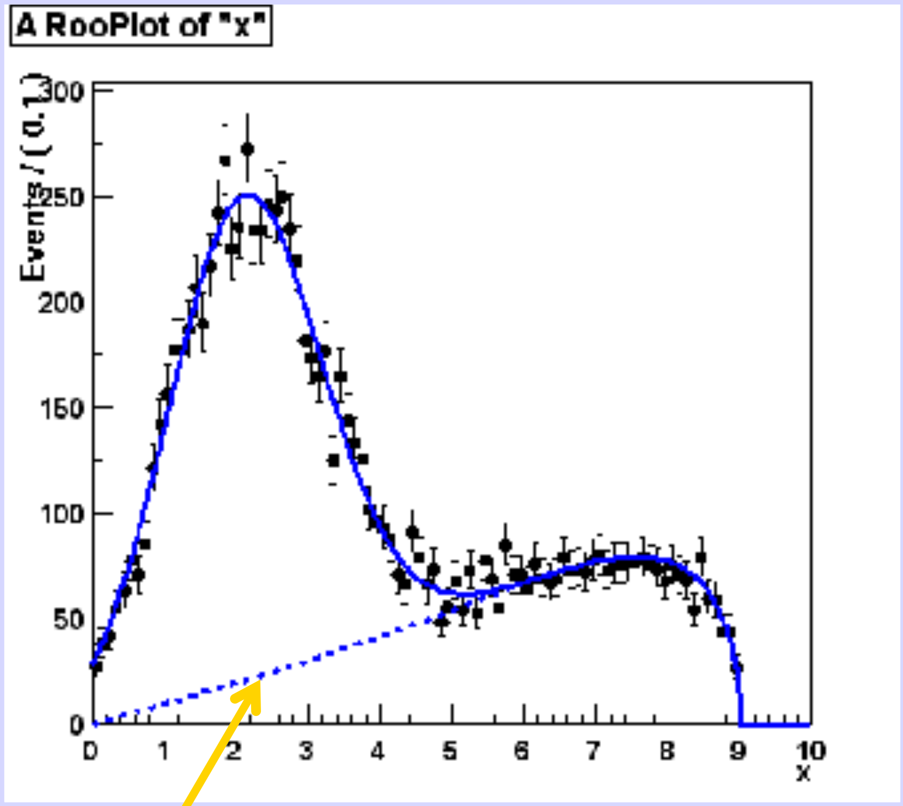
$$F(x) = \frac{N_S}{N_S + N_B} \times S(x) + \frac{N_B}{N_S + N_B} B(x) \quad ; \quad N_{\exp} = N_S + N_B$$

  - 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 | p) -> L(x|p)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

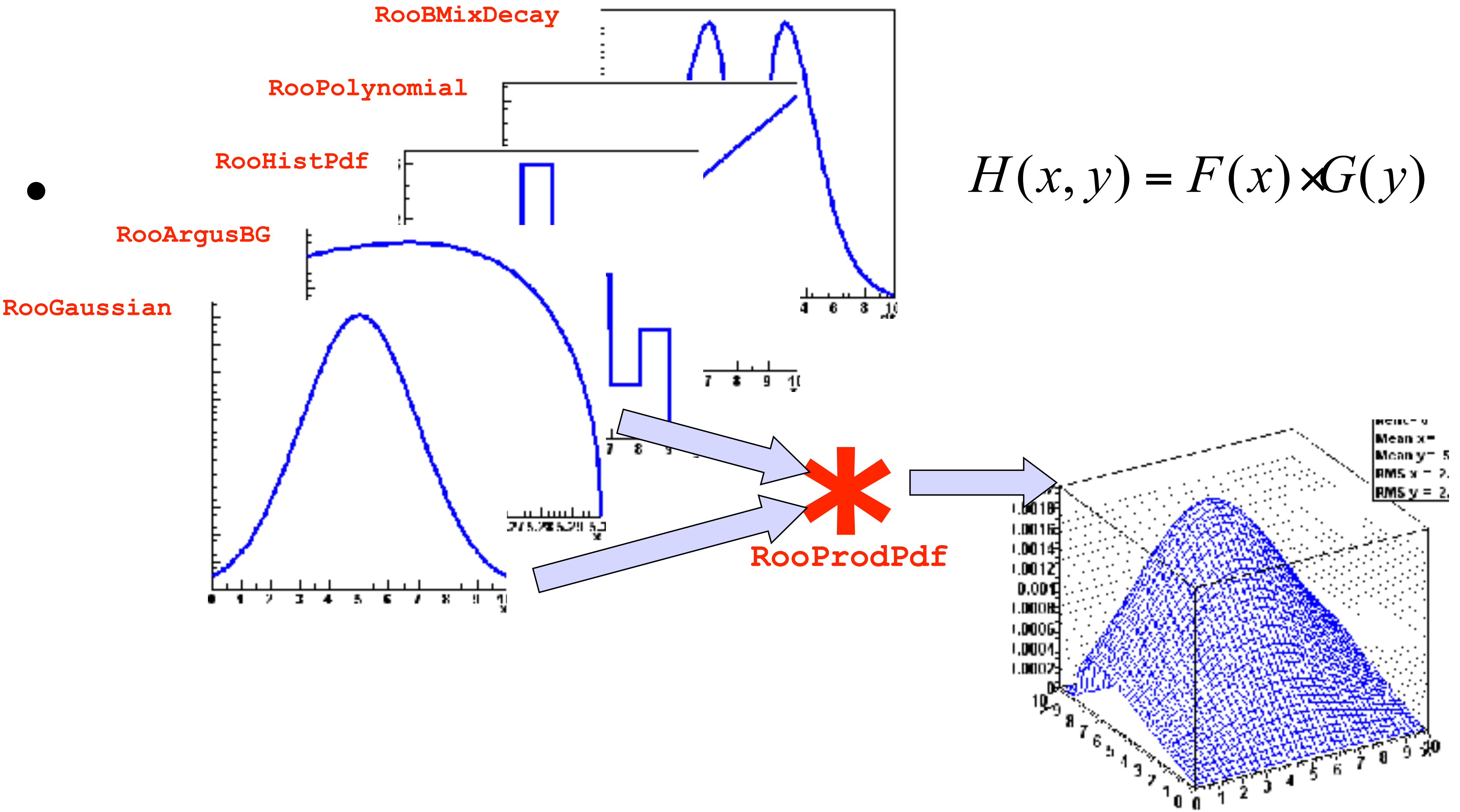



```
// Plot only argus components
w::sum.plotOn(frame,Components("argus"),LineStyle(kDashed)) ;

// 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** **nD**

$$H(x,y) = F(x) \times G(y) \qquad H(x^{\{i\}}) = \prod_i F^{\{i\}}(x^{\{i\}})$$

  - 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

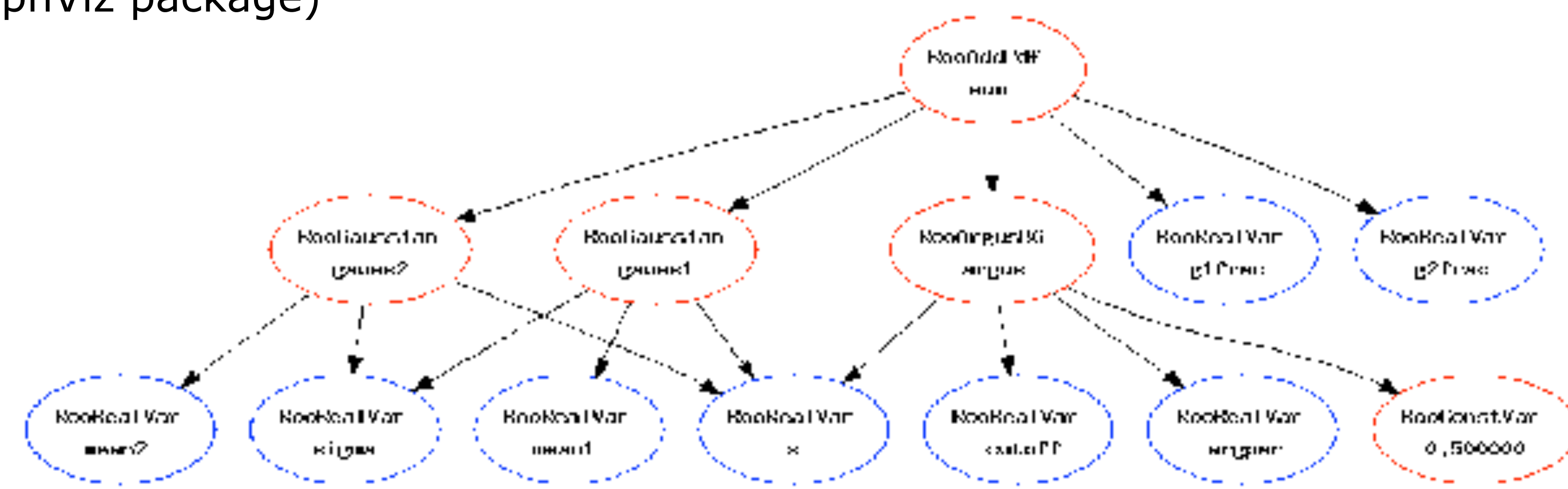
# Operations on specific to composite pdfs

- 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”)

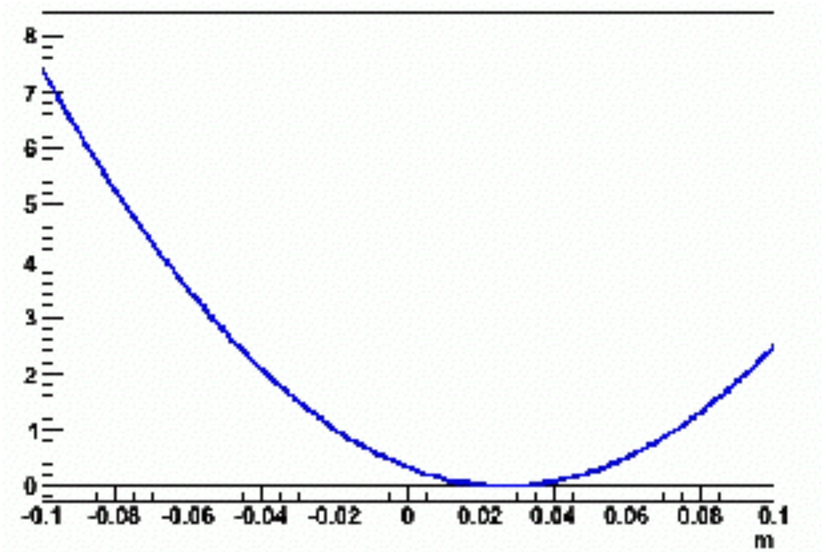
```
RooDataSet *dataA, *dataB ;
RooDataSet dataAB(“dataAB”,”dataAB”,
                  RooArgSet(*w.var(“x”),*w.cat(“index”)),
                  Index(*w.cat(“index”)),
                  Import(“A”,*dataA),Import(“B”,*dataB)) ;
```

# 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")) ;
```

# 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