Introduction to RooFit & RooStats

W. Verkerke (Nikhef/University of Amsterdam)

Introduction

Data analysis in HEP – typical goals:
* establish presence of signal in sample (usually with large bkg)
* measure properties of signal (usually with less background)

Statistical techniques help to achieve these goals allowing to formulate probabilistic statements on signal/background hypothesis that are used to model the data, and/or make statement on (signal) model parameters

Given a dataset D(x,y,z…)*: all statistical inference techniques require probability (density) models S(x,y,z), B_i(x,y,z) that describes the distribution of data under these hypothesis*, to be able to execute these techniques

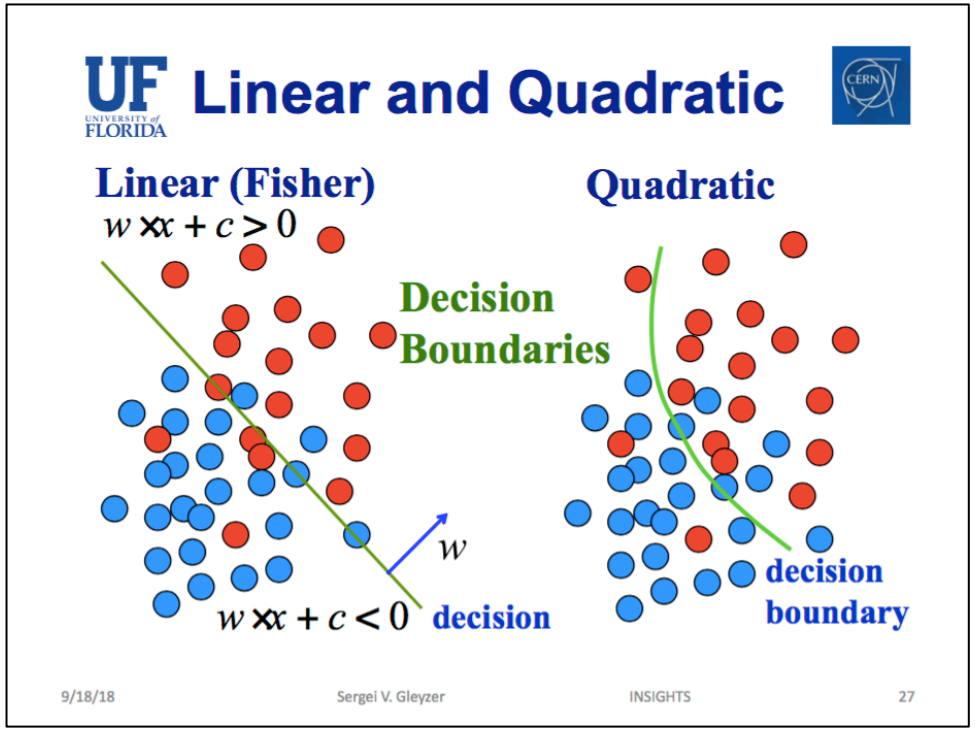
Modeling is key!

This tutorial is about practical building

Modeling in machine learning vs statistical inference

A complete model for S(x,y,z) and B(x,y,z) is not needed for all tasks.

For example in most Machine Learning techniques, only the decision boundary is parameterized



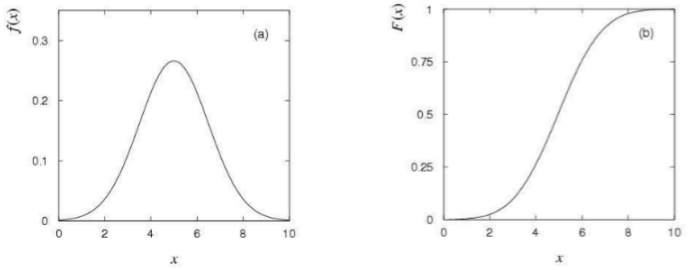
Modeling in machine learning vs statistical inference

- In HEP (ML-assisted) event classification is used to obtain purified data samples
- Final step is (some type of) statistical inference procedure to quantify confidence is signal hypothesis, and/or estimate physically relevent parameters of interest of the signal model
- For this last step, a complete and accurate model of both signal and background hypothesis are required

Cumulative distribution function

Probability to have outcome less than or equal to x is

$$\int_{-\infty}^{x} f(x')\,dx' \equiv F(x) \qquad \text{cumulative distribution function}$$



Alternatively define pdf with $f(x) = \dfrac{\partial F(x)}{\partial x}$

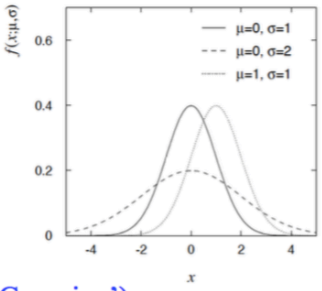
Gaussian distribution

The Gaussian (normal) pdf for a continuous r.v. x is defined by:

$$f(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$$

$E[x] = \mu$ (N.B. often μ, σ^2 denote mean, variance of any r.v., not only Gaussian.)

$V[x] = \sigma^2$



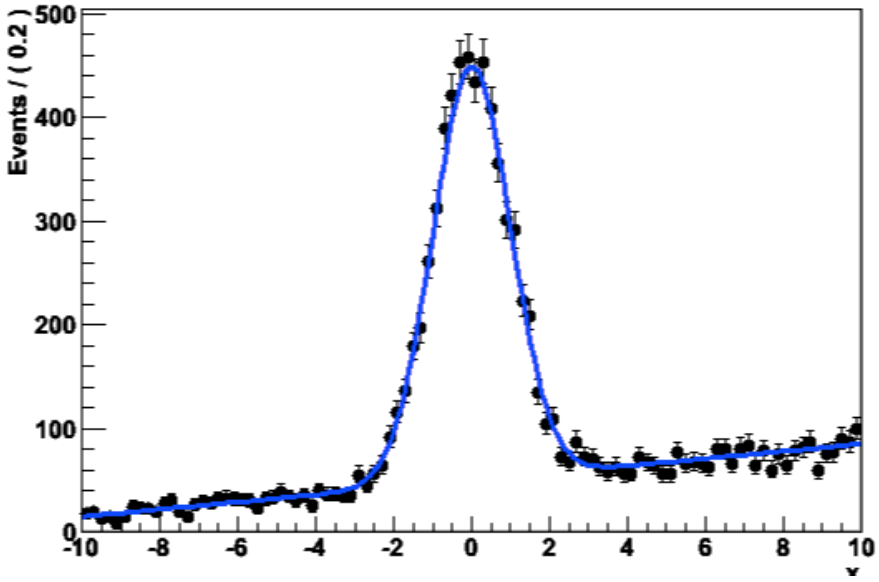
Special case: $\mu = 0$, $\sigma^2 = 1$ ('standard Gaussian'):

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}, \qquad \Phi(x) = \int_{-\infty}^{x} \varphi(x')\,dx'$$

If $y \sim$ Gaussian with μ, σ^2, then $x = (y - \mu)/\sigma$ follows $\varphi(x)$.

Modeling in statistical inference

- Accuracy of statistical inference depends strongly on quality of the models used.
- Building good models is a complex subject on the boundary of physics and statistics.
- Consider a simple example: a dataset with identically distributed and independent measurements of some observable x



Key question in data analysis:

What distribution in x do we assume for signal/background hypotheses?

Information from
- First principles (e.g. CLT)
- Detector/physics simulation
- Other/earlier measurements

supplemented with assumptions (educated guess)

Modeling in statistical inference

- Often information model signal/background is uncertain → quantify these with model parameters, constrained (in case external information is available) or unconstrained (if not)
- Sometimes information is available only in forms that do not trivially map to a prescription for a probability model
  - E.g. two datasets representing different implementation of an underlying physics process simulation
- Accurate model building is often challenging. Models that incorporate all that is known and/or uncertain can be become quite complex (describing anywhere between a handful up to thousands of distributions simultaneously)
- Good tools are important!

Roadmap for today and beyond

- Focus of todays session is RooFit – a C++ toolkit for OO modeling of probability (density) models that is suited to build both simple and highly complex models with a single interface
  - RooFit is part of ROOT and heavily used in particle physics
- Complementary to RooFit is RooStats, is a collection of tools to perform standard statistical tests on RooFit models (confidence intervals etc…)
  - Will only briefly touch on this today, given that most of the statistical techniques have not been covered yet.
- RooFit can describe both analytical probability density models and binned probability models based on template datasets (typically obtained from MC simulation)
- Will focus this afternoon on analytical models to explore RooFits general functionality
- In the December INSIGHTS training event will focus 3 days model on models based on template datasets ('profile likelihood'), statistical theory related to profile likelihood, and advanced model building topics (mapping systematic uncertainties to model parameters), statistical and technical debugging/validation of complex probability models etc.

RooFit

WV + D. Kirkby - 1999

1 Introduction & Overview

- *Introduction*
- *Some basics statistics*
- *RooFit design philosophy*

Wouter Verkerke, NIKHEF

Introduction – Purpose

Model the distribution of observables $\vec{x}$ in terms of

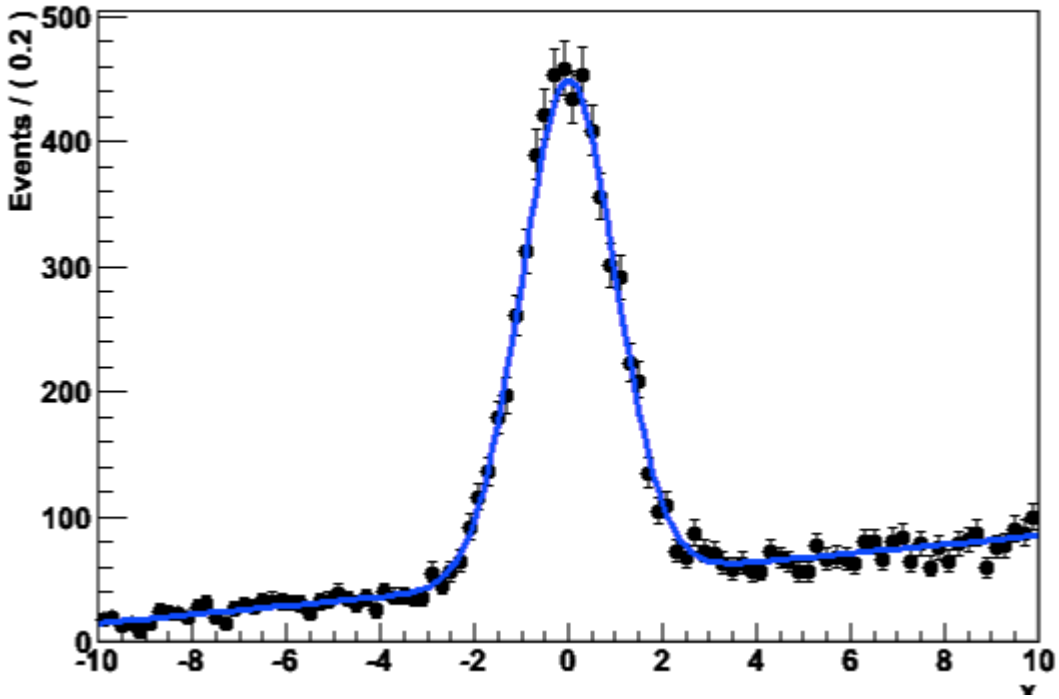
- Physical parameters of interest $\vec{p}$
- Other parameters $\vec{q}$ to describe detector effects (resolution,efficiency,…)

Probability density function $F(\vec{x};\vec{p},\vec{q})$

- normalized over allowed range of the observables **x** w.r.t the parameters **p** and **q**

Introduction -- Focus: coding a probability density function

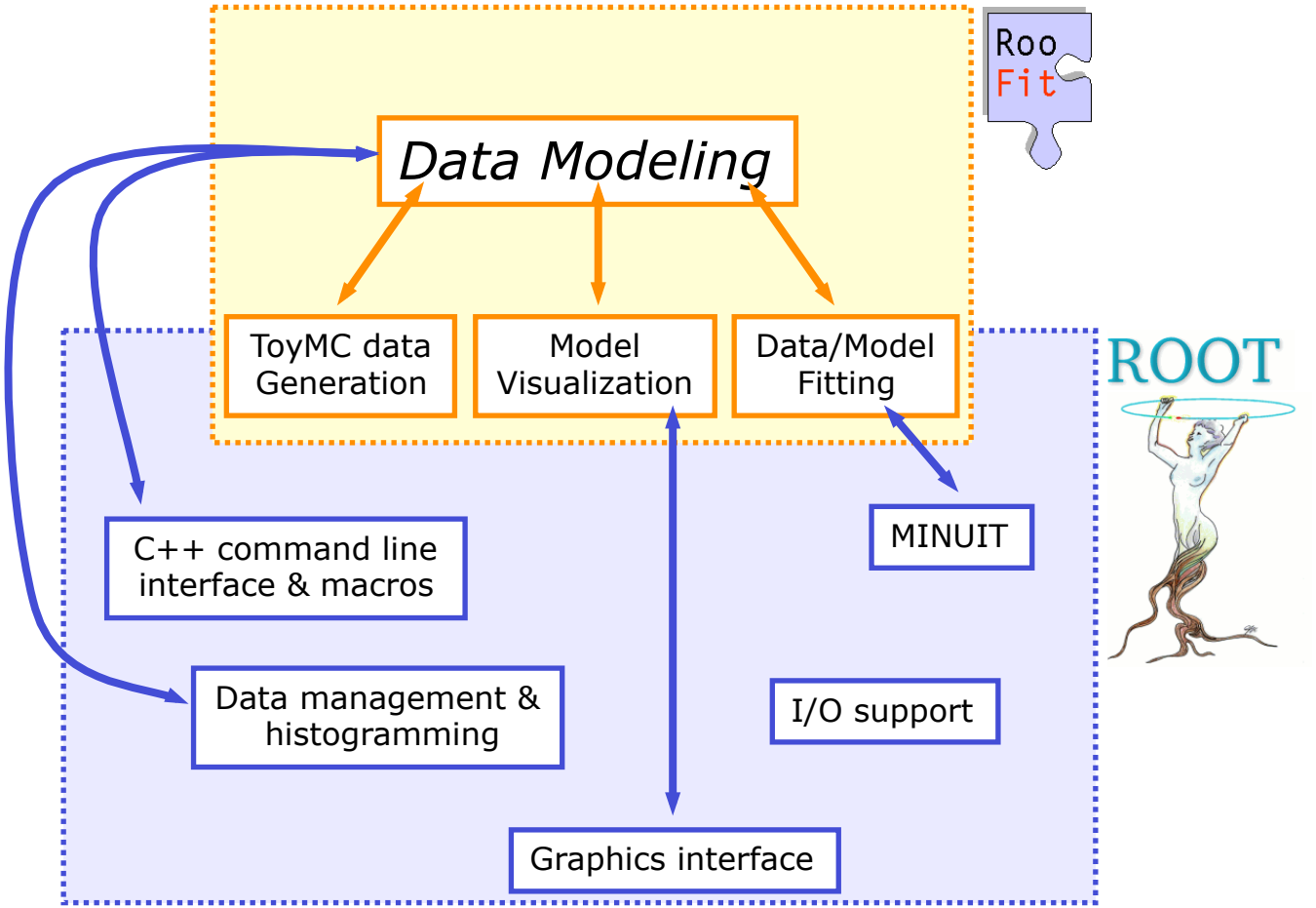
- Focus on one practical aspect of many data analysis in HEP: How do you formulate your p.d.f. in ROOT
  - For 'simple' problems (gauss, polynomial), ROOT built-in models well sufficient



  - But if you want to do unbinned ML fits, use non-trivial functions, or work with multidimensional functions you are quickly running into trouble

Introduction – Relation to ROOT

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



Introduction – Why RooFit was developed

- BaBar experiment at SLAC: Extract sin(2β) from time_ dependent CP violation of B decay: e^+e^- → Y(4s) → $B\bar{B}$
  - Reconstruct both Bs, measure decay time difference
  - Physics of interest is in decay time dependent oscillation

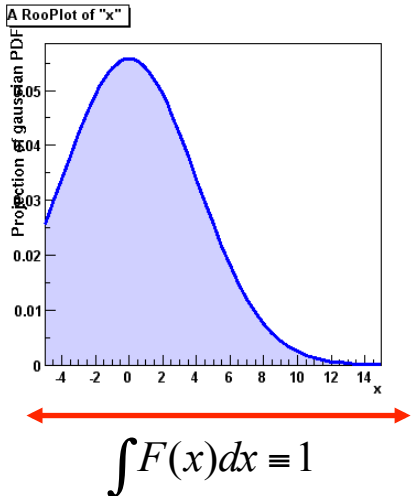
$$f_{sig} \cdot \Big[\mathrm{SigSel}(m;\bar{p}_{sig}) \cdot \big(\mathrm{SigDecay}(t;\vec{q}_{sig},\sin(2\beta)) \otimes \mathrm{SigResol}(t \mid dt;\vec{r}_{sig})\big)\Big] +$$
$$(1-f_{sig})\Big[\mathrm{BkgSel}(m;\bar{p}_{bkg}) \cdot \big(\mathrm{BkgDecay}(t;\vec{q}_{bkg}) \otimes \mathrm{BkgResol}(t \mid dt;\vec{r}_{bkg})\big)\Big]$$

- Many issues arise
  - Standard ROOT function framework clearly insufficient to handle such complicated functions → must develop new framework
  - Normalization of p.d.f. not always trivial to calculate → may need numeric integration techniques
  - Unbinned fit, >2 dimensions, many events → computation performance important → must try optimize code for acceptable performance
  - Simultaneous fit to control samples to account for detector performance

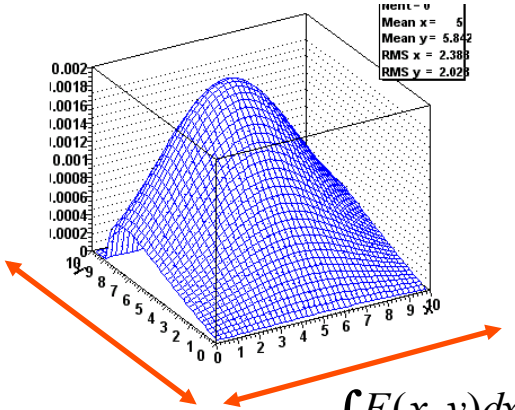
Mathematic – Probability density functions

- 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

RooFit core design philosophy

- Mathematical objects are represented as C++ objects

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

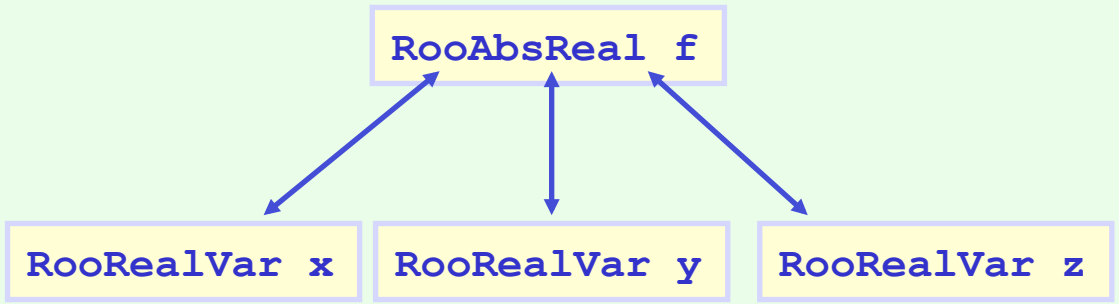
RooFit core design philosophy

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

Math

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

RooFit diagram



RooFit code

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

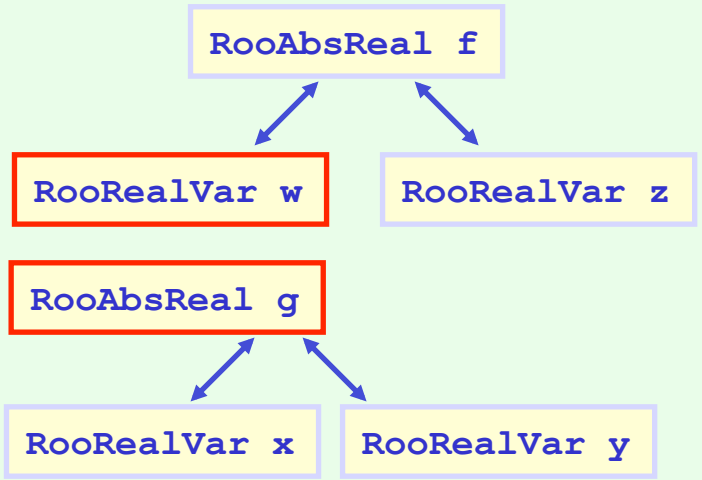
RooFit core design philosophy

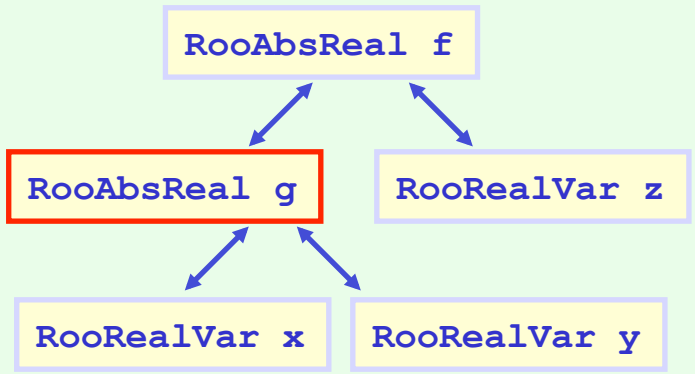
- Composite functions → Composite objects

Math

$f(w,z)$ $g(x,y)$ → $f(g(x,y),z) = f(x,y,z)$

RooFit diagram





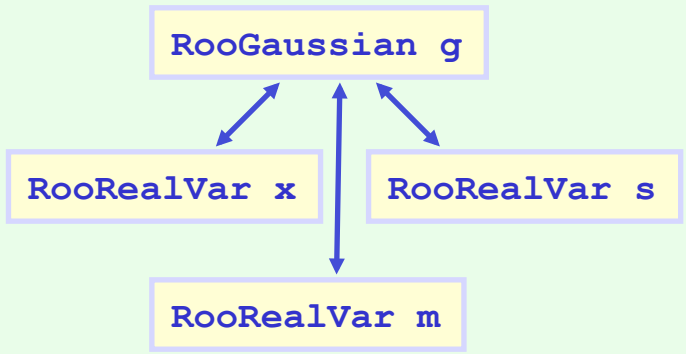
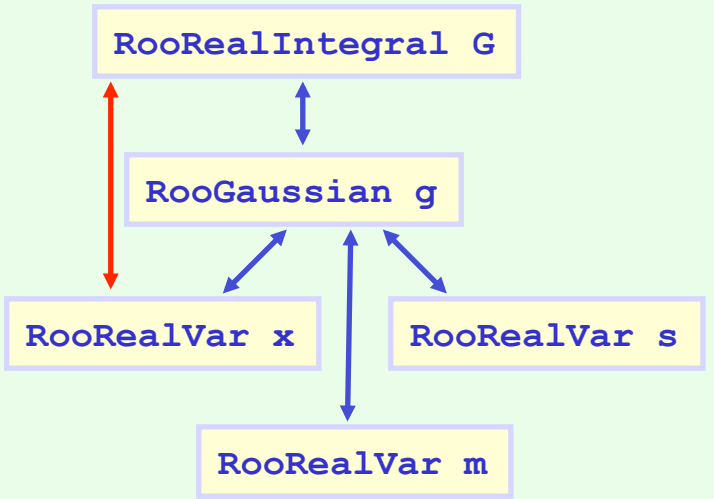
RooFit code

```
RooRealVar x("x","x",2) ;
RooRealVar y("y","y",3) ;
RooGooFunc g("g","g",x,y) ;

RooRealVar w("w","w",0) ;
RooRealVar z("z","z",5) ;
RooFooFunc f("f","f",w,z) ;
```

```
RooRealVar x("x","x",2) ;
RooRealVar y("y","y",3) ;
RooGooFunc g("g","g",x,y) ;

RooRealVar z("z","z",5) ;
RooFooFunc f("f","f",g,z) ;
```

RooFit core design philosophy

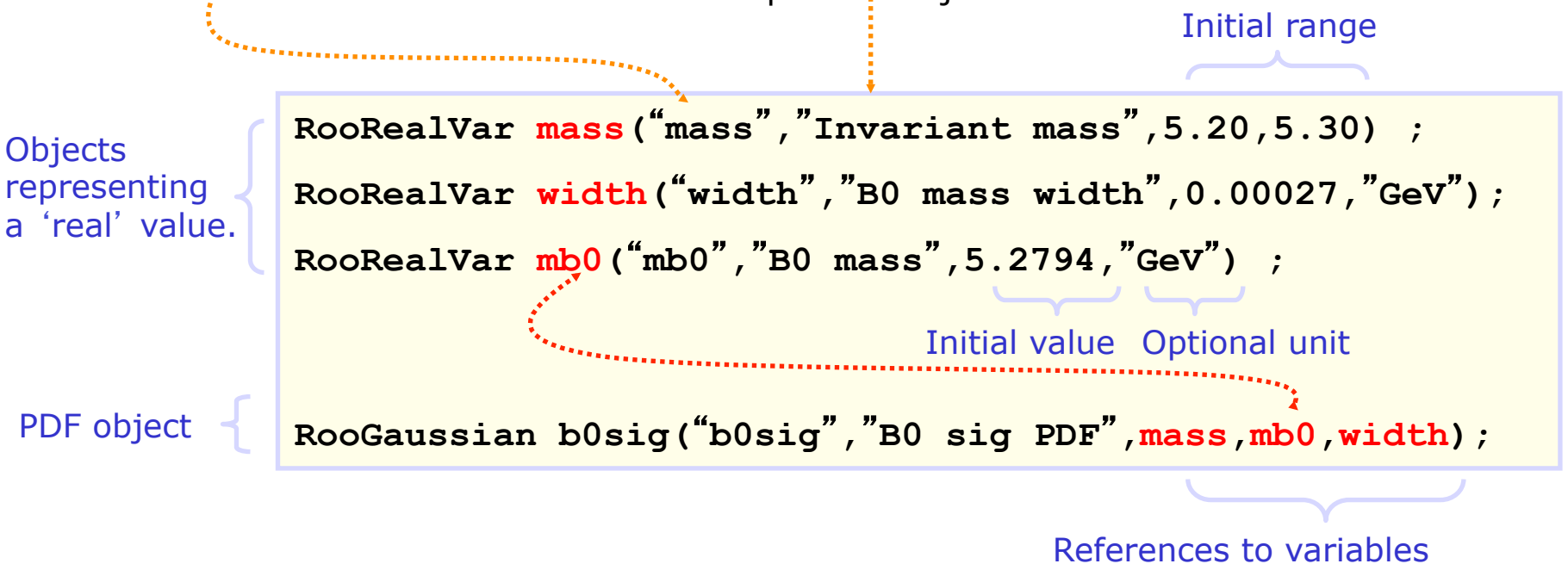
- Represent integral as an object,
  instead of representing integration as an action

Math	$g(x,m,s)$	$\int_{x_{\min}}^{x_{\max}} g(x,m,s)dx = G(m,s,x_{\min},x_{\max})$
RooFit diagram		

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)`
		`RooAbsReal *G =`
		`  g.createIntegral(x) ;`

Object-oriented data modeling

- In RooFit every variable, data point, function, PDF represented in a C++ object
  - Objects classified by data/function type they represent, not by their role in a particular setup
  - All objects are **self documenting**
    - **Name** - Unique identifier of object
    - **Title** – More elaborate description of object



RooFit designed goals for easy-of-use in macros

- Mathematical concepts mimicked as much as possible in class design
  - Intuitive to use
- Every object that can be constructed through composition should be fully functional
  - No implementation level restrictions
  - No zombie objects
- All methods must work on all objects
  - Integration, toyMC generation, etc
  - No half-working classes

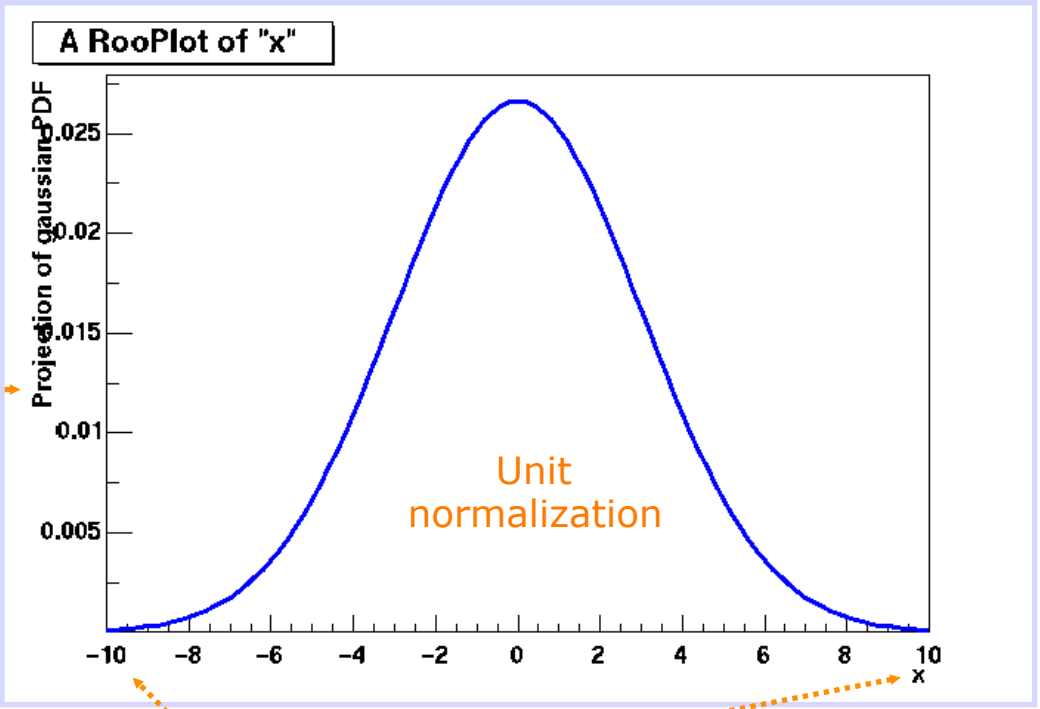
2 Basic Functionality

- *Creating a p.d.f*
- *Basic fitting, plotting, event generation*
- *Some details on normalization, event generation*
- *Library of basic shapes (including non-parametric shapes)*

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

Setup gaussian PDF and plot

```
// Build Gaussian PDF
RooRealVar x("x","x",-10,10) ;
RooRealVar mean("mean","mean of gaussian",0,-10,10) ;
RooRealVar sigma("sigma","width of gaussian",3) ;

RooGaussian gauss("gauss","gaussian PDF",x,mean,sigma) ;

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



Axis label from `gauss` title

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

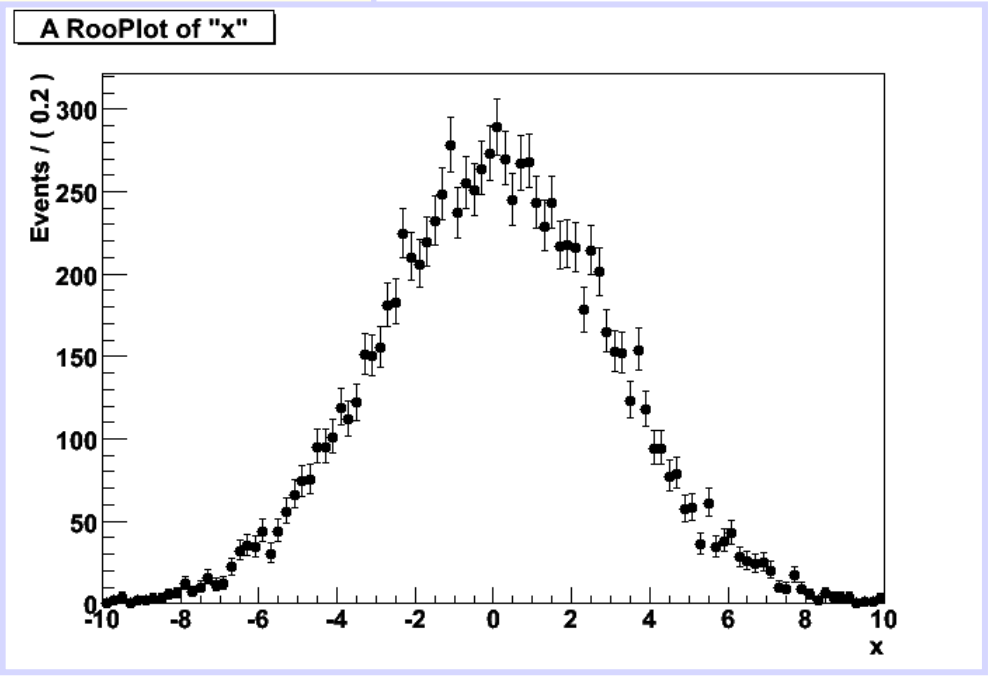
Plot range taken from limits of `x`

Basics – Generating toy MC events

demo1.cc

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

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

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

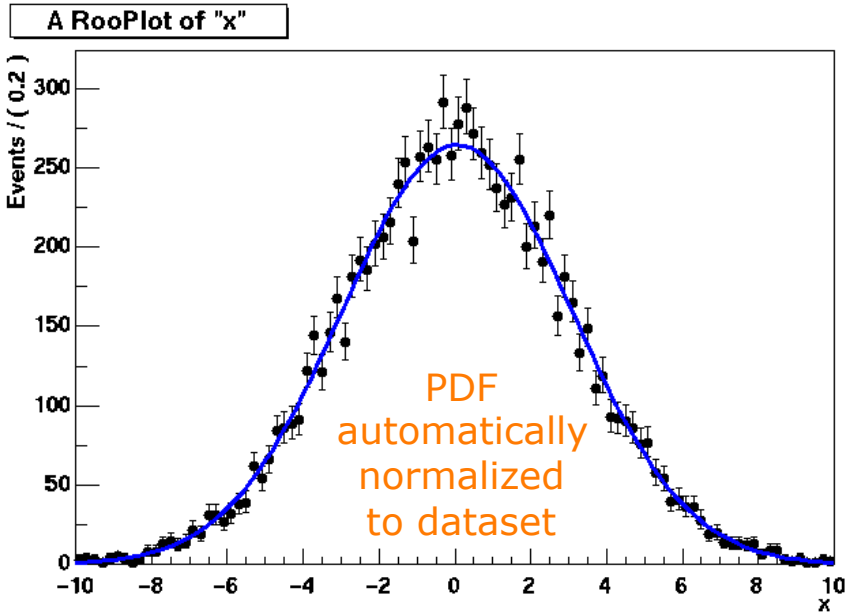


Returned dataset is ***unbinned*** dataset (like a ROOT TTree with a RooRealVar as branch buffer)

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

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

demo1.cc



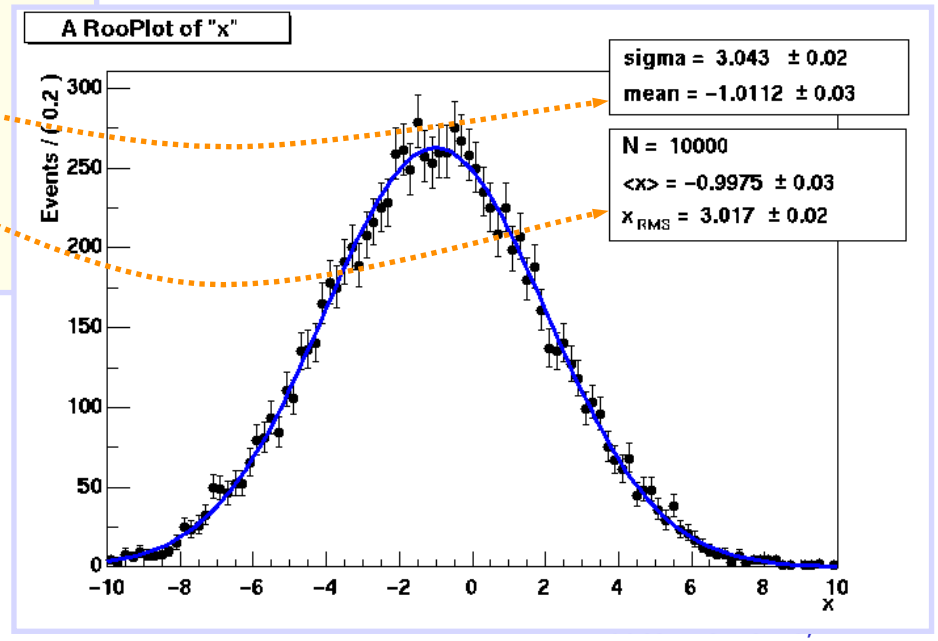

```
// ML fit of gauss to data
gauss.fitTo(*data) ;
(MINUIT printout omitted)

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

// Plot fitted PDF and toy data overlaid
RooPlot* xframe2 = x.frame() ;
data->plotOn(xframe2) ;
gauss.plotOn(xframe2) ;
xframe2->Draw() ;
```

Basics – RooPlot Decoration

- A RooPlot is an empty frame that can contain
  - RooDataSet projections
  - PDF and generic real-valued function projections
  - Any ROOT drawable object (arrows, text boxes etc)

- Adding a dataset statistics box / PDF parameter box

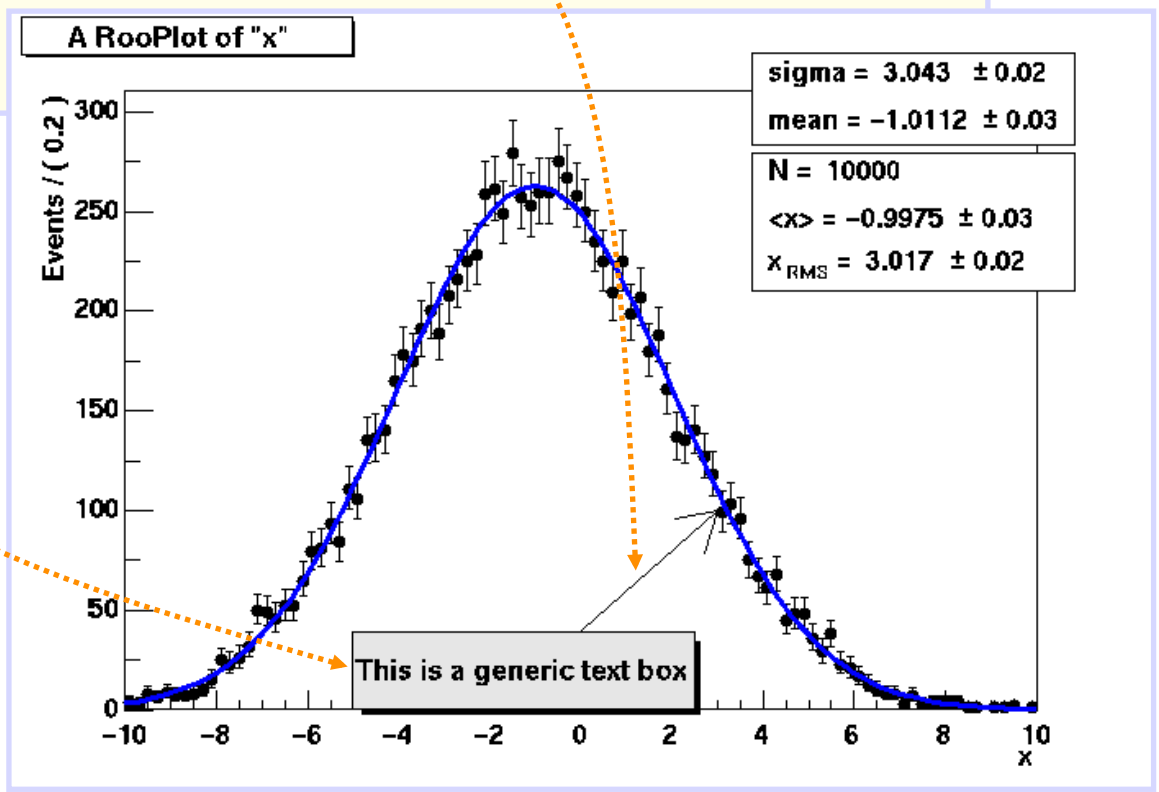
```
RooPlot* frame = x.frame() ;
data.plotOn(xframe) ;
pdf.plotOn(xframe) ;
pdf.paramOn(xframe,data) ;
data.statOn(xframe) ;
xframe->Draw() ;
```



Basics – RooPlot decoration

- Adding generic ROOT text boxes, arrows etc.

```
TPaveText* tbox = new TPaveText(0.3,0.1,0.6,0.2,"BRNDC");
tbox->AddText("This is a generic text box") ;
TArrow* arr = new TArrow(0,40,3,100) ;

xframe2->addObject(arr) ;
xframe2->addObject(tbox) ;
```



You can save a RooPlot with all its decorations in a ROOT file

Basics – Observables and parameters of Gauss

- Class **`RooGaussian`** has *no intrinsic notion* of distinction between observables and parameters

- Distinction always implicit in use context with dataset
  - **x** = observable (as it is a variable in the dataset)
  - **mean,sigma** = parameters

- Choice of observables (for unit normalization) always passed to **`gauss.getVal()`**

```
gauss.getVal() ;  // Not normalized (i.e. this is _not_ a pdf)
gauss.getVal(x) ; // Guarantees Int[xmin,xmax] Gauss(x,m,s)dx==1
gauss.getVal(s) ; // Guarantees Int[smin,smax] Gauss(x,m,s)ds==1
```

How does it work – Normalization

- Flexible choice of normalization facilitated by explicit normalization step in RooFit p.d.f.s

`gauss.getVal(x)`

$$g(\mathbf{x};m,s)=\frac{g(x,m,s)}{\int_{x_{\min}}^{x_{\max}} g(x,m,s)dx}$$

`gauss.getVal(s)`

$$g(\mathbf{s};m,x)=\frac{g(x,m,s)}{\int_{s_{\min}}^{s_{\max}} g(x,m,s)ds}$$

- Supporting class **`RooRealIntegral`** responsible for calculation of any

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

  - Negotiation with p.d.f on which (partial) integrals it can internally perform analytically
  - Missing parted are supplemented with numerical integration
  - Class **`RooRealIntegral`** can in principle integrate *everything.*

How does it work – Normalization

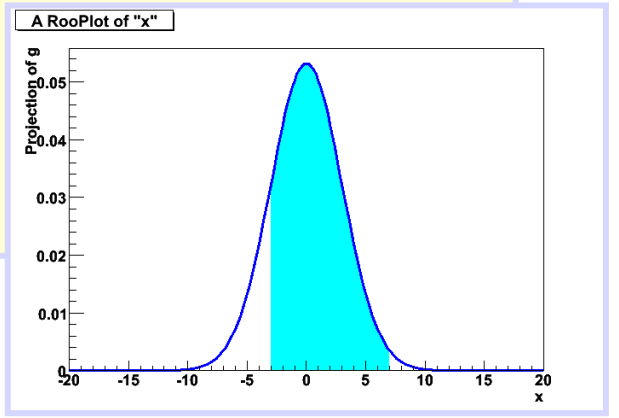
- A peak in the code of class **RooGaussian**

```
// Raw (unnormalized value) of Gaussian
Double_t RooGaussian::evaluate() const {
  Double_t arg= x - mean;
  return exp(-0.5*arg*arg/(sigma*sigma)) ;
}

// Advertise that x can be integrated internally
Int_t RooGaussian::getAnalyticalIntegral(RooArgSet& allVars,
      RooArgSet& analVars, const char* /*rangeName*/) const {
  if (matchArgs(allVars,analVars,x)) return 1 ;
  return 0 ;
}

// Implementation of analytical integral over x
Double_t RooGaussian::analyticalIntegral(Int_t code,
                                         const char* rname) const {
  static const Double_t root2 = sqrt(2.) ;
  static const Double_t rootPiBy2 = sqrt(atan2(0.0,-1.0)/2.0);
  Double_t xscale = root2*sigma;
  return rootPiBy2*sigma*(RooMath::erf((x.max(rname)-mean)/xscale)
                         -RooMath::erf((x.min(rname)-mean)/xscale));
}
```

Basics – Integrals over p.d.f.s

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

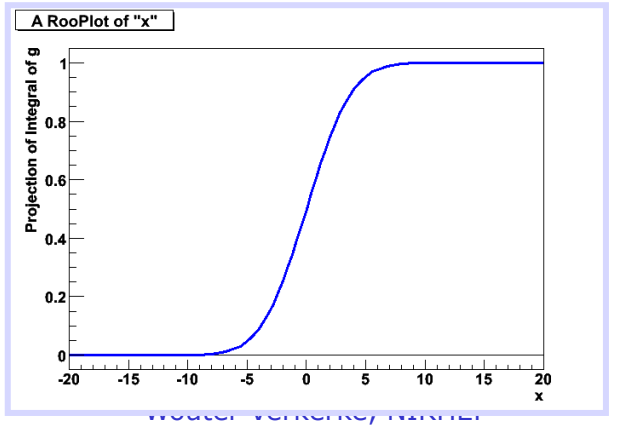
```
x.setRange("sig",-3,7) ;
RooAbsReal* ig = g.createIntegral(x,NormSet(x),Range("sig")) ;
cout << ig.getVal() ;
0.832519
mean=-1
cout << ig.getVal() ;
0.743677
```



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

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

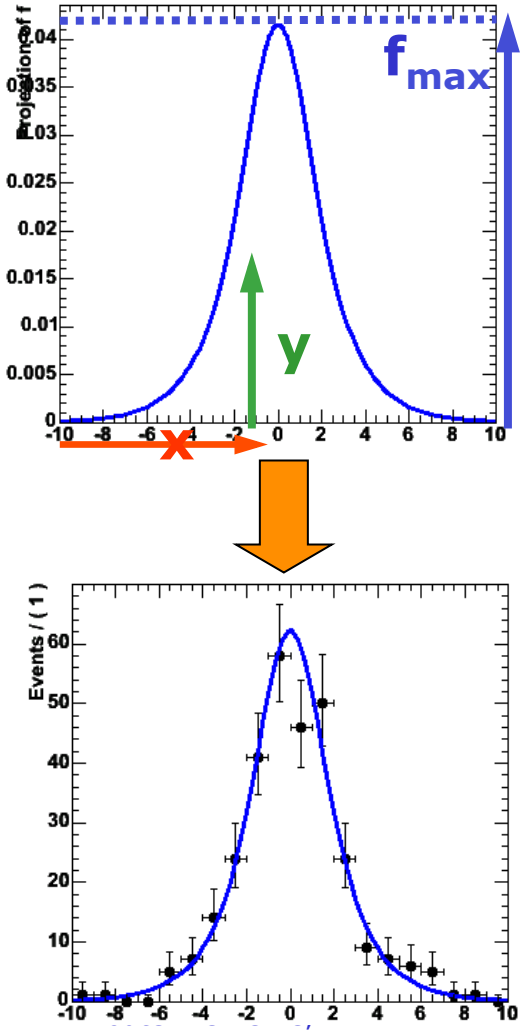
```
RooAbsReal* cdf = gauss.createCdf(x) ;
RooPlot* frame = x.frame() ;
cdf->plotOn(frame)->Draw() ;
```



How does it work – toy event generation

- By default RooFit implements an accept/reject sampling technique to generate toy events from a p.d.f.

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

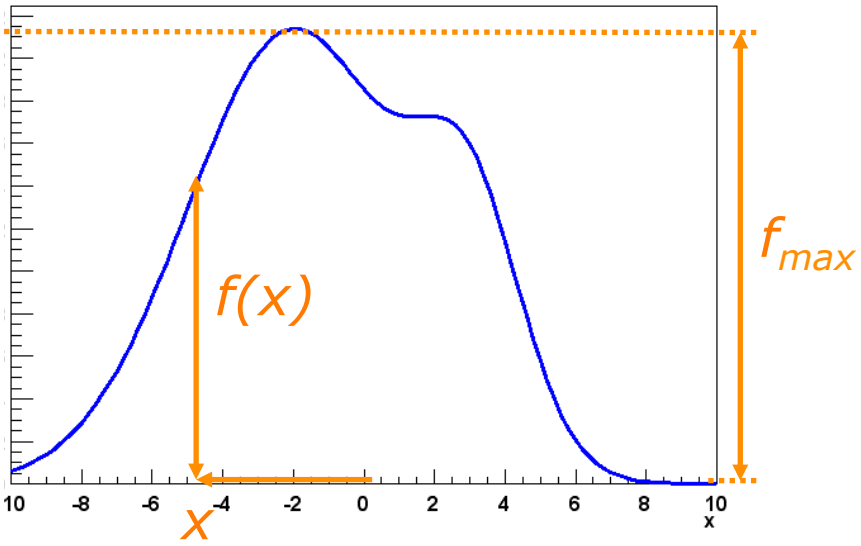


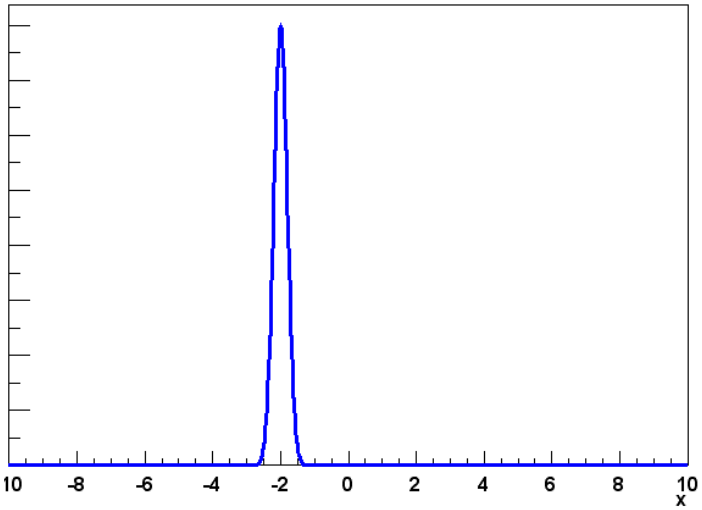
How does it work – toy event generation

- Accept/reject method can be very inefficient
  - Generating efficiency is

$$\frac{\int_{x_{max}}^{x_{min}} f(x)dx}{(x_{max} - x_{min}) \cdot f_{max}}$$

  - Efficiency is very low for narrowly peaked functions
  - Initial sampling for f_{max} requires very large trials sets in multiple dimension (~10000000 in 3D)

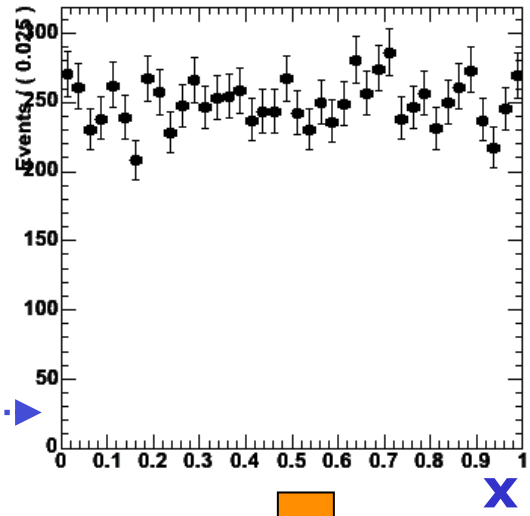




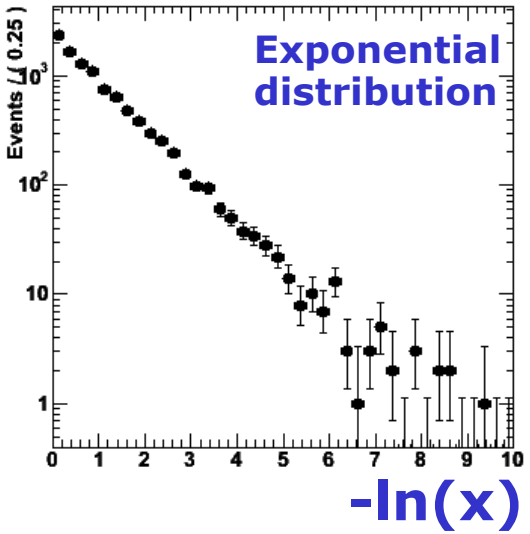
Toy MC generation – Inversion method

- Analoguous to integration, p.d.f can ad generator in case it can be done with a technique
- E.g. function inversion

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



Take –log(x)

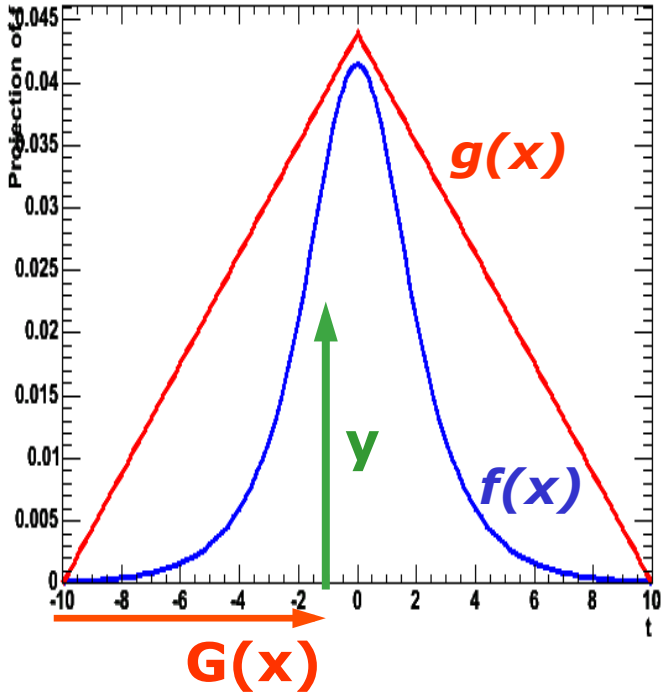


- Maximally efficient, but only works for class of p.d.f.s that is invertible

Toy MC generation – hybrid method

- Hybrid technique of importance sampling applicable to larger class of p.d.f.s

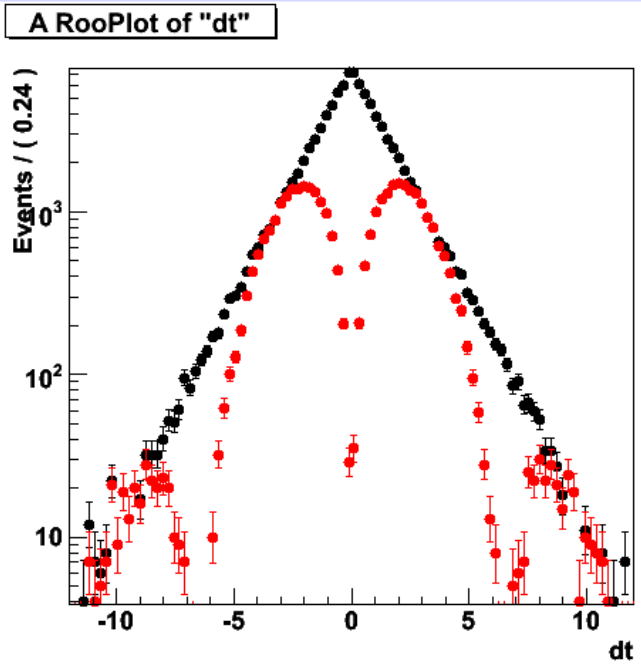
1) Find 'envelope function' g(x) that is invertible into G(x) and that fulfills g(x)>=f(x) for all x
2) Generate random number x from G using inversion method
3) Throw random number 'y'
4) If y<f(x)/g(x) keep x, otherwise return to step 2



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

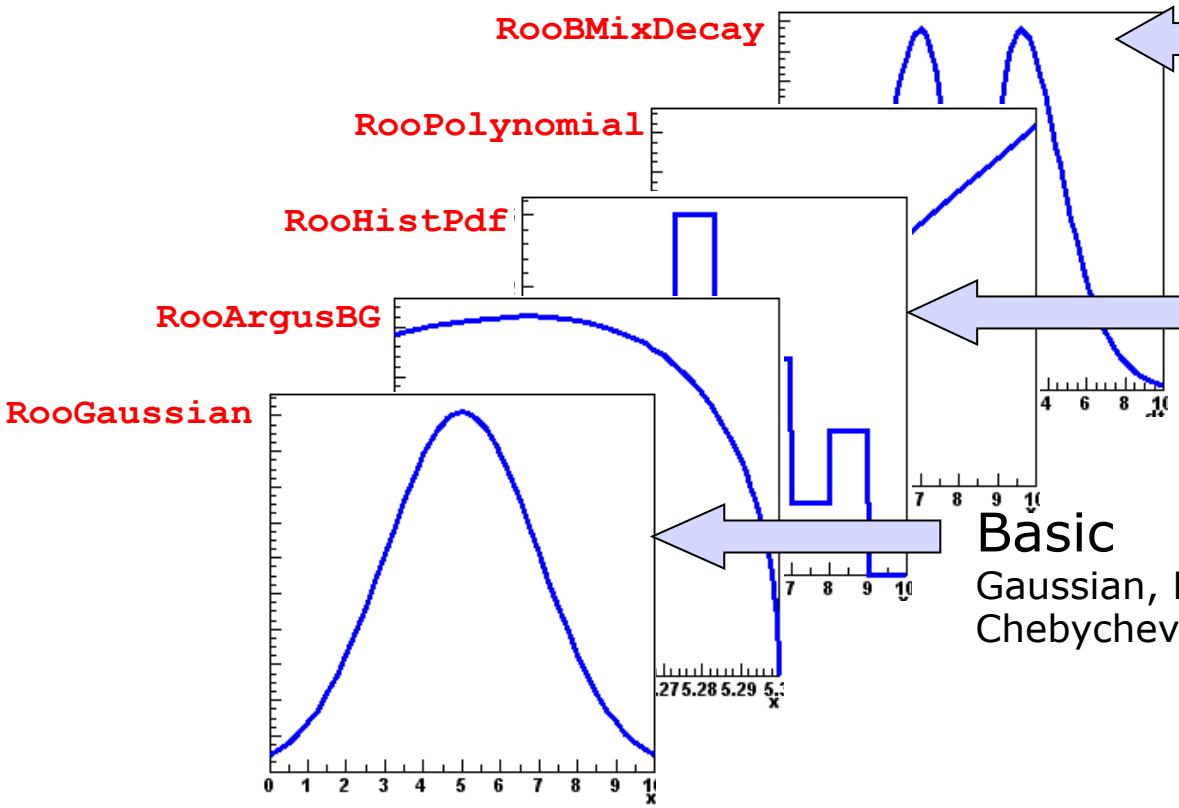
Toy MC generation – A peek inside RooBMixDecay

```
void RooBMixDecay::generateEvent(Int_t code) {
  while(1) {
    // Exponential decay envelope function through inversion
    Double_t rand = RooRandom::uniform() ;
    Double_t tval = -_tau*log(rand);

    // Importance sampling of envelope
    Double_t dil = 1-2.*mistag ;
    Double_t maxAcceptProb = 1 + TMath::Abs(delMistag) + TMath::Abs(dil) ;
    Double_t acceptProb = (1-tagFlav*delMistag) + _mixState*dil*cos(dm*tval);
    Bool_t mixAccept = maxAcceptProb*RooRandom::uniform() < acceptProb ? kTRUE : kFALSE ;

    // Accept event if t is in generated range
    if (tval<_t.max() && tval>_t.min() && mixAccept) {
      _t = tval ;
      break ;
    }
  }
}
```



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 – Generic expression-based PDFs

- If your favorite PDF isn't there
  and you don't want to code a PDF class right away
  → use **RooGenericPdf**
- Just write down the PDFs expression as a C++ formula

```
// PDF variables
RooRealVar x("x","x",-10,10) ;
RooRealVar y("y","y",0,5) ;
RooRealVar a("a","a",3.0) ;
RooRealVar b("b","b",-2.0) ;

// Generic PDF
RooGenericPdf gp("gp","Generic PDF","exp(x*y+a)-b*x",
                 RooArgSet(x,y,a,b)) ;
```

- Numeric normalization automatically provided

Model Building – Writing your own class

- Factory class exists (**`RooClassFactory`**) that can write, compile, link C++ code for RooFit p.d.f. and function classes
- **Example 1**:
  - Write class **`MyPdf`** with variable x,y,a,b in files **`MyPdf.h, MyPdf.cxx`**

```
RooClassFactory::makePdf("MyPdf","x,y,a,b");
```

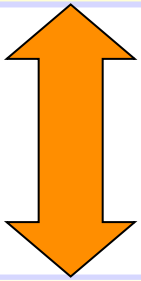
  - Only need to fill **`evaluate()`** method in **`MyPdf.cxx`** in terms of a,b,x
  - Can add optional code to support for analytical integration, internal event generation

Model Building – Writing your own class

- **Example 2**:
  - Functional equivalent to `RooGenericPdf`: Write class `MyPdf` with prefilled one-line function expression, compile and link p.d.f, create and return instance of class

Compiled code

```
RooAbsPdf* gp = RooClassFactory::makePdfInstance("gp",
                        "exp(x*y+a)-b*x",RooArgSet(x,y,a,b));
```



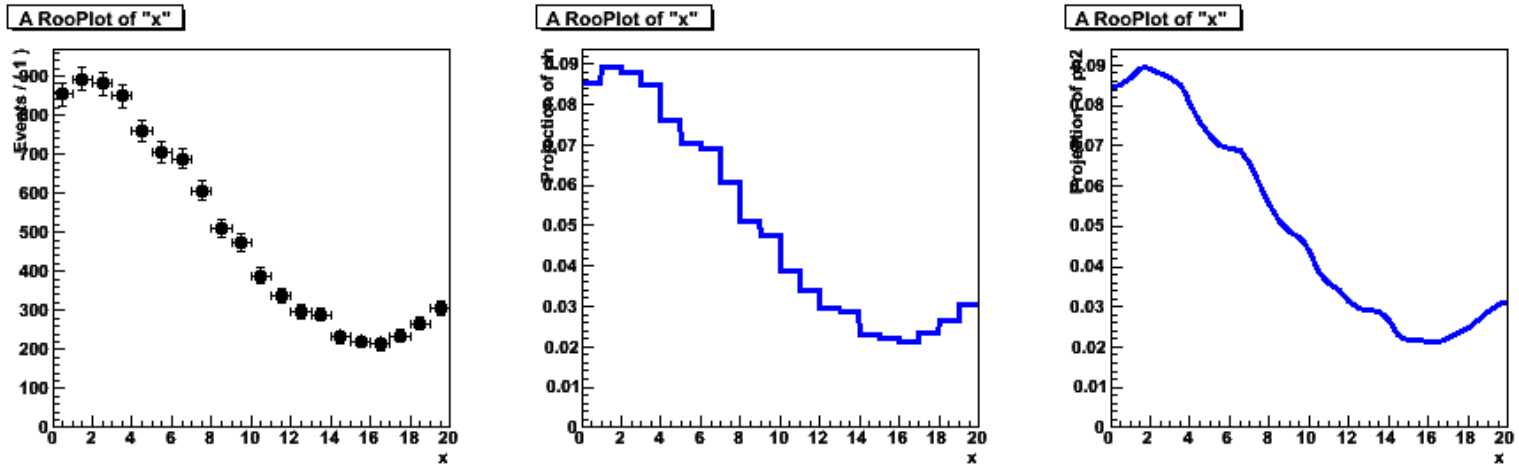
Interpreted code

```
RooGenericPdf gp("gp","Generic PDF","exp(x*y+a)-b*x",
                 RooArgSet(x,y,a,b)) ;
```

Highlight of non-parametric shapes - histograms

- Will highlight two types of non-parametric p.d.f.s
- Class **`RooHistPdf`** – a p.d.f. described by a histogram

`dataHist` **`RooHistPdf(N=0)`** **`RooHistPdf(N=4)`**




```
// Histogram based p.d.f with N-th order interpolation
RooHistPdf ph("ph","ph",x,*dataHist,N) ;
```

– Not so great at low statistics (especially problematic in >1 dim)

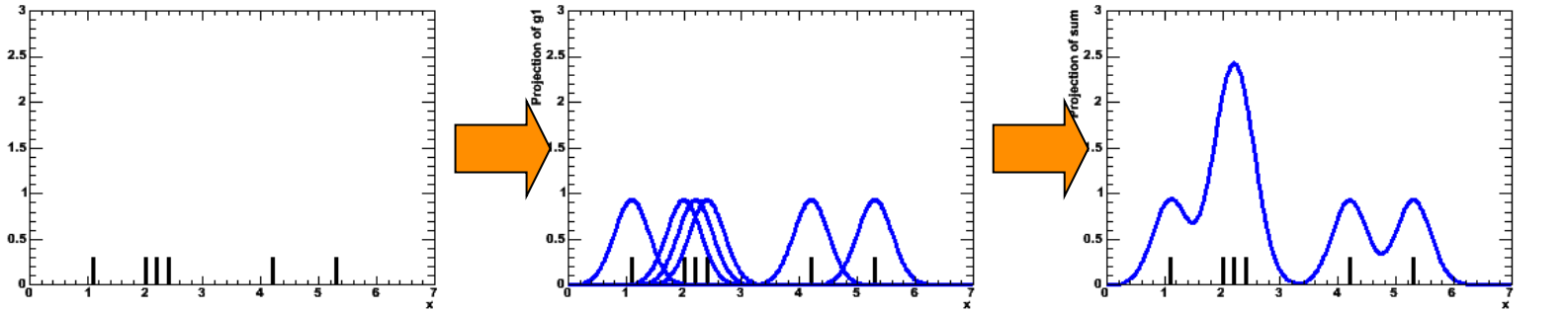
Highlight of non-parametric shapes – kernel estimation

- Class **`RooKeysPdf`** – A kernel estimation p.d.f.
  - Uses *unbinned* data
  - Idea represent each event of your MC sample as a Gaussian probability distribution
  - Add probability distributions from all events in sample

Sample of events

Gaussian probability distributions for each event

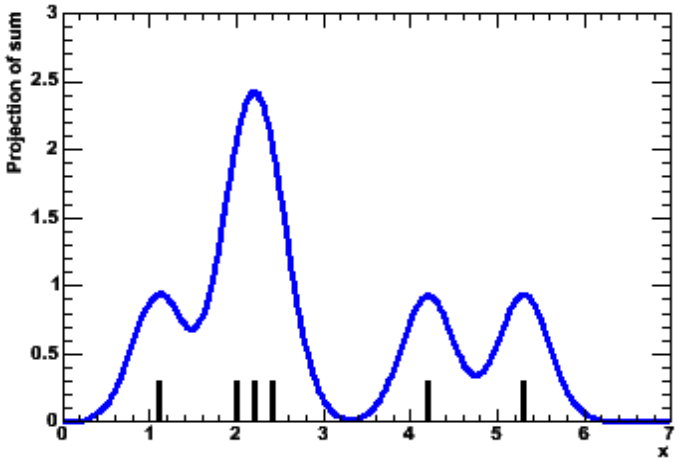
Summed probability distribution for all events in sample



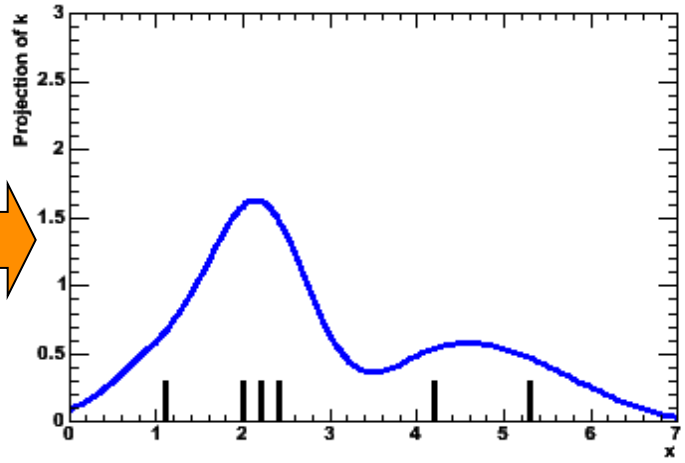
Highlight of non-parametric shapes – kernel estimation

- Width of Gaussian kernels need not be the same for all events
  - As long as each event contributes 1/N to the integral
- Idea: 'Adaptive kernel' technique
  - Choose wide Gaussian if local density of events is low
  - Choose narrow Gaussian if local density of events is high
  - Preserves small features in high statistics areas, minimize jitter in low statistics areas
  - Automatically calculated

Static Kernel (with of all Gaussian identical)



Adaptive Kernel (width of all Gaussian depends on local density of events)

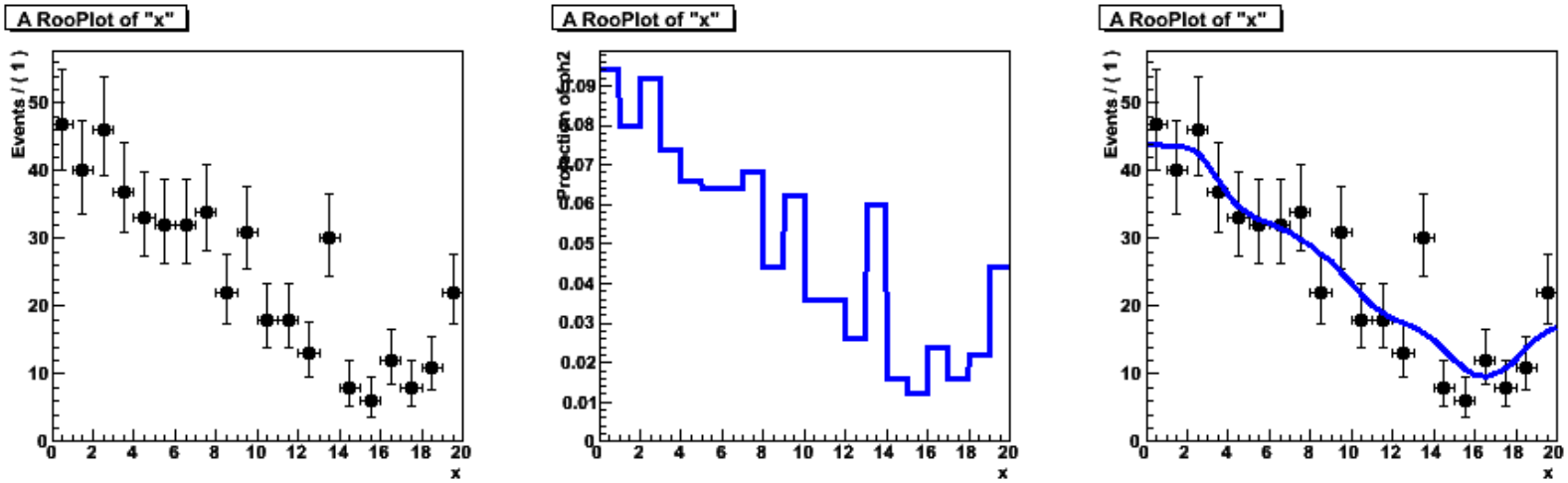


Highlight of non-parametric shapes – kernel estimation

- Example with comparison to histogram based p.d.f
  - Superior performance at low statistics
  - Can mirror input data over boundaries to reduce 'edge leakage'
  - Works also in >1 dimensions (class **RooNDKeysPdf**)

```
// Adaptive kernel estimation p.d.f
 RooKeysPdf k("k","k",x,*d,RooKeysPdf::MirrorBoth) ;
```

Data (N=500) **RooHistPdf(data)** **RooKeysPdf(data)**



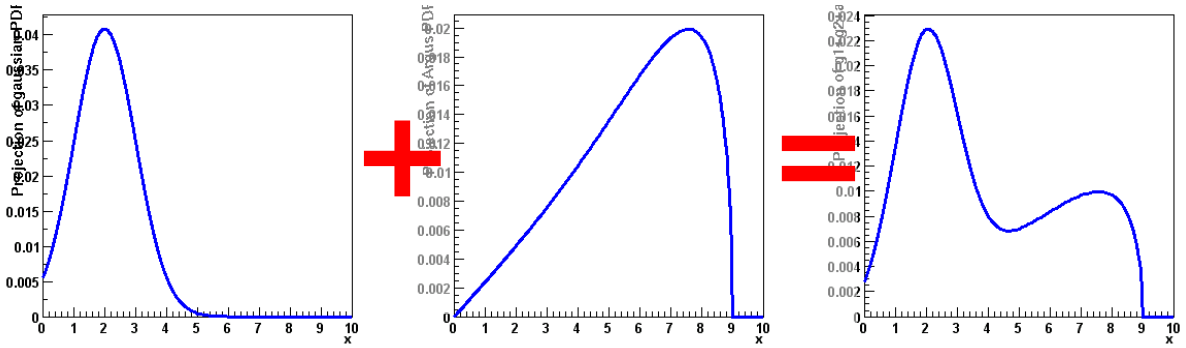
3 P.d.f. addition & convolution

- *Using the addition operator p.d.f*
- *Using the convolution operator p.d.f.*

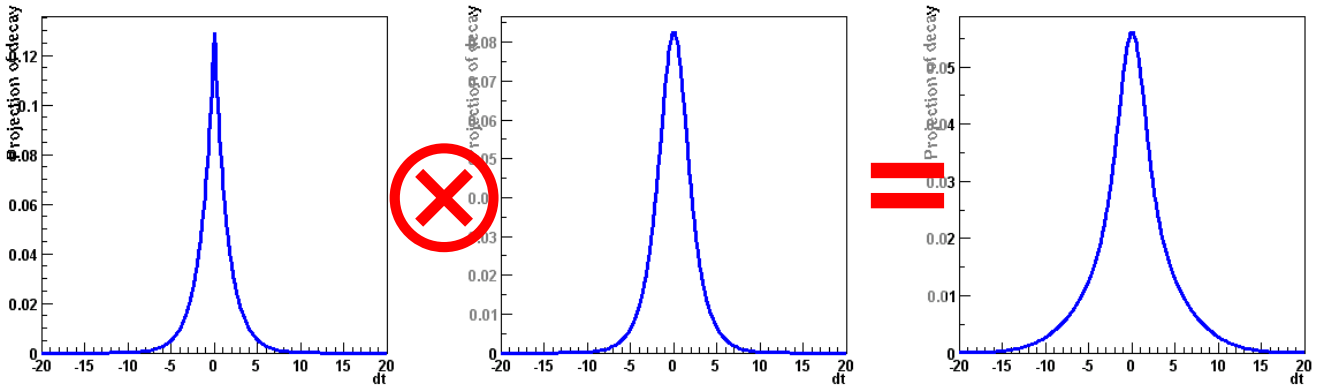
Wouter Verkerke, NIKHEF

Building realistic models

- Complex PDFs be can be trivially composed using operator classes
  - Addition

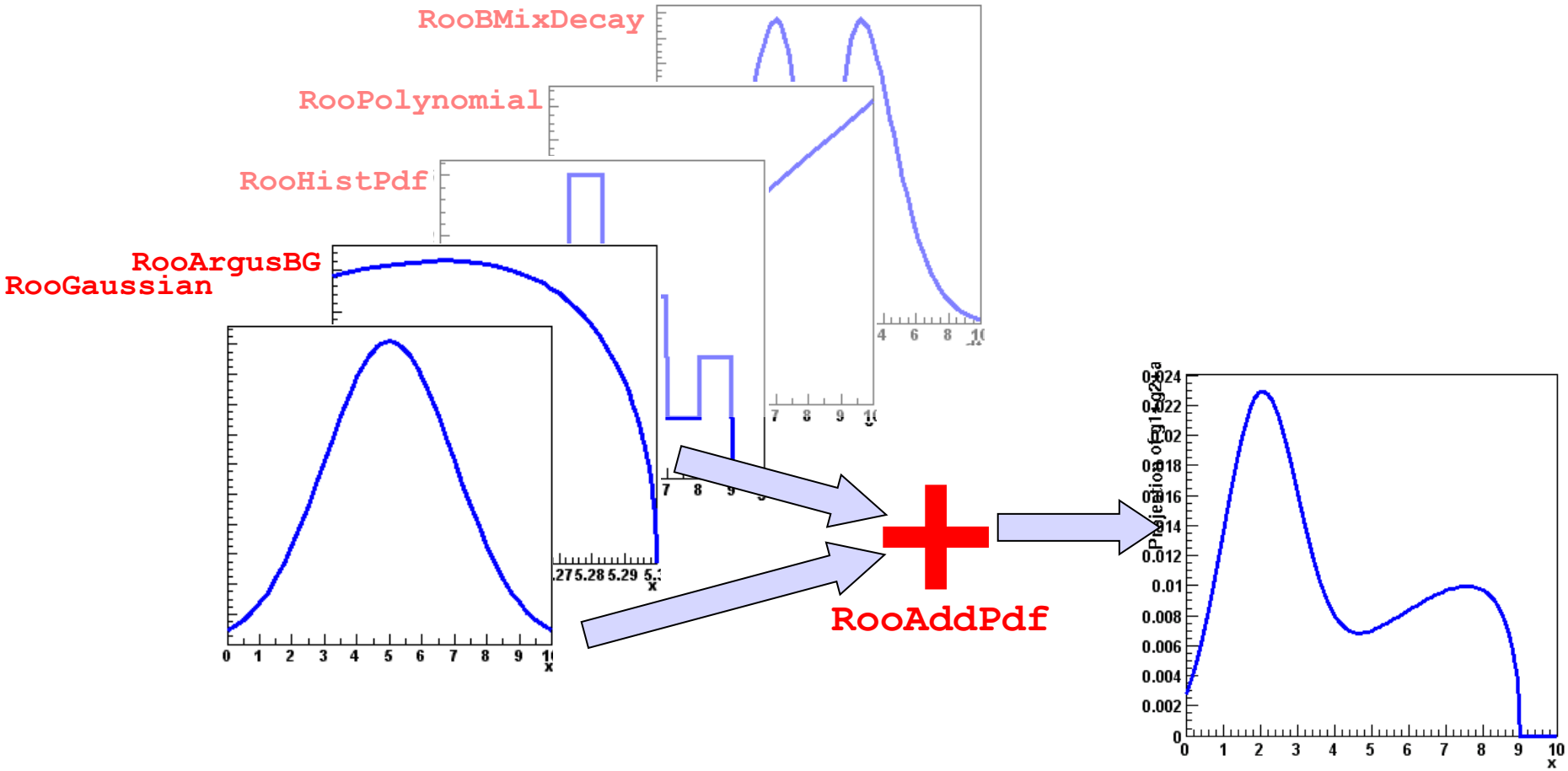


  - Convolution



Model building – (Re)using standard components

- Most realistic models are constructed as the sum of one or more p.d.f.s (e.g. signal and background)
- Facilitated through operator p.d.f **RooAddPdf**



Adding p.d.f.s – Mathematical side

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

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

  - Generically for *N* components P_0-P_N

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

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

Constructing a sum of p.d.f.s

RooAddPdf constructs the sum of N PDFs with N-1 coefficients:

$$S = c_0 P_0 + c_1 P_1 + c_2 P_2 + \ldots + c_{n-1} P_{n-1} + \left(1 - \sum_{i=0,n-1} c_i\right) P_n$$

Build 2 Gaussian PDFs

Build ArgusBG PDF

```
// Build two Gaussian PDFs
RooRealVar x("x","x",0,10) ;
RooRealVar mean1("mean1","mean of gaussian 1",2) ;
RooRealVar mean2("mean2","mean of gaussian 2",3) ;
RooRealVar sigma("sigma","width of gaussians",1) ;
RooGaussian gauss1("gauss1","gaussian PDF",x,mean1,sigma) ;
RooGaussian gauss2("gauss2","gaussian PDF",x,mean2,sigma) ;

// Build Argus background PDF
RooRealVar argpar("argpar","argus shape parameter",-1.0) ;
RooRealVar cutoff("cutoff","argus cutoff",9.0) ;
RooArgusBG argus("argus","Argus PDF",x,cutoff,argpar) ;

// Add the components
RooRealVar g1frac("g1frac","fraction of gauss1",0.5) ;
RooRealVar g2frac("g2frac","fraction of gauss2",0.1) ;
RooAddPdf  sum("sum","g1+g2+a",RooArgList(gauss1,gauss2,argus),
                               RooArgList(g1frac,g2frac)) ;
```

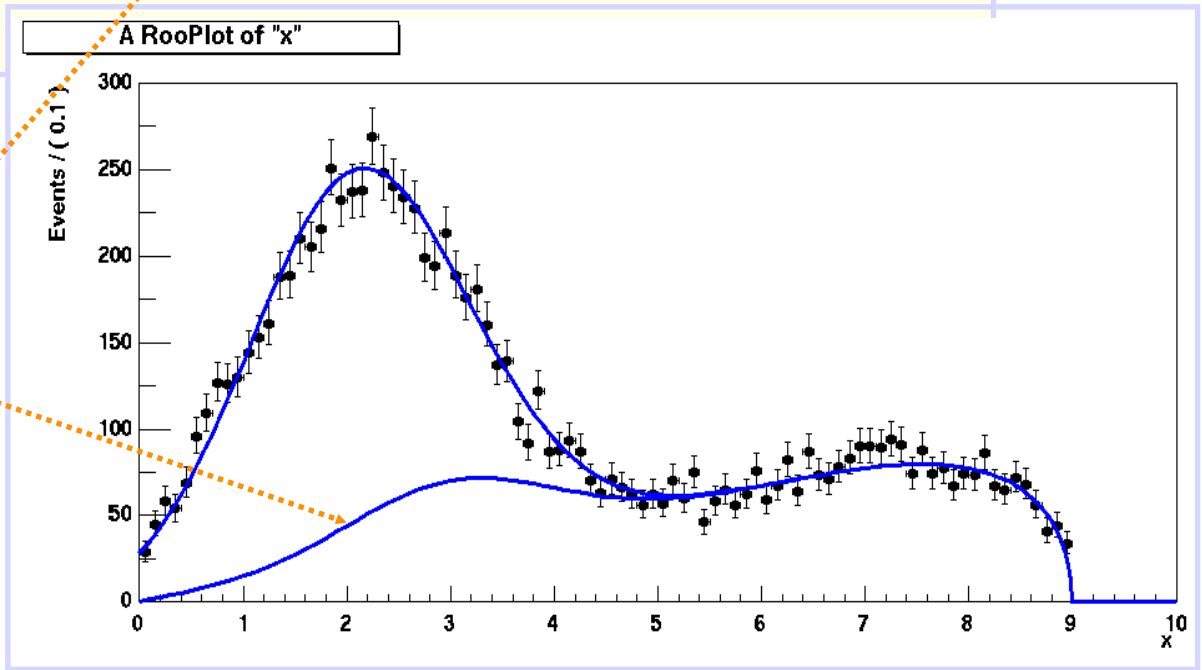
List of PDFs

List of coefficients

Plotting a sum of p.d.f.s, and its components

```
// Generate a toyMC sample
RooDataSet *data =
     sum.generate(x,10000) ;

// Plot data and PDF overlaid
RooPlot* xframe = x.frame() ;
data->plotOn(xframe) ;
sum->plotOn(xframe) ;

// Plot only argus and gauss2
sum->plotOn(xframe,Components(RooArgSet(argus,gauss2))) ;
xframe->Draw() ;
```

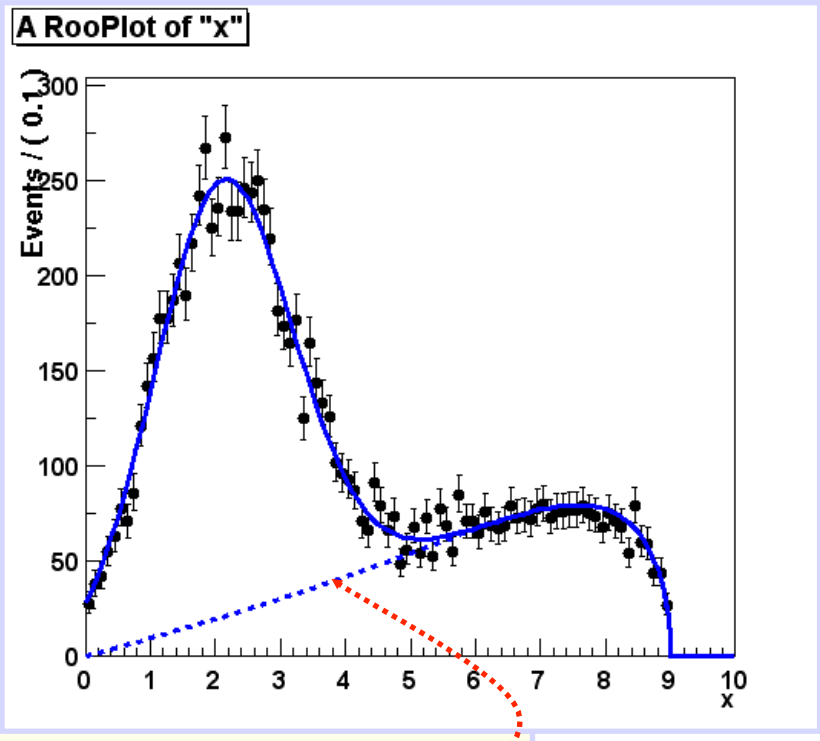
Plot selected components of a `RooAddPdf`



Component plotting - Introduction

- Also special tools for plotting of components in RooPlots
  - Use Method `Components()`

- Example:
  Argus + Gaussian PDF




```
// Plot data and full PDF first

// Now plot only argus component
sum->plotOn(xframe,
            Components(argus), LineStyle(kDashed)) ;
```

Component plotting – Selecting components

There are various ways to select single or multiple components to plot

Can refer to components either by name or reference

```
// Single component selection
pdf->plotOn(frame,Components(argus)) ;
pdf->plotOn(frame,Components("gauss")) ;


// Multiple component selection
pdf->plotOn(frame,Components(RooArgSet(pdfA,pdfB))) ;
pdf->plotOn(frame,Components("pdfA,pdfB")) ;



// Wild card expression allowed
pdf->plotOn(frame,Components("bkgA*,bkgB*")) ;
```

Recursive fraction form of RooAddPdf

- Fitting a sum of >2 p.d.f.s can pose some problems as the sum of the coefficients $f_1 \ldots f_{N-1}$ may become >1
  - This results in a negative remainder component ($\equiv 1-\Sigma_i f_i$)
  - Composite p.d.f may still be positive definite, but interpretation less clear
  - Could set limits on fractions f_i to avoid $\Sigma f_i > 1$ scenario, but where to put limits?
- Viable alternative to write as sum of *recursive* fractions

$$S_2(x) = f_1 P_1(x) + (1-f_1) P_2(x)$$
$$S_3(x) = f_1 P_1(x) + (1-f_1)\left(f_2 P_2(x) + (1-f_2) P_3(x)\right)$$
$$S_4(x) = f_1 P_1(x) + (1-f_1)\left(f_2 P_2(x) + (1-f_2)\left(f_3 P_3(x) + (1-f_3) P_4(x)\right)\right)$$

```
// Add the components with recursive fractions
RooAddPdf  sum("sum","fA*a+(fG*g1+g2)",RooArgList(a,g1,g2),
                                       RooArgList(afrac,gfrac),kTRUE) ;
```

Extended p.d.f form of RooAddPdf

- If extended ML term is introduced, we can fit expected number of events (N_{exp}) in addition to shape parameters
- In case of sum of p.d.f.s it is convenient to *re-parameterize* sum of p.d.f.s.

$$\begin{pmatrix} f_{sig} \\ N_{\exp} \end{pmatrix} \Rightarrow \begin{pmatrix} N_{sig} \equiv f_{sig} N_{\exp} \\ N_{bkg} \equiv (1 - f_{sig}) N_{\exp} \end{pmatrix}$$

- This transformation is applied automatically in **RooAddPdf** if equal number of p.d.f.s and coefs are given

```
RooRealVar nsig("nsig","number of signal events",100,0,10000) ;
RooRealVar nbkg("nbkg","number of backgnd events",100,0,10000) ;
RooAddPdf sume("sume","extended sum pdf",RooArgList(gauss,argus),
                                         RooArgList(nsig,nbkg)) ;
```

General features of extended p.d.f.s

- Extended term –log(Poisson(N_{obs},N_{exp})) is not added by default to likelihood
  - Use the `Extended()` argument to fit to have it added

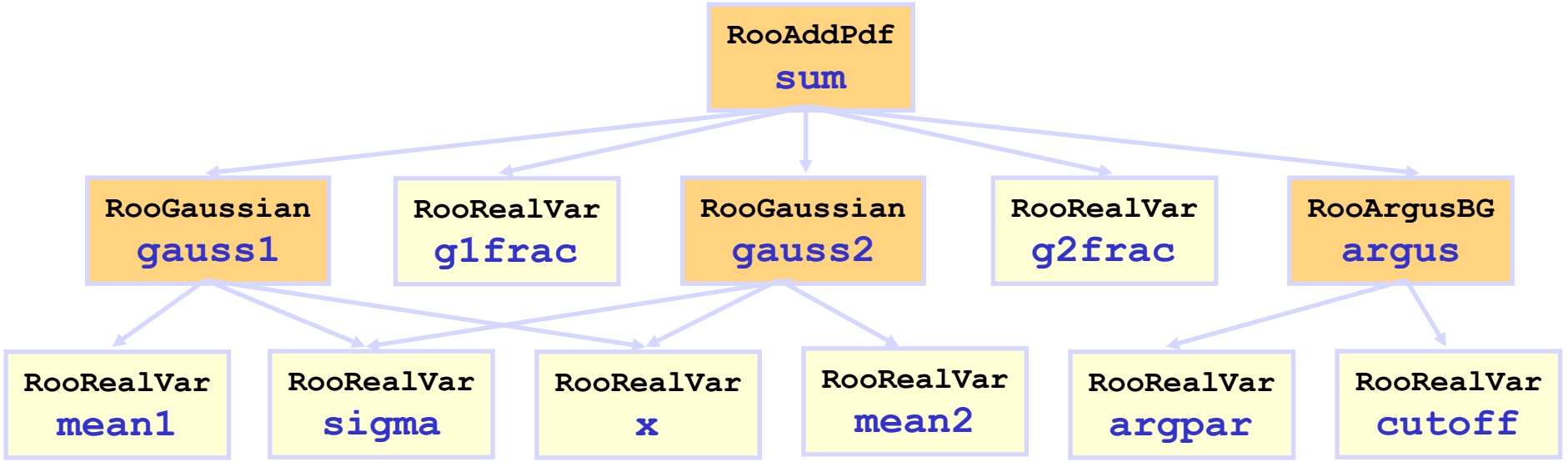
```
// Regular maximum likelihood fit
pdf.fitTo(*data) ;

// Extended maximum likelihood fit
pdf.fitTo(*data,Extended(kTRUE)) ;
```

- If p.d.f. is extended, N_{exp} is default number of events to generate

```
// Generate pdf.expectedEvents() events
RooDataSet* data = pdf.generate(x) ;


// Generate 1000 events
RooDataSet* data = pdf.generate(x,1000) ;
```

Dealing with composite p.d.f.s

- A RooAddPdf is an example of a composite p.d.f
  - The value of the sum is represented by a *tree* of components



  - The compositeness of a p.d.f. is completely transparent to most high-level operations
  - Can e.g. do `sum->fitTo(*data)` or `sum->generate(x,1000)` without being aware of composite nature of p.d.f.

Dealing with composite p.d.f.s

- The observables reported by a composite p.d.f and the 'leaf' of the expression tree
  - For example, request for list of parameters of composite sum, will return parameters of components of sum

```
RooArgSet *paramList = sum.getParameters(data) ;
paramList->Print("v") ;
RooArgSet::parameters:
  1) RooRealVar::argpar : -1.00000 C
  2) RooRealVar::cutoff :  9.0000 C
  3) RooRealVar::g1frac :  0.50000 C
  4) RooRealVar::g2frac :  0.10000 C
  5) RooRealVar::mean1  :  2.0000 C
  6) RooRealVar::mean2  :  3.0000 C
  7) RooRealVar::sigma  :  1.0000 C
```

- In general, composite p.d.f.s work *exactly the same* as basic p.d.f.s.

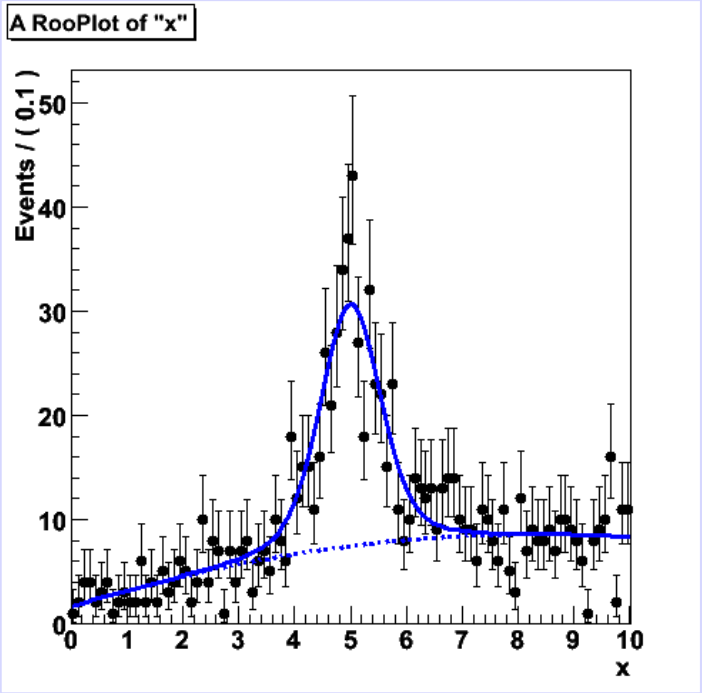
Visualization tools for composite objects

- Special tools exist to visualize the tree structure of composite objects
  - On the command line

```
Root> sum.Print("t") ;
0x927b8d0 RooAddPdf::sum (g1+g2+a)  [Auto]
  0x9254008 RooGaussian::gauss1 (gaussian PDF)  [Auto] V
    0x9249360 RooRealVar::x (x) V
    0x924a080 RooRealVar::mean1 (mean of gaussian 1) V
    0x924d2d0 RooRealVar::sigma (width of gaussians) V
  0x9267b70 RooRealVar::g1frac (fraction of gauss1) V
  0x9259dc0 RooGaussian::gauss2 (gaussian PDF)  [Auto] V
    0x9249360 RooRealVar::x (x) V
    0x924cde0 RooRealVar::mean2 (mean of gaussian 2) V
    0x924d2d0 RooRealVar::sigma (width of gaussians) V
  0x92680e8 RooRealVar::g2frac (fraction of gauss2) V
  0x9261760 RooArgusBG::argus (Argus PDF)  [Auto] V
    0x9249360 RooRealVar::x (x) V
    0x925fe80 RooRealVar::cutoff (argus cutoff) V
    0x925f900 RooRealVar::argpar (argus shape parameter) V
    0x9267288 RooConstVar::0.500000 (0.500000) V
```

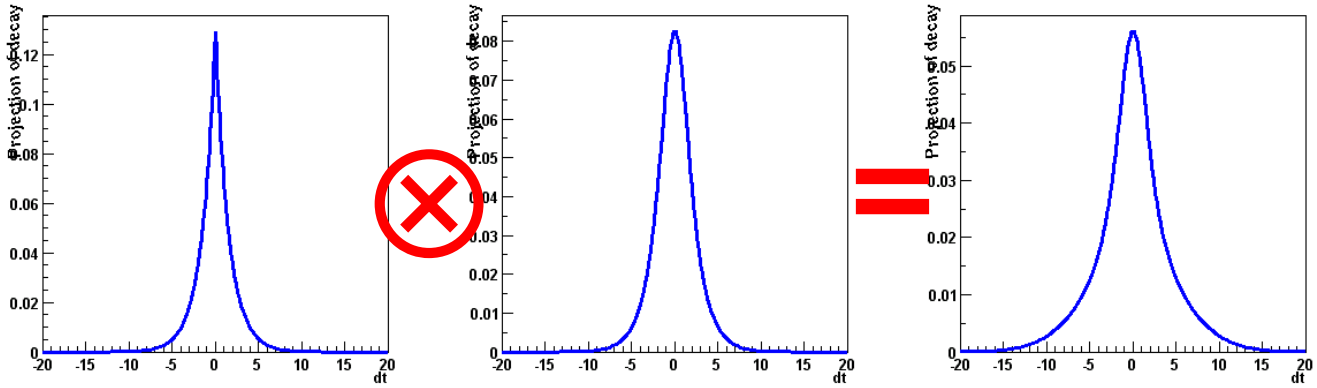
Putting it all together – Extended unbinned ML Fit to signal and background

```
// Declare observable x
RooRealVar x("x","x",0,10) ;

// Creation of 'sig', 'bkg' component p.d.f.s omitted for clarity

// Model = Nsig*sig + Nbkg*bkg (extended form)
RooRealVar nsig("nsig","#signal events",300,0.,2000.) ;
RooRealVar nbkg("nbkg","#background events",700,0,2000.) ;
RooAddPdf  model("model","sig+bkg",RooArgList(sig,bkg),RooArgList(nsig,nbkg)) ;

// Generate a data sample of Nexpected events
RooDataSet *data = model.generate(x) ;

// Fit model to data
model.fitTo(*data, Extended(kTRUE)) ;

// Plot data and PDF overlaid
RooPlot* xframe = x.frame() ;
data->plotOn(xframe) ;
model.plotOn(xframe) ;
model.plotOn(xframe,Components(bkg),
             LineStyle(kDashed)) ;
xframe->Draw() ;
```



Building models – Convolutions

- Many experimental observable quantities are well described by convolutions
  - Typically physics distribution smeared with experimental resolution (e.g. for B0 → J/ψ K_S exponential decay distribution smeared with Gaussian)



  - By explicitly describing observed distribution with a convolution p.d.f can disentangle detector and physics
    - To the extent that enough information is in the data to make this possible

Mathematical introduction & Numeric issues

- Mathematical form of convolution
  - Convolution of two functions

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

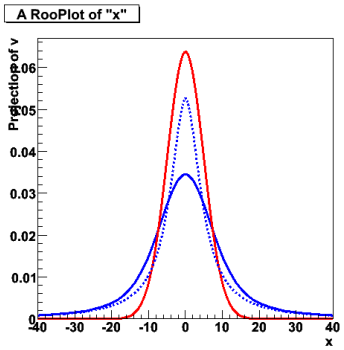
  - Convolution of two normalized p.d.f.s itself is *not* automatically normalized, so expression for convolution p.d.f is

$$F(x) \otimes G(x) = \frac{\int_{-\infty}^{+\infty} F(x) G(x - x') dx'}{\int_{x_{\min}}^{x_{\max}} \int_{-\infty}^{+\infty} F(x) G(x - x') dx' dx}$$

  - Because of (multiple) integrations required convolution are difficult to calculate
  - Convolution integrals are best done analytically, but often not possible

Convolution operation in RooFit

- RooFit has several options to construct convolution p.d.f.s
  - Class **RooNumConvPdf** – 'Brute force' numeric calculation of convolution (and normalization integrals)
  - Class **RooFFTConvPdf** – Calculate convolution integral using discrete FFT technology in fourier-transformed space.
  - Bases classes **RooAbsAnaConvPdf**, **RooResolutionModel**. Framework to construct analytical convolutions (with implementations mostly for B physics)
  - Class **RooVoigtian** – Analytical convolution of non-relativistic Breit-Wigner shape with a Gaussian



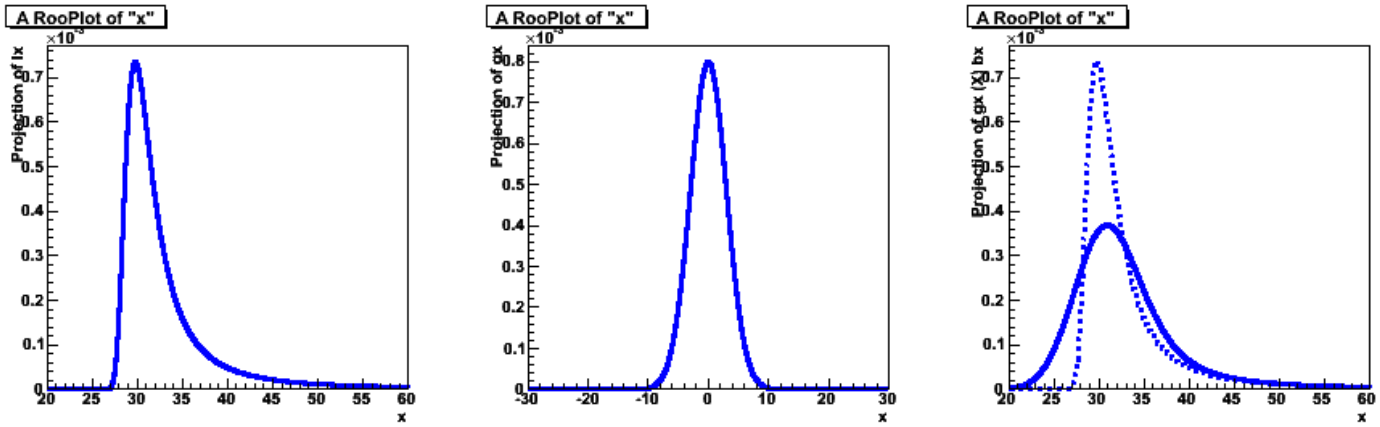
- All convolution in one dimension so far
  - N-dim extension of **RooFFTConvPdf** foreseen in future

Numeric convolutions – Class RooNumConvPdf

- Properties of **RooNumConvPdf**
  - Can convolve *any* two input p.d.f.s
  - Uses special numeric integrator that can compute integrals in [-∞,+∞] domain
  - Slow (very!) especially if requiring sufficient numeric precision to allow use in MINUIT (requires ~10^{-7} estimated precision). *Converge problems in MINUIT if precision is insufficient*

```
// Construct landau (x) gauss
RooNumConvPdf lxg("lxg","landau (X) gauss",t,landau,gauss) ;
```

Landau *Gauss* *Landau ⊗ Gauss*



Numeric convolutions – Class RooFFTConvPdf

- Properties of **`RooFFTConvPdf`**
  - Uses convolution theorem to compute *discrete* convolution in Fourier-Transformed space.
  - Transforms both input p.d.f.s with forward FFT

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}kn} \qquad k = 0, \ldots, N-1$$

(x_i are sampled values of p.d.f)

  - Makes use of Circular Convolution Theorem in Fourier Space

$$\begin{aligned}\mathcal{F}^{-1}\{\mathbf{X}\cdot\mathbf{Y}\}_n &= \sum_{l=0}^{N-1} x_l \sum_{m=-\infty}^{\infty} y_m \left(\sum_{p=-\infty}^{\infty} \delta_{m(n-l-pN)} \right) \\ &= \sum_{l=0}^{N-1} x_l \sum_{p=-\infty}^{\infty} \left(\sum_{m=-\infty}^{\infty} y_m \cdot \delta_{m(n-l-pN)} \right) \\ &= \sum_{l=0}^{N-1} x_l \left(\sum_{p=-\infty}^{\infty} y_{n-l-pN} \right) \stackrel{\text{def}}{=} (\mathbf{x} * \mathbf{y_N})_n \ ,\end{aligned}$$

  - *Convolution can be computed in terms of products of Fourier components (easy)*
  - Apply inverse Fourier transform to obtained convoluted p.d.f in space domain

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N}kn} \qquad n = 0, \ldots, N-1.$$

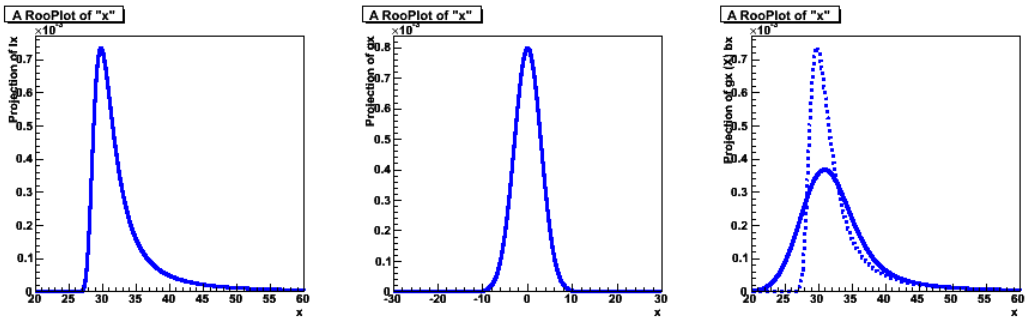
Numeric convolutions – Class RooFFTConvPdf

- Fourier transforms calculated by FFTW3 package
  - Interfaced in ROOT through `TVirtualFFT` class
- About 100x faster than `RooNumConvPdf`
  - Also much better numeric stability (c.f. MINUIT converge)
  - Choose sufficiently large number of samplings to obtain smooth output p.d.f
  - CPU time is **not** proportional to number of samples, e.g. 10000 bins works fine in practice
- Note: p.d.f.s are not sampled from $[-\infty,+\infty]$, but from $[x_{min},x_{max}]$
- Note: p.d.f is explicitly treated as *cyclical* beyond range
  - Excellent for cyclical observables such as angles
  - If p.d.f converges to zero towards both ends of range if non-cyclical observable, all works out fine
  - If p.d.f does not converge to zero towards domain end, cyclical leakage will occur

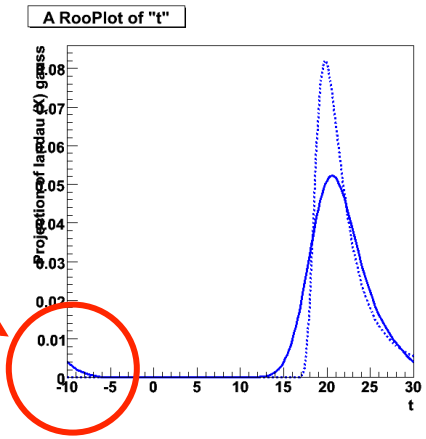
Numeric convolutions – Class RooFFTConvPdf

- Usage example

```
// Construct landau (x) gauss (10000 samplings 2nd order interpolation)
t.setBins(10000,"cache") ;
RooFFTConvPdf lxg("lxg","landau (X) gauss",t,landau,gauss,2) ;
```



- Example with cyclical 'leakage'
  - Can reduce this by specifying a 'buffer zone' in FFT calculation beyond end of ranges
    `conv.setBufferFraction(0.3)`

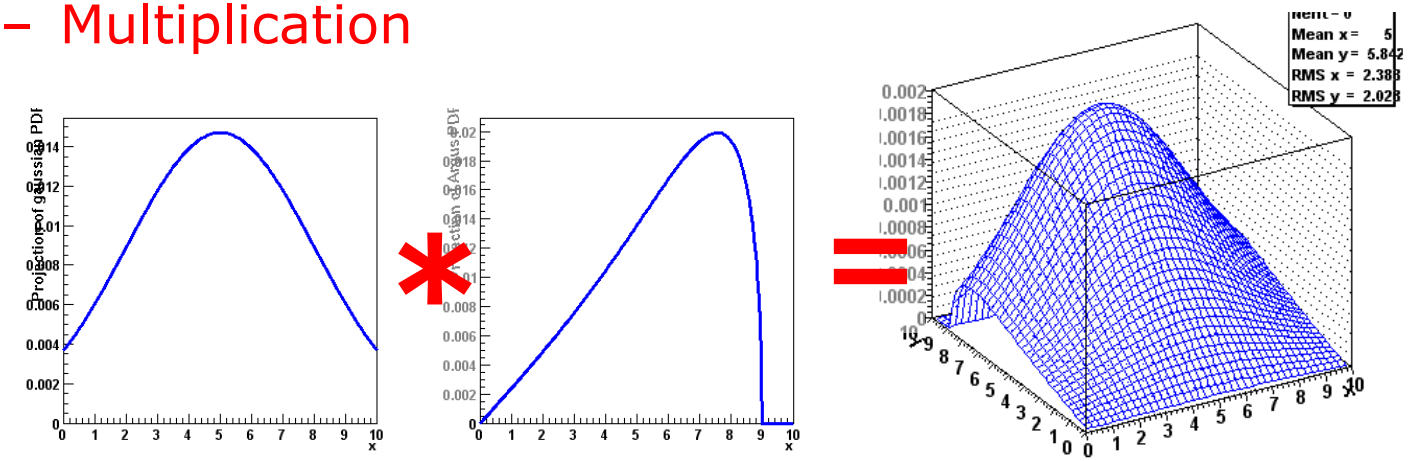


4 Multidimensional models

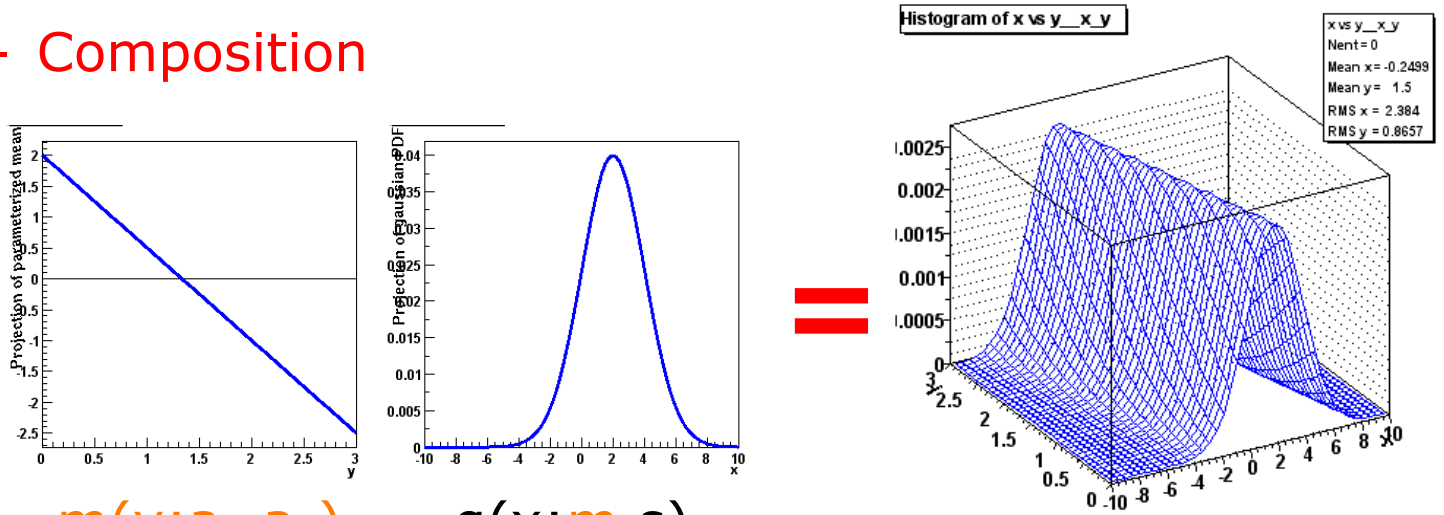
- *Uncorrelated products of p.d.f.s*
- *Using composition to p.d.f.s with correlation*
- *Products of conditional and plain p.d.f.s*

Building realistic models

- Multiplication



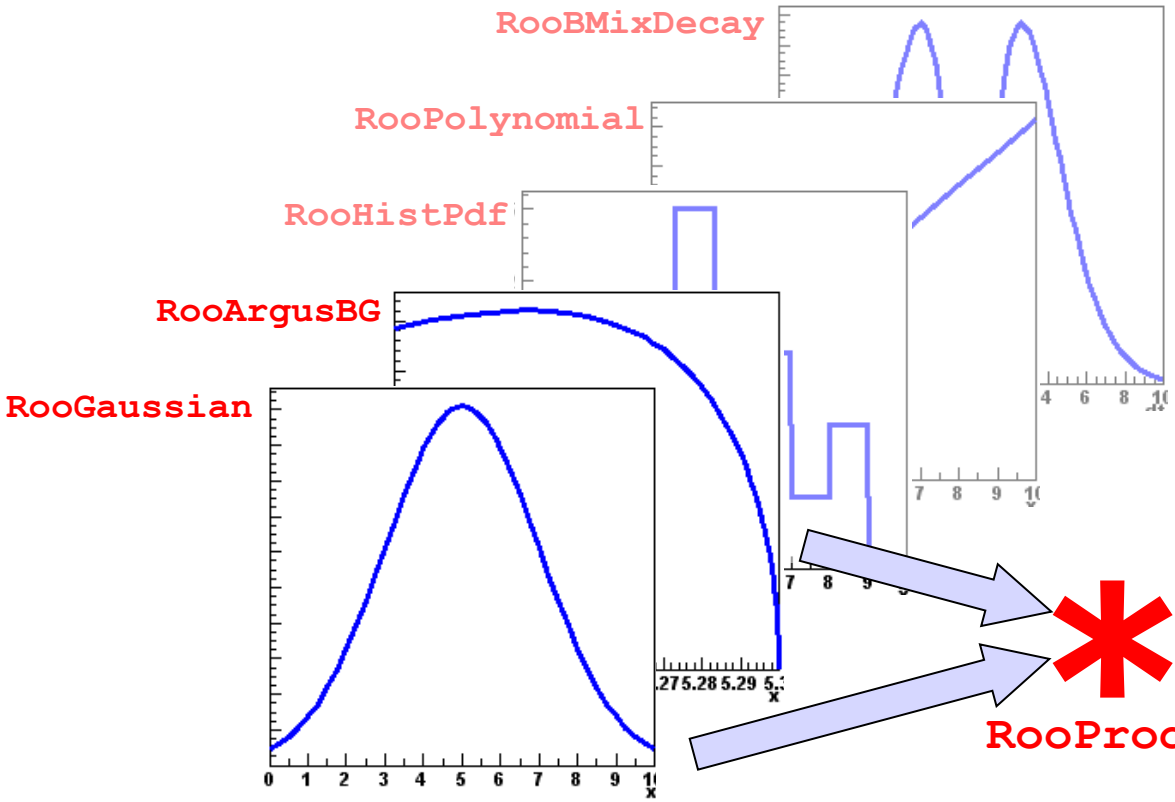
- Composition



$m(y;a_0,a_1)$ $g(x;m,s)$ $g(x,y;a0,a1,s)$

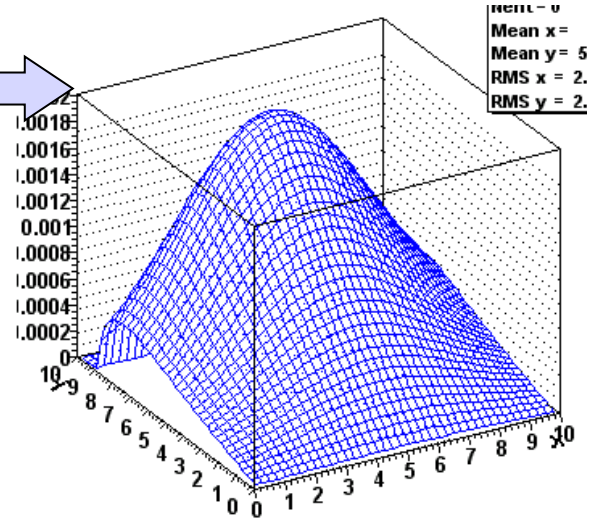
Possible in *any* PDF
No explicit support in PDF code needed

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



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

RooProdPdf



Uncorrelated products – Mathematics and constructors

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

2D

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

nD

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

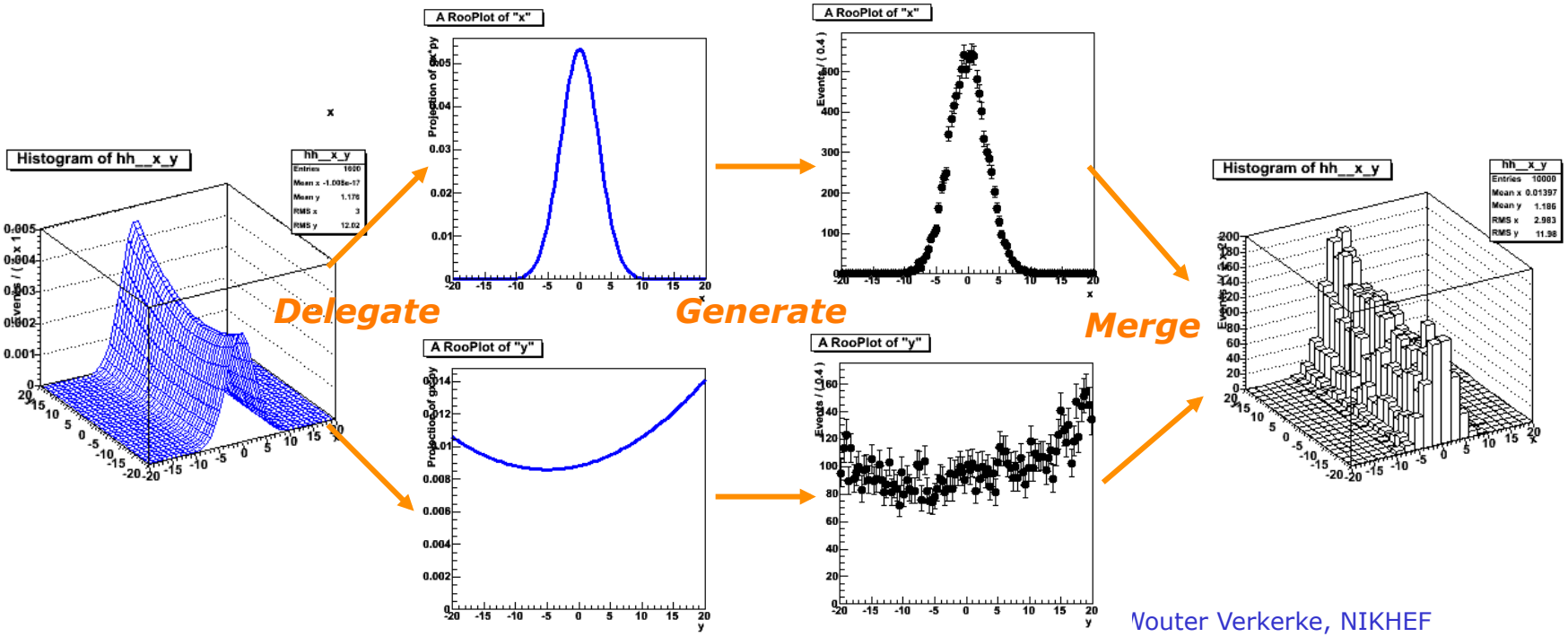
  - No explicit normalization required → If input p.d.f.s are unit normalized, product is also unit normalized
    (this is true *only* because of the absence of correlations)

- Corresponding RooFit operator p.d.f. is **`RooProdPdf`**
  - Returns product of *normalized* input p.d.f values

```
RooGaussian gx("gx","gaussian PDF",x,meanx,sigmax) ;
RooGaussian gy("gy","gaussian PDF",y,meany,sigmay) ;

// Multiply gaussx and gaussy into a two-dimensional p.d.f. gaussxy
RooProdPdf  gaussxy("gxy","gx*gy",RooArgList(gx,gy)) ;
```

How it work – event generation on uncorrelated products

- If p.d.f.s are uncorrelated, each observable can be generated separately
  - Reduced dimensionality of problem (important for e.g. accept/ reject sampling)
  - Actual event generation delegated to component p.d.f (can e.g. use internal generator if available)
  - **`RooProdPdf`** just aggregates output in single dataset



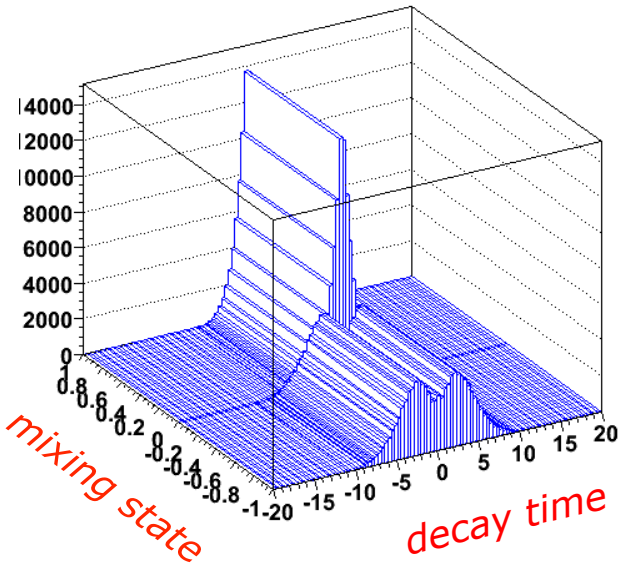
Fundamental multi-dimensional p.d.fs

- It also possible define multi-dimensional p.d.f.s that do not arise through a product construction
  - For example

```
RooGenericPdf gp("gp","sqrt(x+y)*sqrt(x-y)",RooArSet(x,y)) ;
```

  - But *usually n-dim p.d.f.s are constructed more intuitively* through product constructs. Also correlations can be introduced efficiently (more on that in a moment)

- Example of fundamental 2-D B-physics p.d.f. **RooBMixDecay**
  - Two observables:
    decay time (t, continuous)
    mixingState (m, discrete [-1,+1])

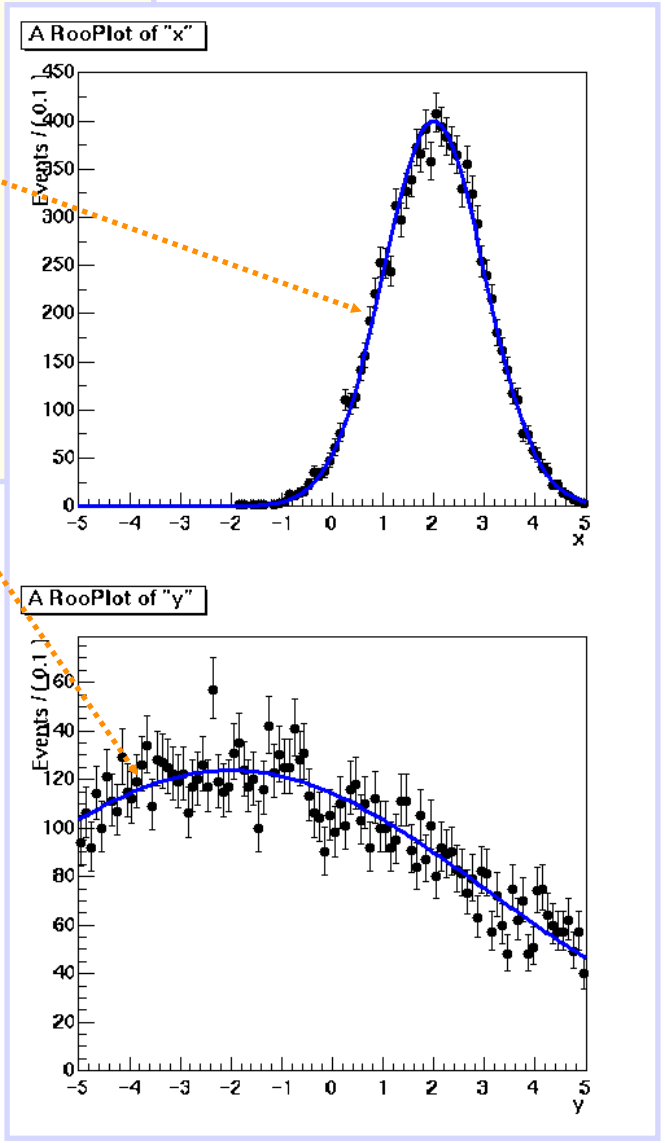


Plotting multi-dimensional PDFs

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

c->cd(2) ;
RooPlot* yframe = y.frame() ;
data->plotOn(yframe) ;
prod->plotOn(yframe) ;
yframe->Draw() ;
```

$$f(x) = \int pdf(x,y)dy$$

$$f(y) = \int pdf(x,y)dx$$



-Plotting a dataset D(x,y) versus x represents a *projection over y*

-To overlay PDF(x,y), you must plot *Int(dy)PDF(x,y)*

-RooFit automatically takes care of this!

- RooPlot remembers dimensions of plotted datasets

Projecting out hidden dimensions

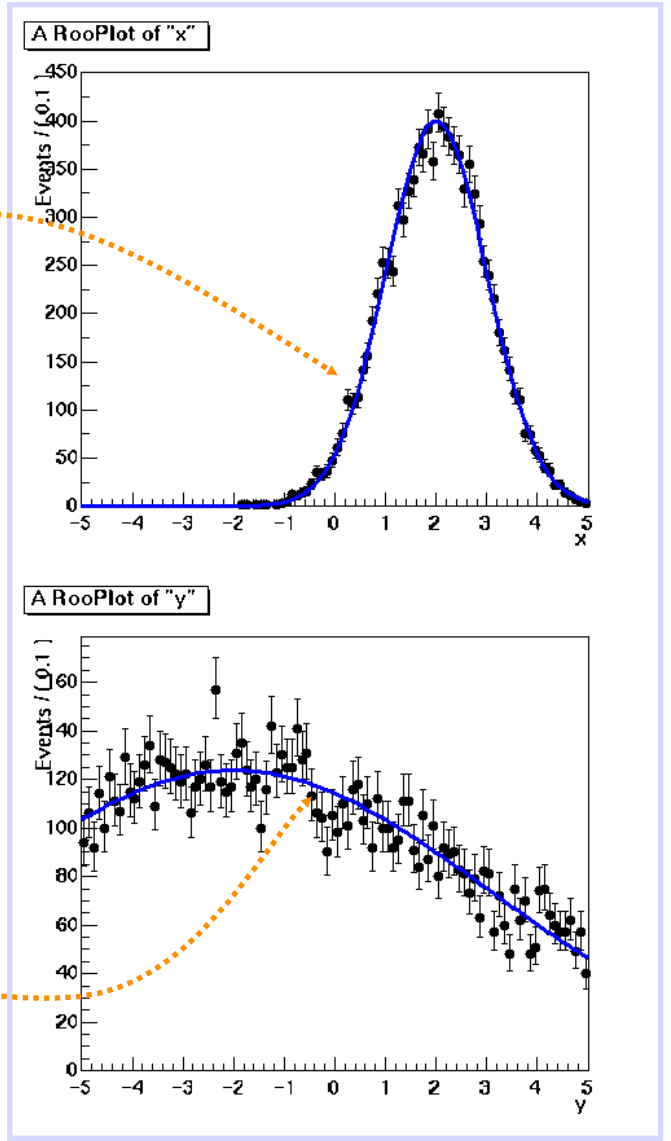
- Example in 2 dimensions
  - 2-dim dataset D(x,y)
  - 2-dim PDF P(x,y)=gauss(x)*gauss(y)

- 1-dim plot versus x

$$P_p(x) = \frac{\int p(x,y)dy}{\int p(x,y)dxdy}$$

- 1-dim plot versus y

$$P_p(y) = \frac{\int p(x,y)dx}{\int p(x,y)dxdy}$$



RooProdPdf automatic optimization for uncorrelated terms

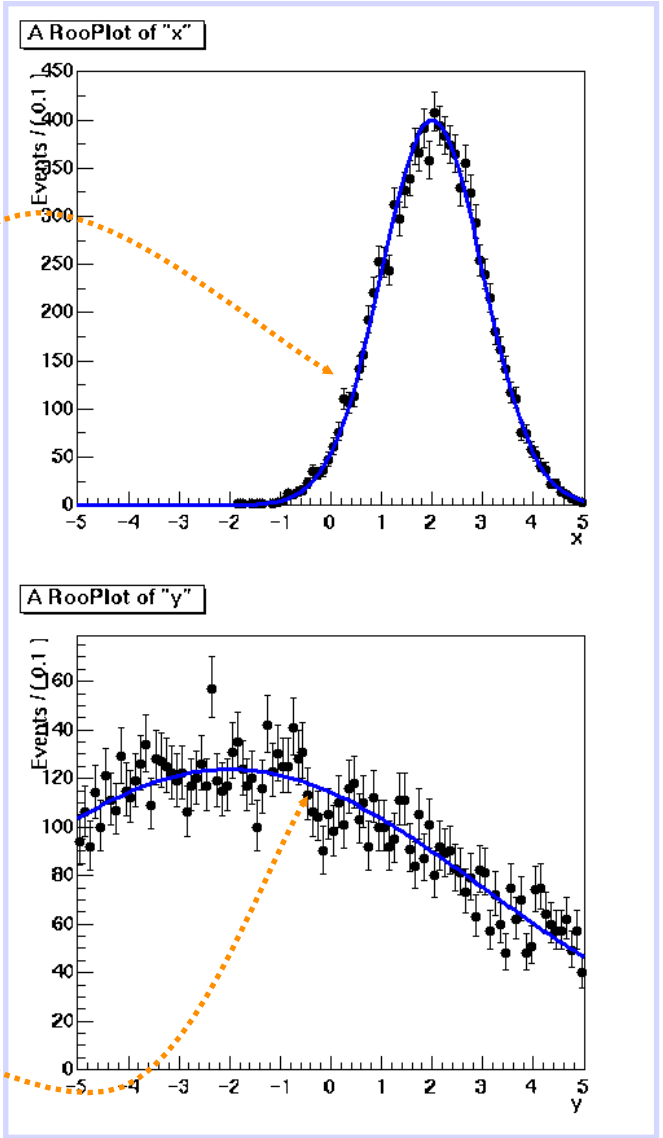
- Example in 2 dimensions
  - 2-dim dataset D(x,y)
  - 2-dim PDF P(x,y)=gaus(x)*gauss(y)

- 1-dim plot versus x

$$P_p(x) = \frac{\int g(x)g(y)dy}{\int g(x)g(y)dxdy} = \frac{g(x)\int g(y)dy}{\int g(x)dx\int g(y)dy} = \frac{g(x)}{\int g(x)dx}$$

- 1-dim plot versus y

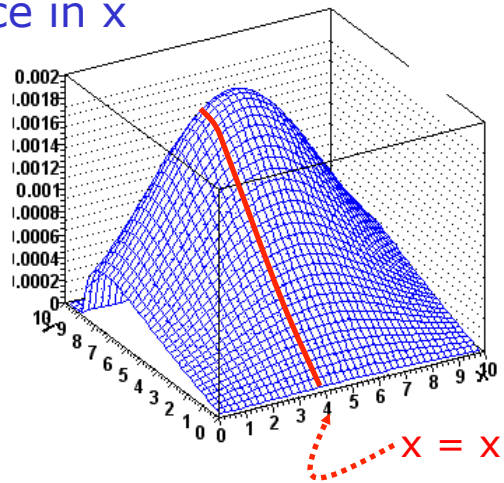
$$P_p(y) = \frac{\int g(x)g(y)dx}{\int g(x)g(y)dxdy} = \frac{\int g(x)dx \cdot g(y)}{\int g(x)dx\int g(y)dy} = \frac{g(y)}{\int g(y)dy}$$



Introduction to slicing

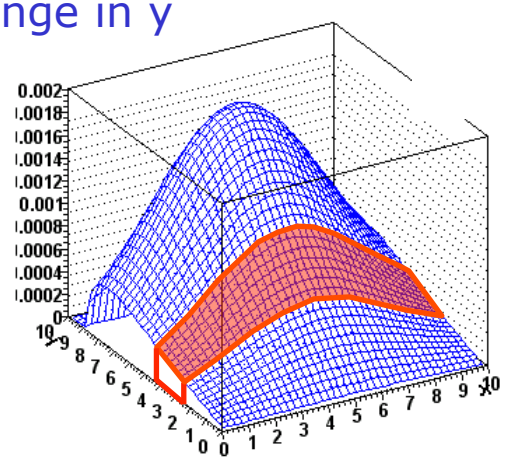
- With multidimensional p.d.f.s it is also often useful to be able to plot a slice of a p.d.f
- In RooFit
  - A *slice* is thin
  - A *range* is thick
- Slices mostly useful in discrete observables
  - A slice in a continuous observable has no width and usually no data with the corresponding cut (e.g. “x=5.234”)
- Ranges work for both continuous and discrete observables
  - Range of discrete observable can be list of >=1 state

Slice in x



x = x.getVal()

Range in y

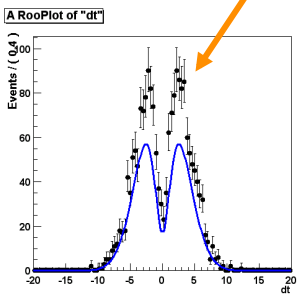


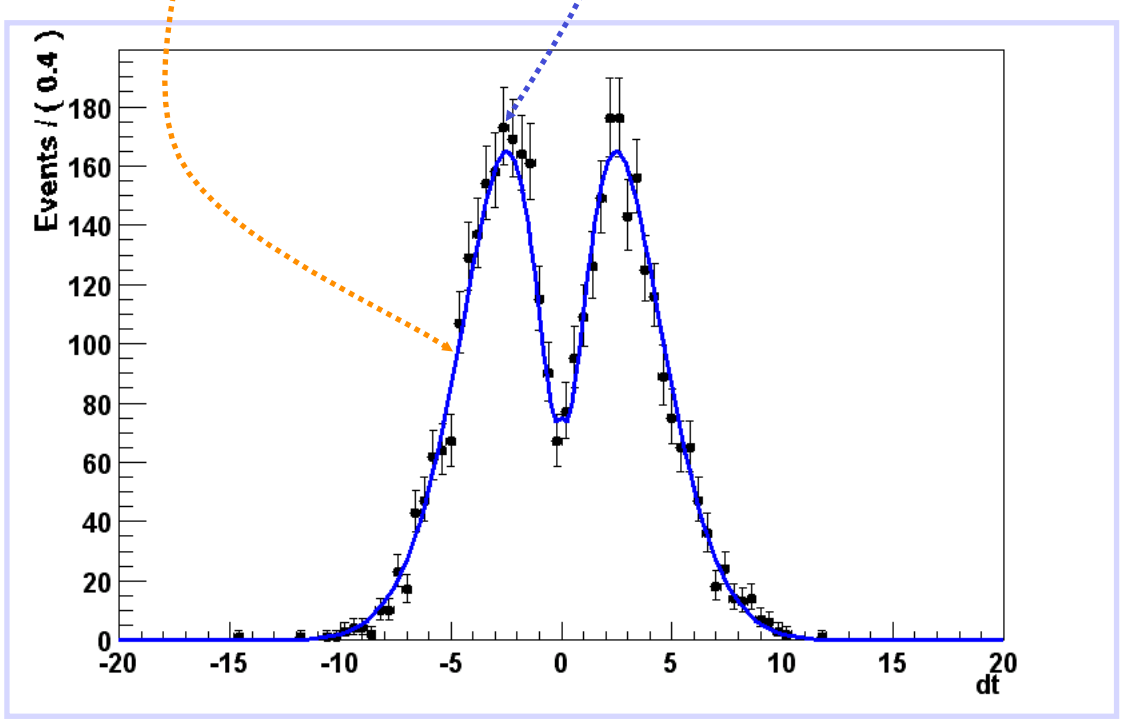
Plotting a *slice* of a p.d.f

```
RooPlot* dtframe = dt.frame() ;
data->plotOn(dtframe,Cut("mixState==mixState::mixed")) ;

mixState = "mixed" ;
bmix.plotOn(dtframe,Slice(mixState)) ;
dtframe->Draw() ;
```

Slice is positioned at 'current' value of sliced observable

For slices both data and p.d.f normalize with respect to full dataset. If fraction 'mixed' in above example disagrees between data and p.d.f prediction, this discrepancy will show in plot



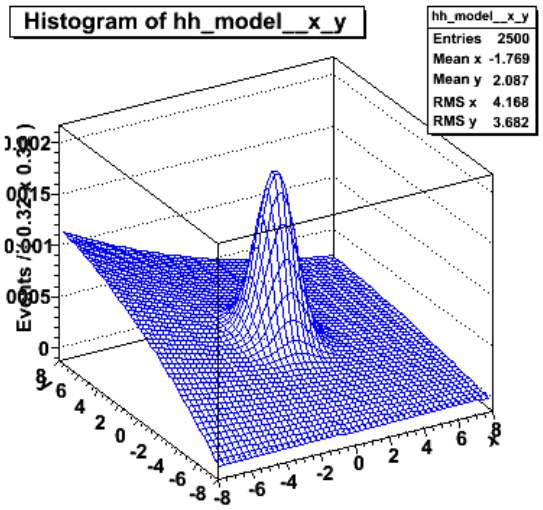


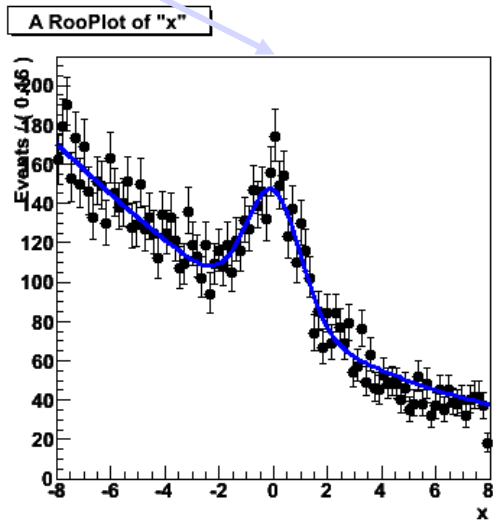
Plotting a *range* of a p.d.f and a dataset

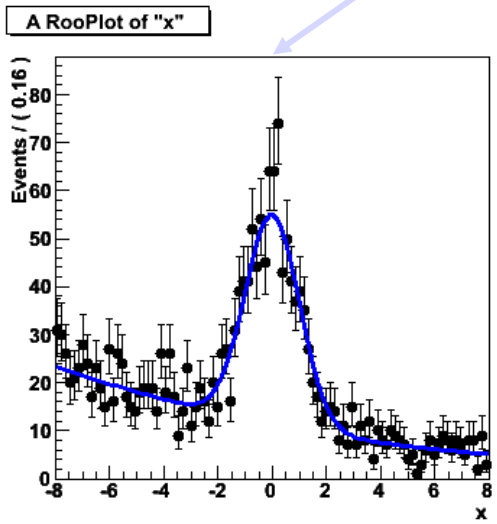
model(x,y) = gauss(x)*gauss(y) + poly(x)*poly(y)

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

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







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

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

Plotting non-rectangular PDF regions

- Why is this interesting? Because with this technique we can trivially implement projection over arbitrarily shaped regions.
  - Any cut prescription that you can think of to apply to data works

- Example: Likelihood ratio projection plot
  - Common technique in rare decay analyses
  - PDF typically consist of N-dimensional event selection PDF, where N is large (e.g. 6.)
  - Projection of data & PDF in any of the N dimensions doesn't show a significant excess of signal events
  - To demonstrate purity of selected signal, plot data distribution (with overlaid PDF) in one dimension, while selecting events with a cut on the likelihood ratio of signal and background in the remaining N-1 dimensions

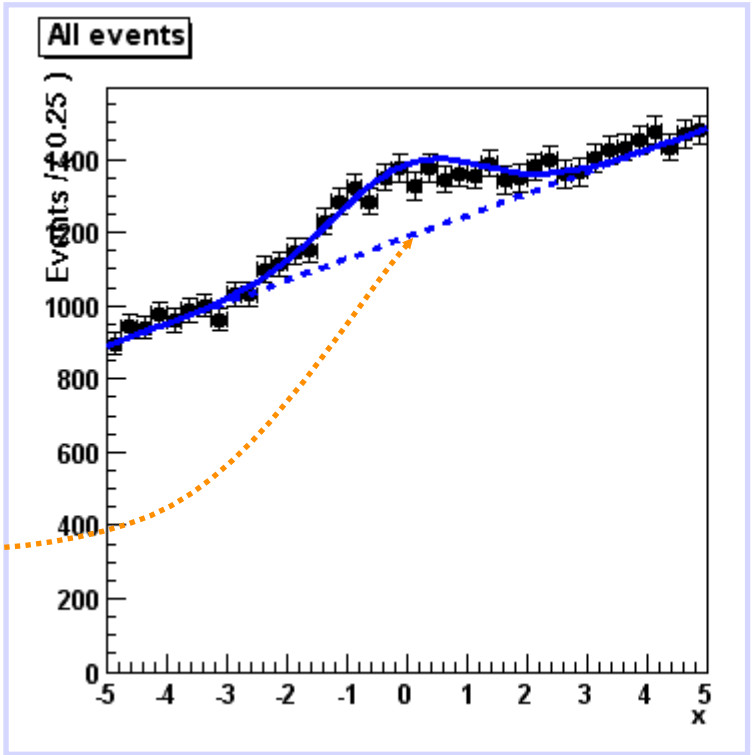
Plotting data & PDF with a likelihood ratio cut

- Simple example
  - 3 observables (x,y,z)
  - Signal shape: gauss(x)·gauss(y)·gauss(z)
  - Background shape: (1+a·x)(1+b·y)(1+c·z)
  - Plot distribution in x

```
// Plot x distribution of all events
RooPlot* xframe1 = x.frame(40) ;
data->plotOn(xframe1) ;
sum.plotOn(xframe1) ;
```

Integrated projection of data/PDF on X doesn't reflect signal/background discrimination power of PDF in y,z

Use LR ratio technique to only plot events with are signal-like according to p.d.f in projected observable (y,z)



Plotting data & PDF with a likelihood ratio cut

- Given a p.d.f. with three observable (x,y,z)
  how do you calculate the S'(y,z)/(S'(y,z)+B'(y,z)) L ratio
- First calculate projected likelihoods S' and B'
  - Use the built-in createProjection method which returns a projection of a given p.d.f.s

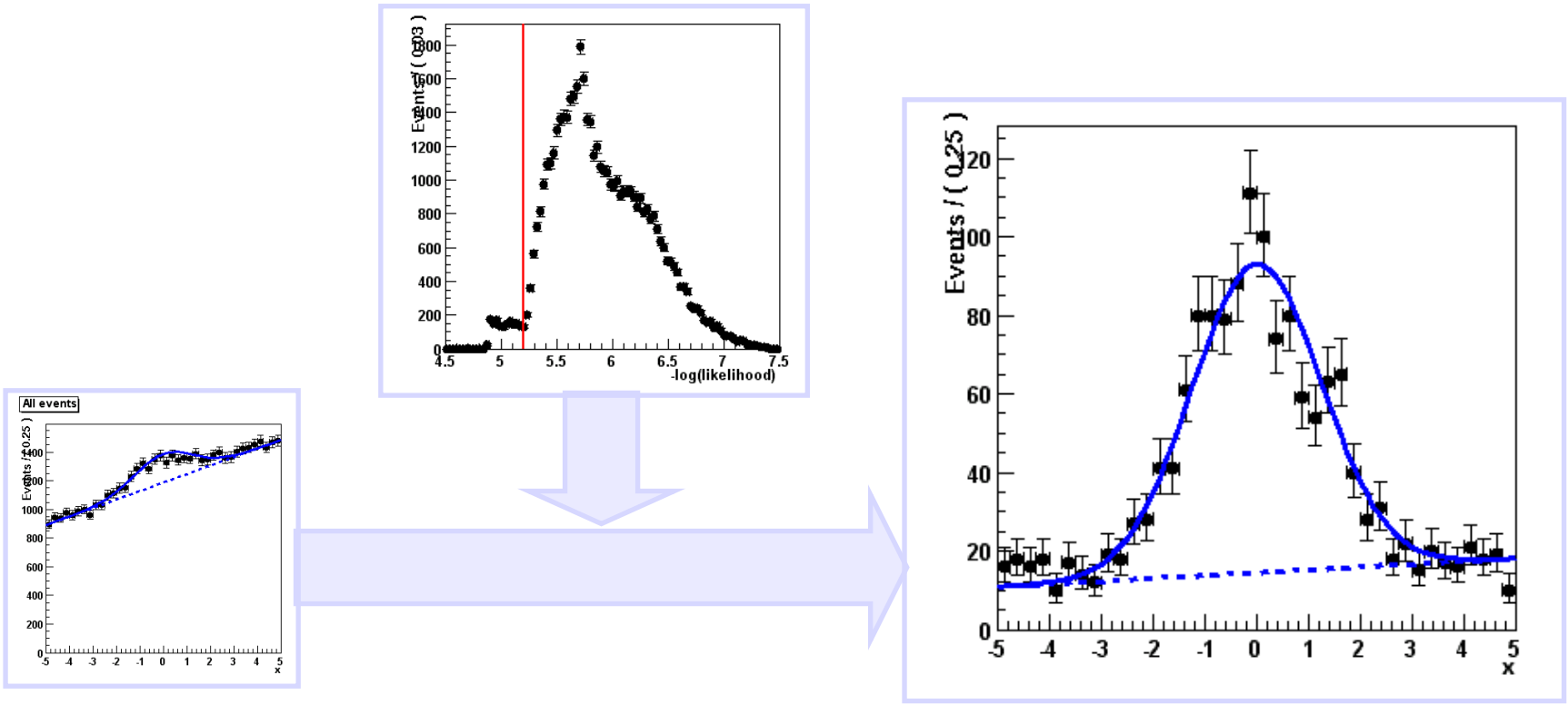
```
RooAbsPdf* sigYZ = sig->createProjection(x) ;
RooAbsPdf* totYZ = model->createProjection(x) ;
```

- The calculate ratio for each event

```
// Formula expression of LR
RooFormulaVar LR("LR","-log(sigYZ)-(-log(totYZ)",
                 RooArgSet(*sigYX,*totYZ)) ;

// Add column to dataset with precalculate value of LR
data->addColumn(LR) ;
```

Plotting data & PDF with a likelihood cut

- Look at distribution of per-event LR in toy MC sample and decide on suitable cut

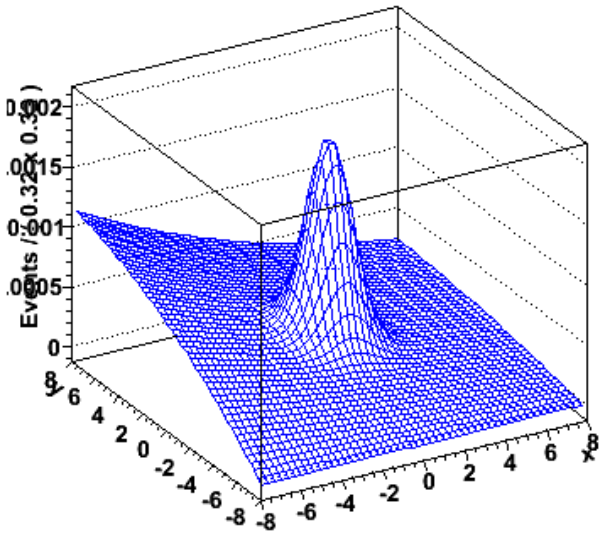


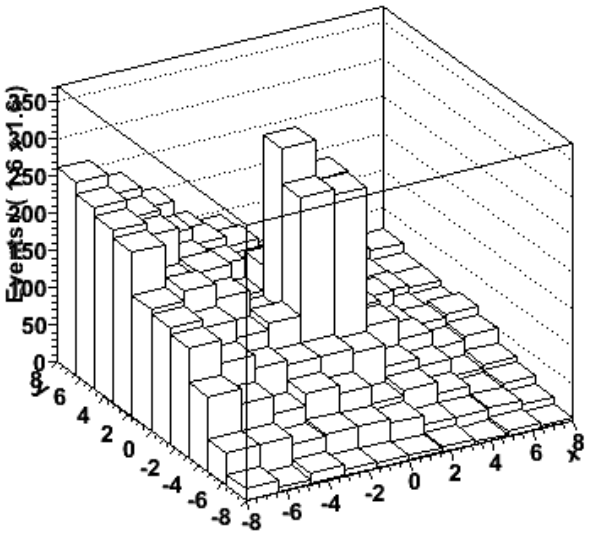
- Apply cut to both data sample and toyMC sample for projection and make plot

Plotting in more than 2,3 dimensions

- No equivalent of RooPlot for >1 dimensions
  - Usually >1D plots are not overlaid anyway
- Easy to use **`createHistogram()`** methods provided in both **`RooAbsData`** and **`RooAbsPdf`** to fill ROOT 2D,3D histograms

```
TH2D* ph2 = pdf.createHistogram("ph2",x,YVar(y)) ;

TH2* dh2 = data.createHistogram("dg2",x,Binning(10),
                                YVar(y,Binning(10)));
ph2->Draw("SURF") ;
dh2->Draw("LEGO") ;
```





Building models – Introducing correlations

- Easiest way to do this is
  - start with 1-dim p.d.f. and change on of its parameters into a function that depends on another observable

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

  - Natural way to think about it
- Example problem
  - Observable is reconstructed mass M of some object.
  - Fitting Gaussian g(M,mean,sigma) some background to dataset D(M)
  - But reconstructed mass has *bias* depending on some other observable X
  - Rewrite fit functions as g(M,meanCorr(mtrue,X,alpha),sigma) where meanCorr is an (emperical) function that corrects for the bias depending on X

Coding the example problem

How do you code the preceding example problem

$$PDF(x,y) = gauss(x,m(y),s)$$

$$m(y) = m_0 + m_1 \cdot sqrt(y)$$

How do you do that? Just like that:

```
RooRealVar x("x","x",-10,10) ;
RooRealVar y("y","y",0,3) ;

// Build a parameterized mean variable for gauss
RooRealVar mean0("mean0",“mean offset",0.5) ;
RooRealVar mean1("mean1",“mean slope",3.0) ;
RooFormulaVar mean("mean","mean0+mean1*y",
                   RooArgList(mean0,mean1,y)) ;

RooRealVar sigma("sigma","width of gaussian",3) ;
RooGaussian gauss("gauss","gaussian",x,mean,sigma);
```

Build a function object $m(y)=m_0+m_1*sqrt(y)$

Simply plug in function mean(y) where mean value is expected!

PDF expects a real-valued object as input, not necessarily a variable

Plug-and-play parameters!

Generic real-valued functions

- **RooFormulaVar** makes use of the ROOT **TFormula** technology to build interpreted functions
  - Understands generic C++ expressions, operators etc
  - Two ways to reference RooFit objects
    By name:

```
RooFormulaVar f("f","exp(foo)*sqrt(bar)", RooArgList(foo,bar)) ;
```

By position:

```
RooFormulaVar f("f","exp(@0)*sqrt(@1)",RooArgList(foo,bar)) ;
```

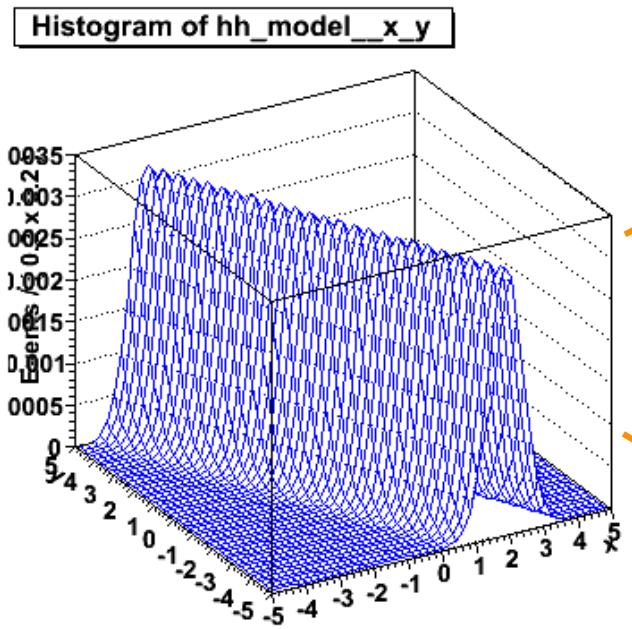
  - You can use **RooFormulaVar** where ever a 'real' variable is requested

- **RooPolyVar** is a compiled polynomial function

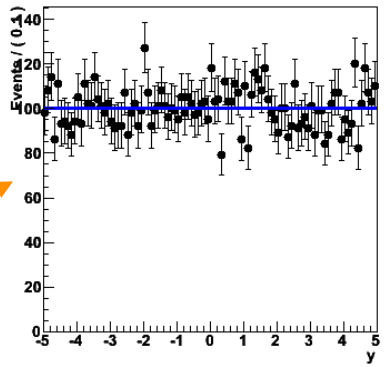
```
RooRealVar x("x","x",0.,1.) ;
RooRealVar p0("p0","p0",5.0) ;
RooRealVar p1("p1","p1",-2.0) ;
RooRealVar p2("p2","p2",3.0) ;
RooFormulaVar f("f","polynomial",x,RooArgList(p0,p1,p2)) ;
```

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

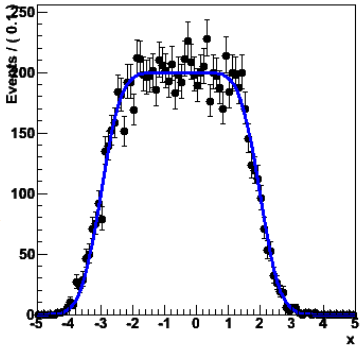
- Make 2D plot of p.d.f in (x,y)



Projection on Y



Projection on X



- Is the correct p.d.f for this problem?
  - Constructed a p.d.f with correct shape in x, given a value of y → OK
  - But p.d.f predicts flat distribution in y → Probably not OK
  - What we want is a pdf for X *given* Y, but without prediction on Y → Definition of a *conditional* p.d.f F(x|y)

Conditional p.d.f.s – Formulation and construction

- Mathematical formulation of a conditional p.d.f
  - A conditional p.d.f is not normalized w.r.t its conditional observables

$$F(\vec{x} \mid \vec{y}; \vec{p}) = \frac{f(\vec{x}, \vec{y}, \vec{p})}{\int f(\vec{x}, \vec{y}, \vec{p}) d\vec{x}}$$

  - Note that denominator in above expression *depends on y* and is thus in general different for each event

- Constructing a conditional p.d.f in RooFit
  - Any RooFit p.d.f can be used as a conditional p.d.f as objects have no internal notion of distinction between parameters, observables and conditional observables
  - Observables that should be used as conditional observables have to be specified in use context (generation, plotting, fitting etc…)

Using a conditional p.d.f – fitting and plotting

- For fitting, indicate in fitTo() call what the conditional observables are

```
pdf.fitTo(data,ConditionalObservables(y))
```

$$F(x \mid y) = \frac{f(x,y)}{\int f(x,y)d\vec{x}}$$

  - You may notice a performance penalty if the normalization integral of the p.d.f needs to be calculated numerically.
    For a conditional p.d.f it must evaluated again for each event

- Plotting: You cannot project a conditional $F(x|y)$ on x without external information on the distribution of y
  - Substitute integration with averaging over y values in data

Integrate over y

$$P_p(x) = \frac{\int p(x,y)dy}{\int p(x,y)dxdy}$$

Sum over all y_i in dataset D

$$P_p(x) = \frac{1}{N}\sum_{D}^{i=1,N} \frac{p(x,y_i)}{\int p(x,y_i)dx}$$

Physics example with conditional p.d.f.s

- Want to fit decay time distribution of B0 mesons (exponential) convoluted with Gaussian resolution

$$F(t) = D(t;\tau) \otimes R(t,m,\sigma)$$

- However, resolution on decay time varies from event by event (e.g. more or less tracks available).
  - We have in the data an error estimate dt for each measurement from the decay vertex fitter ("*per-event error*")
  - Incorporate this information into this physics model

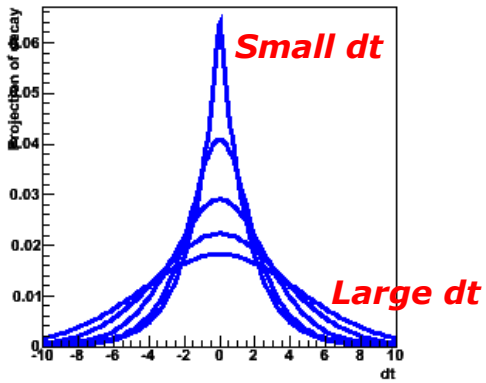
$$F(t \mid \delta t) = D(t;\tau) \otimes R(t,m,\sigma \cdot \delta t)$$

  - Resolution in physics model is adjusted for each event to expected error.
  - Overall scale factor σ can account for incorrect vertex error estimates (i.e. if fitted σ>1 then dt was underestimate of true error)
  - Physics p.d.f must used conditional conditional p.d.f because it give no sensible prediction on the distribution of the per-event errors

Physics example with conditional p.d.f.s

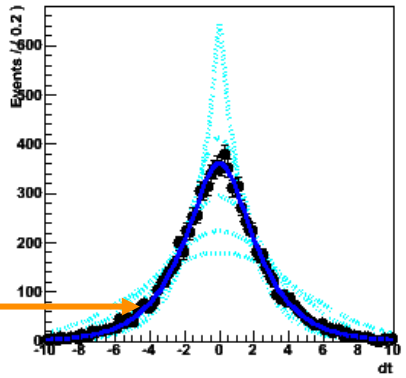
- Some illustrations of decay model with per-event errors
  - Shape of $F(t|\delta t)$ for several values of δt

$$F(t \mid \delta t) = D(t;\tau) \otimes R(t, m, \sigma \cdot \delta t)$$



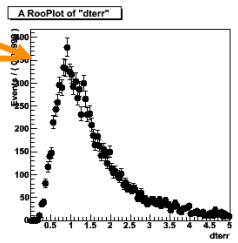
- Plot of *D(t)* and *F(t|dt)* projected over *dt*

```
// Plotting of decay(t|dterr)
 RooPlot* frame = dt.frame() ;
 data->plotOn(frame2) ;
 decay_gm1.plotOn(frame2,ProjWData(*data)) ;
```

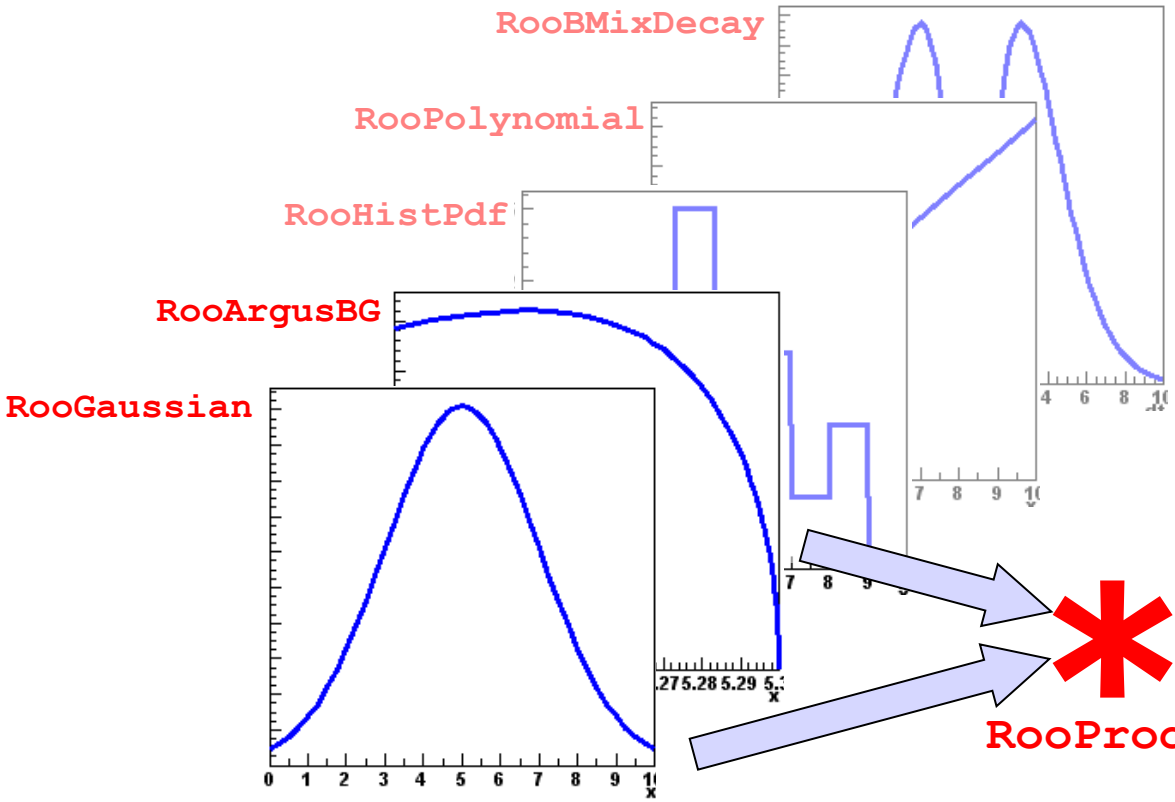


Note that projecting over large datasets can be slow. You can speed this up by projecting with a binned copy of the projection data

$$P_p(x) = \frac{1}{N} \sum_{D}^{i=1,N} \frac{p(x, y_i)}{\int p(x, y_i) dx}$$



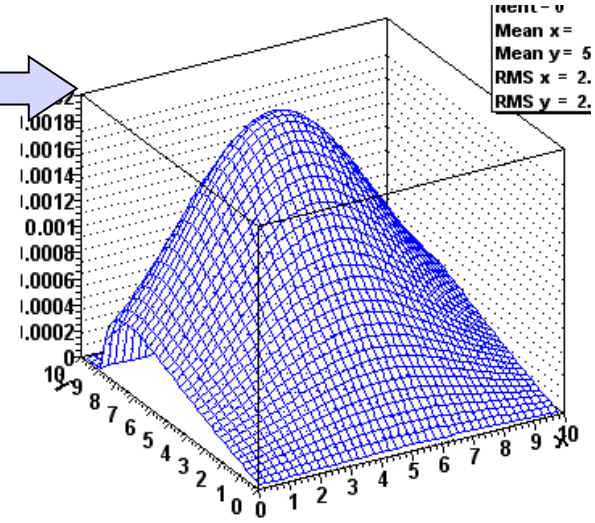
Model building – Products with conditional p.d.f.s



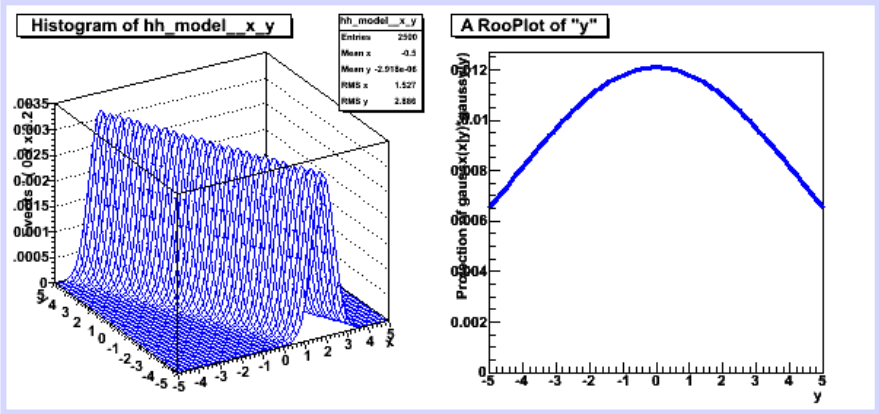

```
RooProdPdf k("k","k",g,
            Conditional(f,x))
```

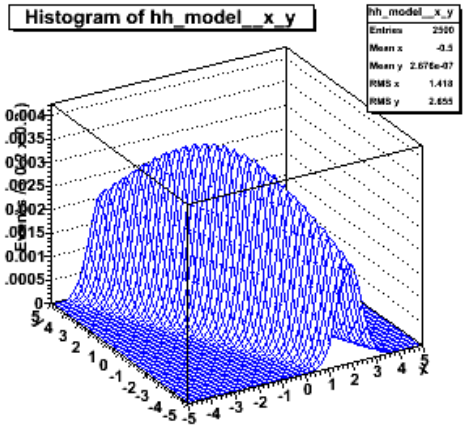
$$K(x,y) = F(x \mid y) \cdot G(y)$$

RooProdPdf



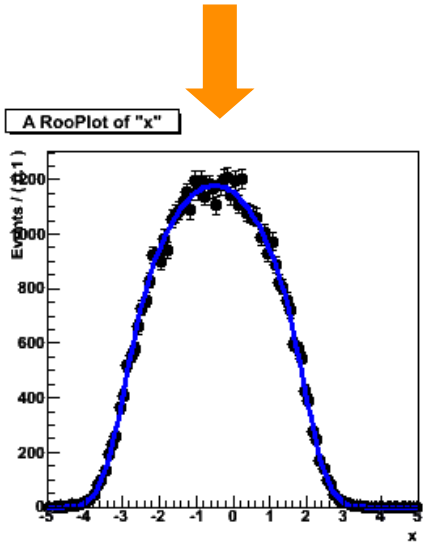
Example with product of conditional and plain p.d.f.





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

```
// Create function f(y) = a0 + a1*y
RooPolyVar fy("fy","fy",y,RooArgSet(a0,a1)) ;

// Create gaussx(x,f(y),0.5)
RooGaussian gaussx("gaussx","gaussx",x,fy,sx) ;

// Create gaussy(y,0,3)
RooGaussian gaussy("gaussy","Gaussian in y",y,my,sy) ;

// Create gaussx(x,sx|y) * gaussy(y)
RooProdPdf model("model","gaussx(x|y)*gaussy(y)",
                 gaussy,Conditional(gaussx,x)) ;
```



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

5 Managing data, discrete variables simultaneous fits

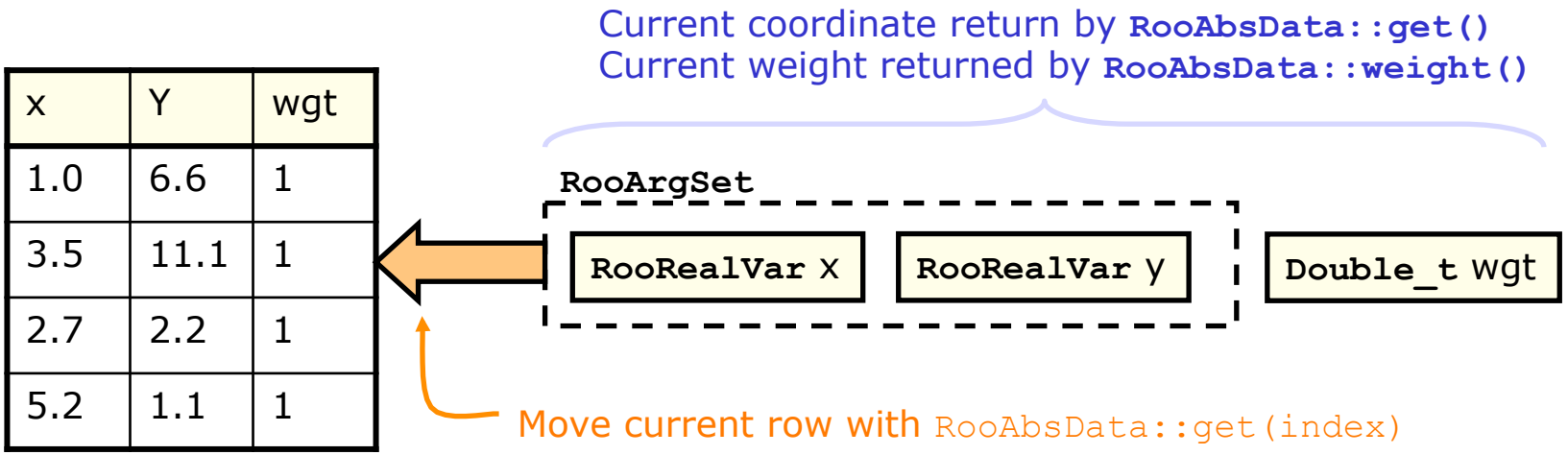
- *Binned, unbinned datasets*
- *Importing data*
- *Using discrete variable to classify data*
- *Simultaneous fits on multiple datasets*

Wouter Verkerke, NIKHEF

A bit more detail on RooFit datasets

- A dataset is a N-dimensional collection of points
  - With optional weights
  - No limit on number of dimensions
  - Observables continuous (**`RooRealVar`**) or discrete (**`RooCategory`**)

- Interface of each dataset is 'current' row
  - Set of RooFit value objects that represent coordinate of current event



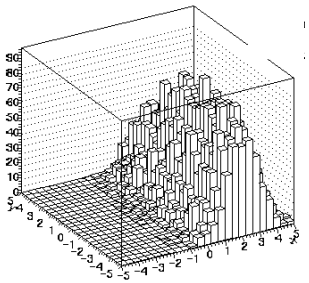
Binned data, or unbinned data (with optional weights)

- Binned or unbinned ML fit?
  - In most RooFit applications it doesn't matter

Unbinned

x	y	z
1	3	5
2	4	6
1	3	5
2	4	6

Binned



Internally binned data is represented the same way as unbinned data,

A ROOT TTree with the bin coordinates

RooDataSet **RooDataHist**

RooAbsData

- For example ML fitting interface takes abstract RooAbsData object
  - Binned data → Binned likelihood
  - Unbinned data → Unbinned likelihood
- Weights are supported in unbinned datasets
  - But use with care. Error analysis in ML fits to weighted unbinned data can be complicated!

Importing unbinned data

- From ROOT trees
  - `RooRealVar` variables are imported from /D /F /I tree branches
  - `RooCategory` variables are imported from /I /b tree branches
  - Mapping between `TTree` branches and dataset variables by name: e.g. `RooRealVar x("x","x",-10,10)` imports `TTree` branch "x"

```
RooRealVar x("x","x",-10,10) ;
RooRealVar c("c","c",0,30) ;
RooDataSet data("data","data",inputTree,RooArgSet(x,c));
```

  - Only events with 'valid' entries are imported. In above example any events with |x|>10 or c<0 or c>30 are *not* imported

- From ASCII files
  - One line per event, order of variables as given in RooArgList

```
RooDataSet* data =
   RooDataSet::read("ascii.file",RooArgList(x,c)) ;
```

Importing binned data

- From ROOT **`THx`** histogram objects

```
RooDataHist bdata1("bdata","bdata",RooArgList(x),histo1d);
RooDataHist bdata2("bdata","bdata",RooArgList(x,y),histo2d);
RooDataHist bdata3("bdata","bdata",RooArgList(x,y,z),histo3d);
```

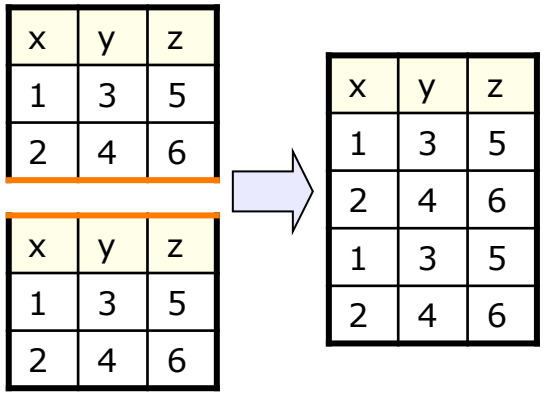
- From a **`RooDataSet`**

```
RooDataHist* binnedData = data->binnedClone() ;
```

Extending and reducing **unbinned** datasets

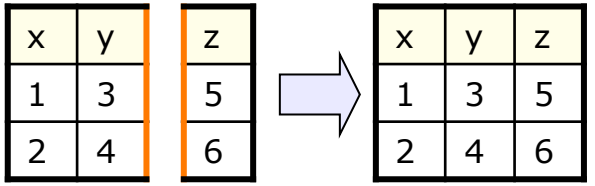
- Appending

```
RooDataSet d1("d1","d1",RooArgSet(x,y,z));
RooDataSet d2("d2","d2",RooArgSet(x,y,z));

d1.append(d2) ;
```



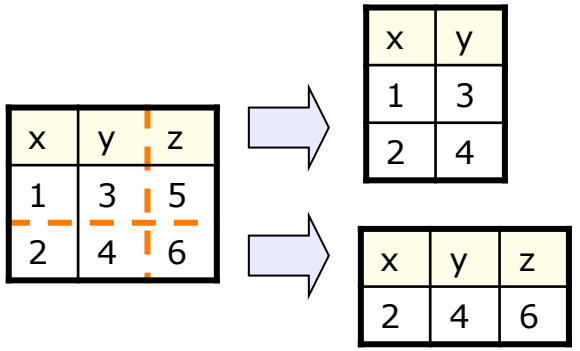
- Merging

```
RooDataSet d1("d1","d1",RooArgSet(x,y) ;
RooDataSet d2("d2","d2",RooArgSet(z)) ;

d1.merge(d2) ;
```



- Reducing

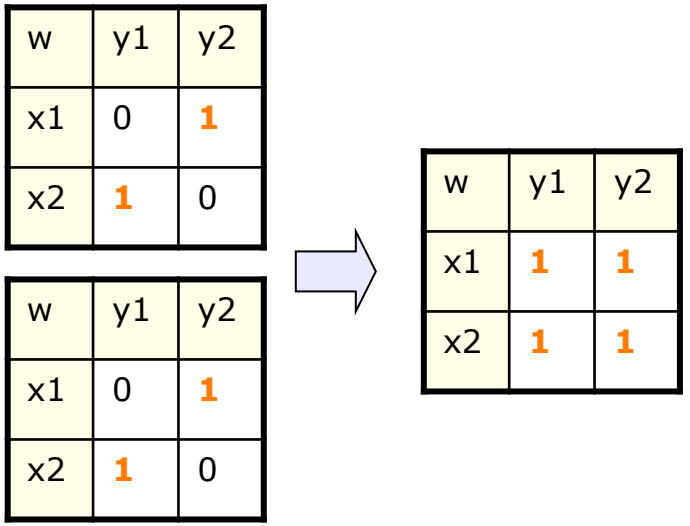
```
RooDataSet d1("d1","d1",RooArgSet(x,y,z) ;

RooDataSet* d2 = d1.reduce(RooArgSet(x,y));

RooDataSet* d3 = d1.reduce("x>1");
```



Adding and reducing **binned** datasets

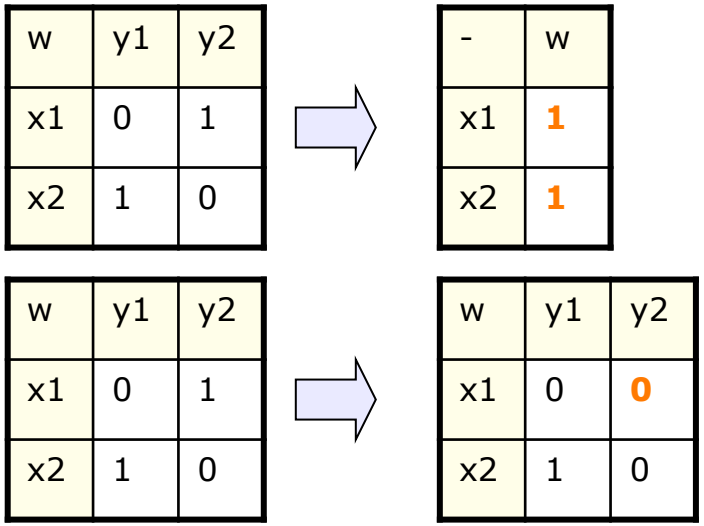
- Adding

```
RooDataHist d1("d1","d1",
                RooArgSet(x,y));
RooDataHist d2("d2","d2",
                RooArgSet(x,y));

d1.add(d2) ;
```



- Reducing

```
RooDataHist d1("d1","d1",
                RooArgSet(x,y) ;

RooDataHist* d2 =
     d1.reduce(x);

RooDataHist* d3 =
     d1.reduce("x>1");
```

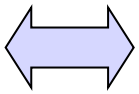


Datasets and discrete observables

- Discrete observables play an important role in management of datasets
  - Useful to classify ‘sub datasets’ inside datasets
  - Can collapse multiple, logically separate datasets into a single dataset by adding them and labeling the source with a discrete observable
  - Allows to express operations such a simultaneous fits as operation on a single dataset

Dataset A
X
5.0
3.7
1.2
4.3

Dataset B
X
5.0
3.7
1.2



Dataset A+B	
X	**source**
5.0	A
3.7	A
1.2	A
4.3	A
5.0	B
3.7	B
1.2	B

Discrete variables in RooFit – RooCategory

- Properties of RooCategory variables
  - Finite set of named states → self documenting
  - Optional integer code associated with each state

At creation, a category has no states

Add states with a label *and index*

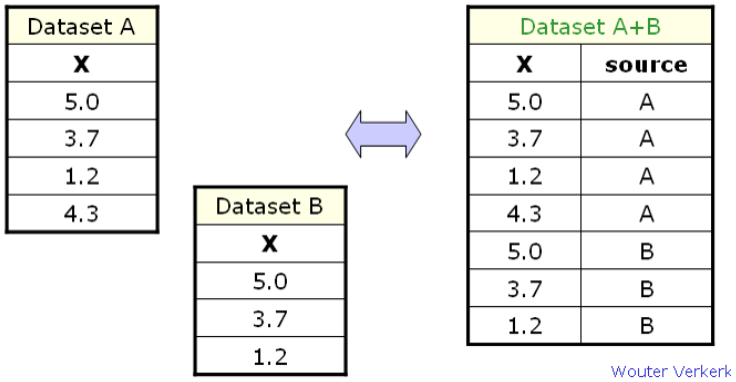
Add states with a label only. *Indices will be automatically assigned*

```
// Define a cat. with explicitly numbered states
RooCategory b0flav("b0flav","B0 flavour") ;
b0flav.defineType("B0",-1) ;
b0flav.defineType("B0bar",1) ;

// Define a category with labels only
RooCategory tagCat("tagCat","Tagging technique") ;
tagCat.defineType("Lepton") ;
tagCat.defineType("Kaon") ;
tagCat.defineType("NetTagger-1") ;
tagCat.defineType("NetTagger-2") ;
```

- Used for classification of data, or to describe occasional discrete fundamental observable (e.g. B^0 flavor)

Datasets and discrete observables – part 2

- Example of appending datasets with label attachment



Dataset A
x
5.0
3.7
1.2
4.3

Dataset B
x
5.0
3.7
1.2

Dataset A+B	
x	source
5.0	A
3.7	A
1.2	A
4.3	A
5.0	B
3.7	B
1.2	B

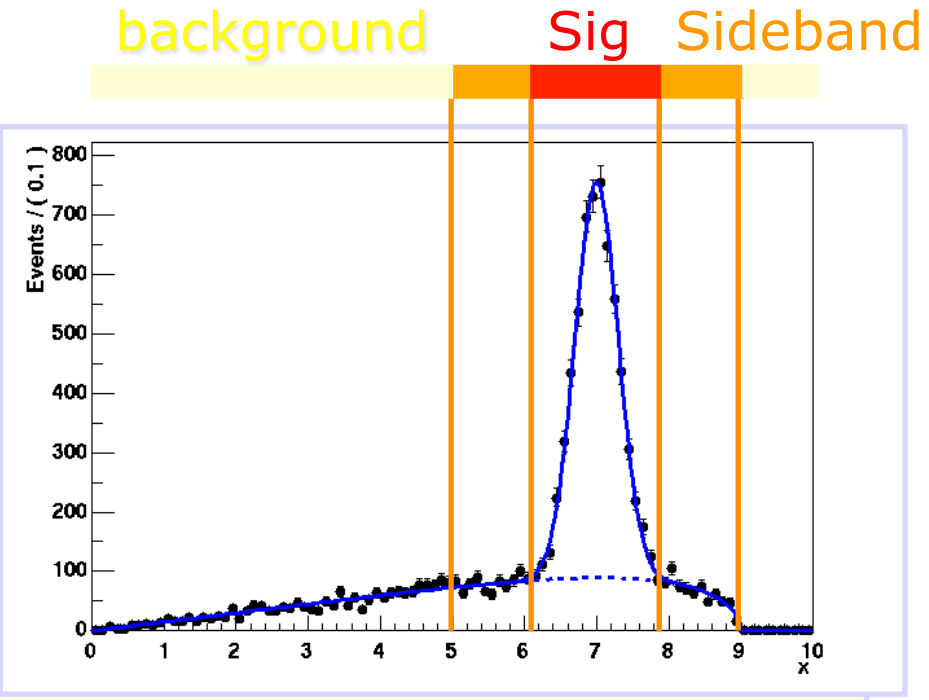
Wouter Verkerk

```
RooCategory c("c","source")
c.defineType("A") ;
c.defineType("B") ;

// Add column with source label
c.setLabel("A") ; dA->addColumn(c) ;
c.setLabel("B") ; dB->addColumn(c) ;

// Make combined dataset
RooDataSet* dAB = dA->Clone("dAB") ;
dAB->append(*dB) ;
```

- But can also derive classification from info within dataset
  - E.g. (10<x<20 = "signal", 0<x<10 | 20<x<30 = "sideband")
  - Encode classification using real→discrete mapping functions

A universal real→discrete mapping function

- Class **RooThresholdCategory** maps ranges of input **RooRealVar** to states of a **RooCategory**




```
// Mass variable
RooRealVar m("m","mass,0,10.);

// Define threshold category
RooThresholdCategory region("region","Region of M",m,"Background");
region.addThreshold(9.0, "SideBand") ;
region.addThreshold(7.9, "Signal") ;
region.addThreshold(6.1,"SideBand") ;
region.addThreshold(5.0,"Background") ;
```

Default state

Define region boundaries

Discrete→Discrete mapping function

- **`RooMappedCategory`** provides cat → cat mapping

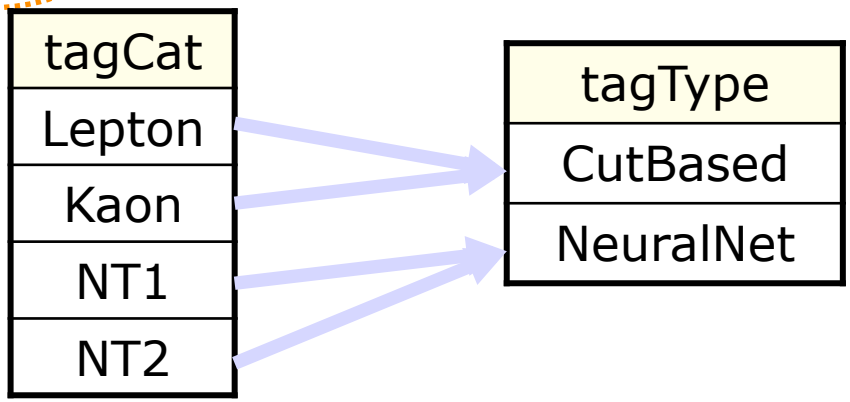
Define input category

Create mapped category

Add mapping rules

```
RooCategory tagCat("tagCat","Tagging category") ;
tagCat.defineType("Lepton") ;
tagCat.defineType("Kaon") ;
tagCat.defineType("NetTagger-1") ;
tagCat.defineType("NetTagger-2") ;

RooMappedCategory tagType("tagType","type",tagCat) ;

tagType.map("Lepton","CutBased") ;
tagType.map("Kaon","CutBased") ;
tagType.map("NT*","NeuralNet") ;
```

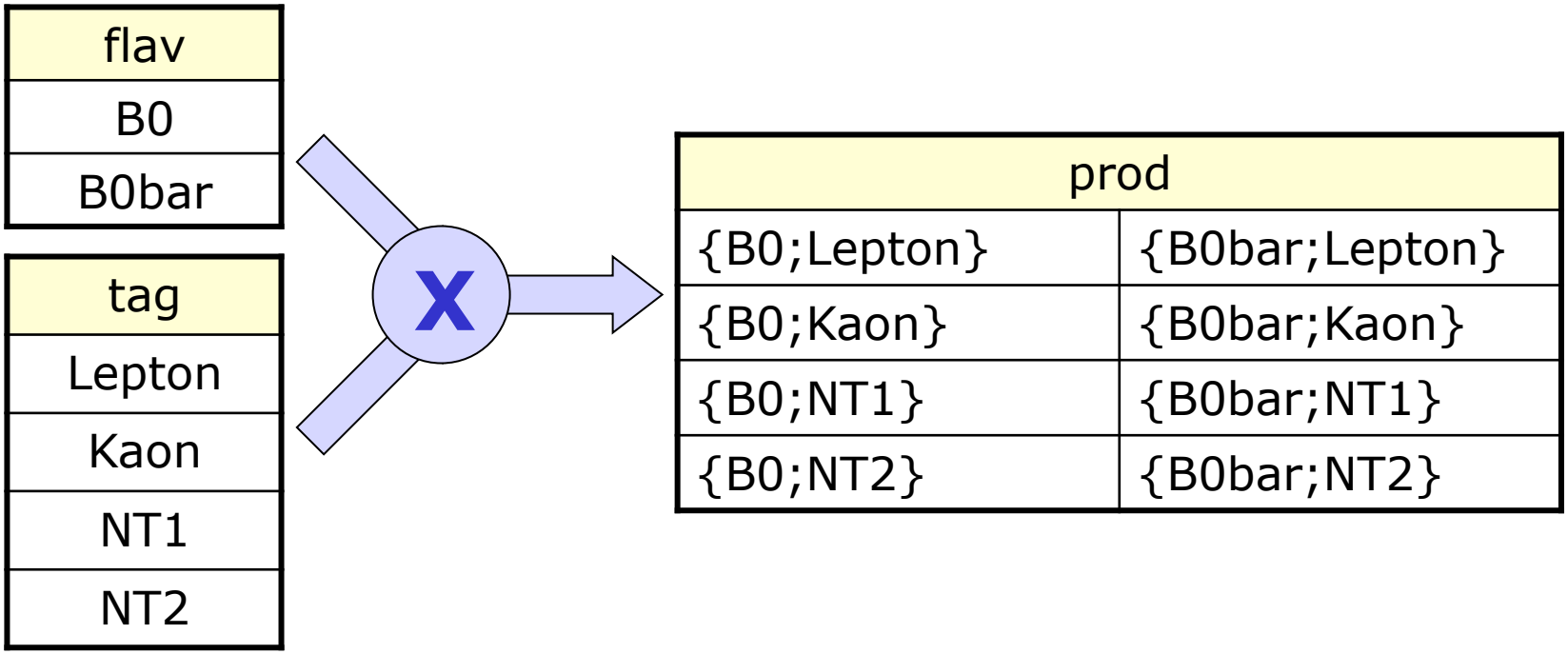
Wildcard expressions allowed



Discrete multiplication function

- **`RooSuperCategory/RooMultiCategory`** provides category multiplication

```
// Define 'product' of tagCat and runBlock
RooSuperCategory prod("prod","prod",RooArgSet(tag,flav))
```



Exploring discrete data

- Like real variables of a dataset can be plotted, discrete variables can be tabulated

Tabulate contents of dataset by category state

```
RooTable* table=data->table(b0flav) ;
table->Print() ;

Table b0flav : aData
  +-------+------+
  |    B0 | 4949 |
  | B0bar | 5051 |
  +-------+------+
```

Extract contents by label

```
Double_t nB0 = table->get("B0") ;
```

Extract contents fraction by label

```
Double_t b0Frac = table->getFrac("B0");
```

Tabulate contents of selected part of dataset

```
data->table(tagCat,"x>8.23")->Print() ;

  Table tagCat : aData(x>8.23)
  +-------------+-----+
  |      Lepton | 668 |
  |        Kaon | 717 |
  | NetTagger-1 | 632 |
  | NetTagger-2 | 616 |
  +-------------+-----+
```

Exploring discrete data

- *Discrete functions*, built from categories in a dataset can be tabulated likewise

Tabulate `RooSuperCategory` states

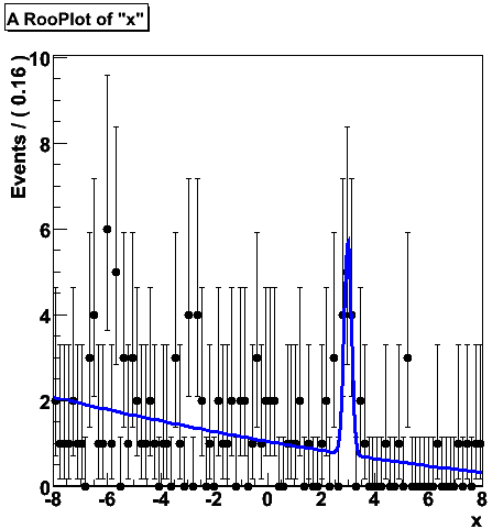
```
data->table(b0Xtcat)->Print() ;

  Table b0Xtcat : aData
  +---------------------+------+
  |          {B0;Lepton} | 1226 |
  |      {B0bar;Lepton} | 1306 |
  |           {B0;Kaon} | 1287 |
  |         {B0bar;Kaon} | 1270 |
  |     {B0;NetTagger-1} | 1213 |
  | {B0bar;NetTagger-1} | 1261 |
  |     {B0;NetTagger-2} | 1223 |
  | {B0bar;NetTagger-2} | 1214 |
  +---------------------+------+
```

Tabulate `RooMappedCategory` states

```
data->table(tcatType)->Print() ;

  Table tcatType : aData
  +----------------+------+
  |        Unknown |    0 |
  |      Cut based | 5089 |
  | Neural Network | 4911 |
  +----------------+------+
```

Fitting multiple datasets simultaneously

- Simultaneous fitting efficient solution to incorporate information from control sample into signal sample
- Example problem: search rare decay
  - Signal dataset has small number entries.

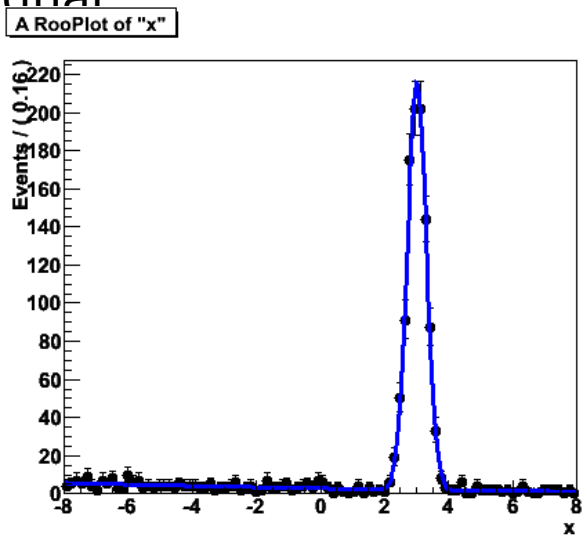


Par	FinalValue +/-	Error
a0	-1.0544e-01 +/-	2.88e-02
a1	2.2698e-03 +/-	4.92e-03
nbkg	1.0933e+02 +/-	1.07e+01
nsig	1.0680e+01 +/-	3.92e+00
mean	2.9787e+00 +/-	6.25e-02
width	1.3764e-01 +/-	6.29e-02

  - Statistical uncertainty on shape in fit contributes significantly to uncertainty on fitted number of signal events
  - However can constrain shape of signal from control sample (e.g. another decay with similar properties that is not rare), so no need to relay on simulations

Fitting multiple datasets simultaneously

- Fit to control sample yields accurate information on shape of signal



Par	FinalValue +/-	Error
a0	-9.9212e-02 +/-	1.75e-02
a1	3.3116e-03 +/-	3.57e-03
nbkg	3.0406e+02 +/-	1.83e+01
nsig	9.9594e+02 +/-	3.21e+01
m	3.0098e+00 +/-	9.83e-03
s	2.9891e-01 +/-	7.39e-03

- Q: What is the most practical way to combine shape measurement on control sample to measurement of signal on physics sample of interest

- A: Perform a simultaneous fit
  - Automatic propagation of errors & correlations
  - Combined measurement
    (i.e. error will reflect contributions from both physics sample and control sample

Discrete observable as data subset classifier

- Likelihood level definition of a simultaneous fit

$$-\log(L) = \sum_{i=1,n} -\log(PDF_A(D_A^i)) + \sum_{i=1,m} -\log(PDF_B(D_B^i))$$

- PDF level definition of a simultaneous fit

$$-\log(L) = \sum_{i=1,n} -\log(simPDF(D_{A+B}^i))$$

RooSimultaneous implements 'switch' PDF:

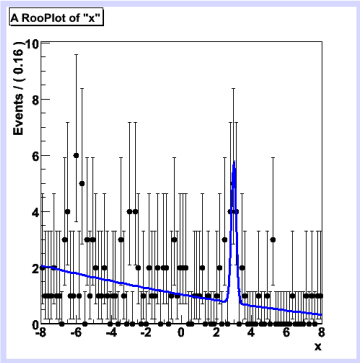
```
case (indexCat) {
  A: return pdfA ;
  B: return pdfB ;
}
```

Likelihood of switchPdf with composite dataset *automatically* constructs sum of likelihoods above

Dataset A+B	
X	**source**
5.0	A
3.7	A
1.2	A
4.3	A
5.0	B
3.7	B
1.2	B

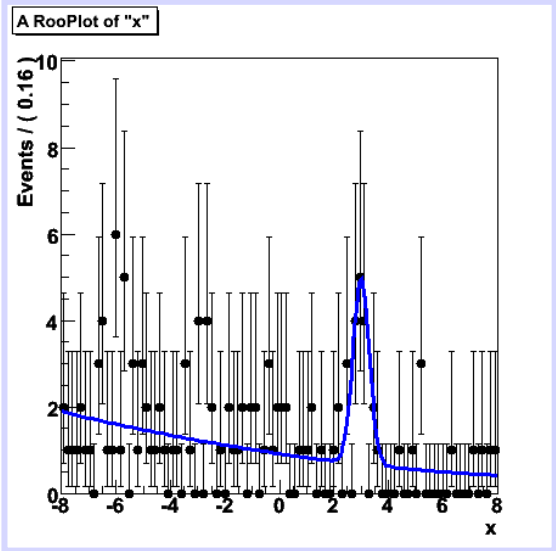
Using RooSimultaneous to implement preceding example

Fit to signal data




```
RooCategory c("c","c") ;
c.defineType("control") ;
c.defineType("physics") ;

RooSimultaneous sim_model("sim_model","",c) ;
sim_model.addPdf(model_phys,"physics") ;
sim_model.addPdf(model_ctrl,"control") ;

sim_model.fitTo(*d,Extended()) ;
```

Combined fit



Signal shape constrained from control sample

Relative error on Nsig improved from 37% to 32%

Parameter	FinalValue +/-	Error
a0_ctrl	-8.0947e-02 +/-	1.47e-02
a0_phys	-1.1825e-01 +/-	3.26e-02
a1_ctrl	2.1004e-04 +/-	3.12e-03
a1_phys	4.2259e-03 +/-	5.55e-03
nbkg_ctrl	3.1054e+02 +/-	1.86e+01
nbkg_phys	1.0633e+02 +/-	1.06e+01
nsig_ctrl	9.8946e+02 +/-	3.20e+01
nsig_phys	1.3647e+01 +/-	4.44e+00
m	2.9983e+00 +/-	9.69e-03
s	2.9255e-01 +/-	7.53e-03

6 Likelihood calculation & minimization

- *Details on the likelihood calculation*
- *Using MINUIT and RooMinuit*
- *Profile likelihoods*

Fitting and likelihood minimization

- What happens when you do **`pdf->fitTo(*data)`**
  - 1) Construct object representing –log of (extended) likelihood
  - 2) Minimize likelihood w.r.t floating parameters using MINUIT

- Can also do these two steps explicitly by hand

```
// Construct function object representing -log(L)
RooNLLVar nll("nll","nll",pdf,data) ;

// Minimize nll w.r.t its parameters
RooMinuit m(nll) ;
m.migrad() ;
m.hesse() ;
```

Constructing the likelihood function

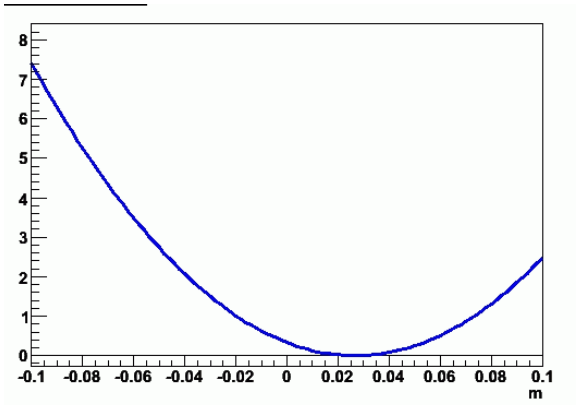
- Class **RooNLLVar** works universally for all p.d.f.s and all types of data
  - Binned data → Binned likelihood
  - Unbinned data → Unbinned likelihood

- Can add named arguments to constructor to control details of likelihood definition and mode of calculation

```
RooNLLVar nll("nll","nll",pdf,data,Extended()) ;
```

  - Works like a regular RooFit function object, i.e. can retrieve value and make plots as usual

```
Double_t val = nll.getVal() ;
RooArgSet* vars = nll.getVariables()

RooPlot* frame = p.frame() ;
nll.plotOn(frame) ;
```



Constructing the likelihood function

- All of the following RooNLLVar options are available under identical name in **`pdf->fitTo()`**

- Definition options
  - **`Extended()`** – Add extended likelihood term with N_{exp} taken from p.d.f and N_{obs} taken from data
  - **`ConditionalObservable(obs)`** – Treat given observables of pdf as conditional observables

- Mode of calculation options
  - **`Verbose()`** – Additional information is printed on how the likelihood calculation is set up
  - **`NumCPU(N)`** – Parallelize calculation of likelihood over N processes. Nice if you have a dual-quad core box (actual speedup is about factor 7.6 for N=8)

Constructing a χ^2 function

- Along similar lines it is also possible to construct a χ^2 function
  - Only takes binned datasets (class **RooDataHist**)
  - Normalized p.d.f is multiplied by Ndata to obtain χ^2

```
// Construct function object representing -log(L)
RooNLLVar chi2("chi2","chi2",pdf,data) ;

// Minimize nll w.r.t its parameters
RooMinuit m(chi2) ;
m.migrad() ;
m.hesse() ;
```

  - MINUIT error definition for χ^2 automatically adjusted to 1 (it is 0.5 for likelihoods) as default error level is supplied through virtual method of function base class **RooAbsReal**

Automatic optimizations in the calculation of the likelihood

- Several automatic computational optimizations are applied the calculation of likelihoods inside RooNLLVar
  - Components that have all constant parameters are pre-calculated
  - Dataset variables not used by the PDF are dropped
  - PDF normalization integrals are only recalculated when the ranges of their observables or the value of their parameters are changed
  - Simultaneous fits: When a parameters changes only parts of the total likelihood that depend on that parameter are recalculated
    - Lazy evaluation: calculation only done when intergal value is requested
- Applicability of optimization techniques is re-evaluated for each use
  - Maximum benefit for each use case
- 'Typical' large-scale fits see significant speed increase
  - Factor of 3x – 10x not uncommon.

Features of class RooMinuit

- Class RooMinuit is an *interface* to the ROOT implementation of the MINUIT minimization and error analysis package.

- RooMinuit takes care of
  - Passing value of miminized RooFit function to MINUIT
  - Propagated changes in parameters both from **`RooRealVar`** to MINUIT and back from MINUIT to **`RooRealVar`**, i.e. it keeps the state of RooFit objects synchronous with the MINUIT internal state
  - Propagate error analysis information back to **`RooRealVar`** parameters objects
  - Exposing high-level MINUIT operations to RooFit uses (MIGRAD,HESSE,MINOS) etc…
  - Making optional snapshots of complete MINUIT information (e.g. convergence state, full error matrix etc)

A brief description of MINUIT functionality

- MIGRAD
  - Find function minimum. Calculates function gradient, follow to (local) minimum, recalculate gradient, iterate until minimum found
    - To see what MIGRAD does, it is very instructive to do RooMinuit::setVerbose(1). It will print a line for each step through parameter space
  - Number of function calls required depends greatly on number of floating parameters, distance from function minimum and shape of function
- HESSE
  - Calculation of error matrix from 2nd derivatives at minimum
  - Gives symmetric error. Valid in assumption that likelihood is (locally parabolic)

$$\hat{\sigma}(p)^2 = \hat{V}(p) = \left(\frac{d^2 \ln L}{d^2 p} \right)^{-1}$$

  - Requires roughly N^2 likelihood evaluations (with N = number of floating parameters)

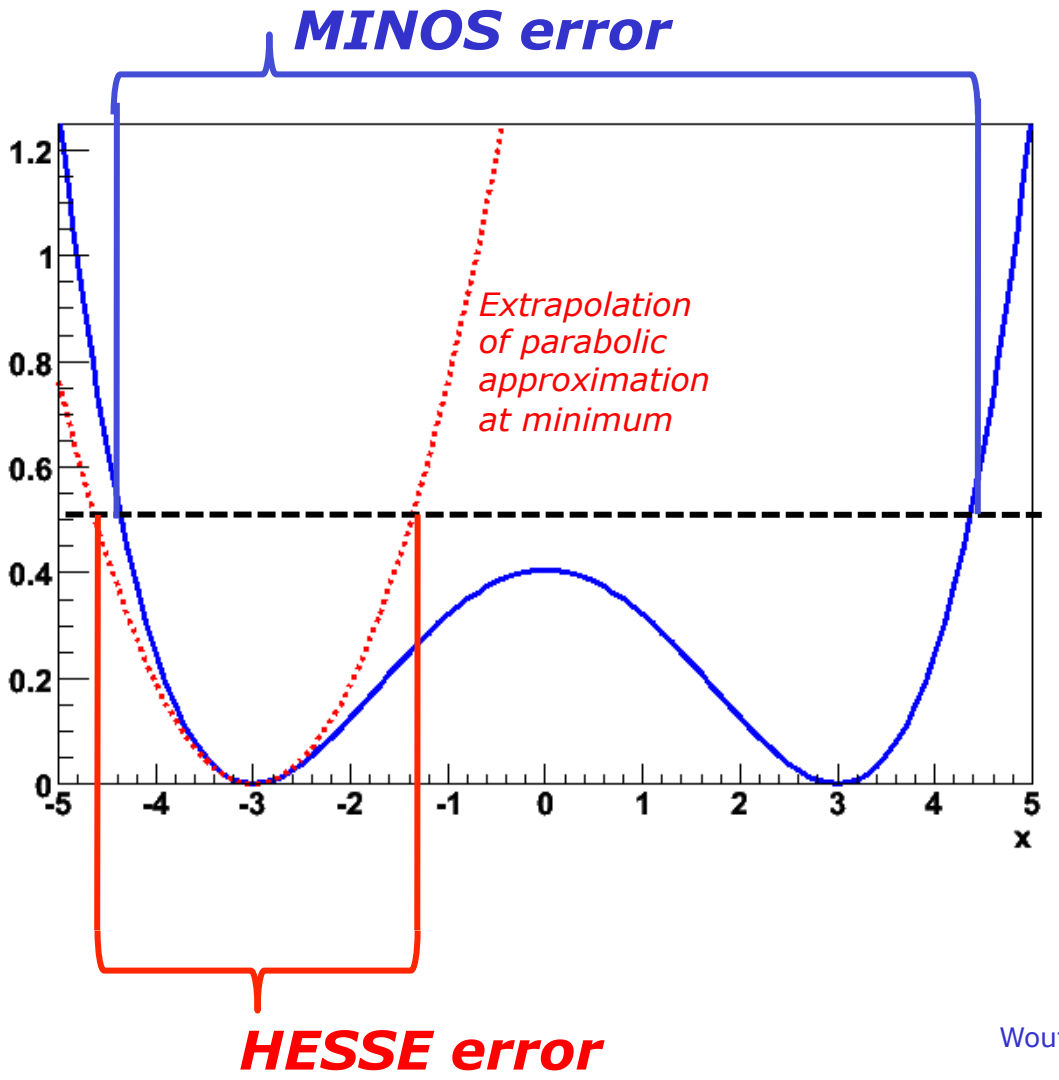
A brief description of MINUIT functionality

- MINOS
  - Calculate errors by explicit finding points (or contour for >1D) where Δ-log(L)=0.5
  - Reported errors can be asymmetric
  - Can be very expensive in with large number of floating parameters

- CONTOUR
  - Find contours of equal Δ-log(L) in two parameters and draw corresponding shape
  - Mostly an interactive analysis tool

Illustration of difference between HESSE and MINOS errors

- 'Pathological' example likelihood with multiple minima and non-parabolic behavior



Demonstration of RooMinuit use

```
// Start Minuit session on above nll
RooMinuit m(nll) ;

// MIGRAD likelihood minimization
m.migrad() ;

// Run HESSE error analysis
m.hesse() ;

// Set sx to 3, keep fixed in fit
sx.setVal(3) ;
sx.setConstant(kTRUE) ;

// MIGRAD likelihood minimization
m.migrad() ;

// Run MINOS error analysis
m.minos()

// Draw 1,2,3 'sigma' contours in sx,sy
m.contour(sx,sy) ;
```

Minuit function MIGRAD

- Purpose: find minimum

```
**********
**    13 **MIGRAD        1000           1
**********
(some output omitted)
MIGRAD MINIMIZATION HAS CONVERGED.
MIGRAD WILL VERIFY CONVERGENCE AND ERROR MATRIX.
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=257.304 FROM MIGRAD    STATUS=CONVERGED      31 CALLS          32 TOTAL
                     EDM=2.36773e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  mean        8.84225e-02   3.23862e-01   3.58344e-04  -2.24755e-02
  2  sigma       3.20763e+00   2.39540e-01   2.78628e-04  -5.34724e-02
                               ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=0.5
 1.049e-01  3.338e-04
 3.338e-04  5.739e-02
PARAMETER  CORRELATION COEFFICIENTS
      NO.   GLOBAL      1      2
       1  0.00430   1.000  0.004
       2  0.00430   0.004  1.000
```

Progress information, watch for errors here

Parameter values and approximate errors reported by MINUIT

Error definition (in this case 0.5 for a likelihood fit)

Minuit function MIGRAD

- Purpose: find minimum

```
**********
**    13 **MIGR
**********
(some output c
MIGRAD MINIMIZ
MIGRAD WILL VERIF      ERGENCE AND ERROR MATRIX.
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=257.304 FROM MIGRAD     STATUS=CONVERGED       31 CALLS          32 TOTAL
                    EDM=2.36773e-06     STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                          STEP          FIRST
 NO.   NAME      VALUE            ERROR                 SIZE      DERIVATIVE
  1  mean         8.84225e-02   3.23862e-01   3.58344e-04  -2.24755e-02
  2  sigma        3.20763e+00   2.39540e-01   2.78628e-04  -5.34724e-02
                                ERR DEF= 0.5
EXTERNAL ERROR MATRIX.     NDIM=  25    NPAR=  2    ERR DEF=0.5
 1.049e-01  3.338e-04
 3.338e-04  5.739e-02
PARAMETER  CORRELATION COEFFICIENTS
      NO.   GLOBAL      1      2
       1  0.00430   1.000  0.004
       2  0.00430   0.004  1.000
```

Value of $\chi 2$ or likelihood at minimum

(NB: χ^2 values are not divided by $N_{d.o.f}$)

Approximate
Error matrix
And covariance matrix

Minuit function MIGRAD

- Purpose: find minimu

Status:
Should be 'converged' but can be 'failed'

Estimated Distance to Minimum
should be small O(10^{-6})

Error Matrix Quality
should be 'accurate', but can be
'approximate' in case of trouble

```
**********
**    13 **MIGRAD        1000
**********
(some output omitted)
MIGRAD MINIMIZATION HAS CONVERGED.
MIGRAD WILL VERIFY CONVERGENCE AND ERROR MATRIX.
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=257.304 FROM MIGRAD    STATUS=CONVERGED      31 CALLS          32 TOTAL
                    EDM=2.36773e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  mean         8.84225e-02   3.23862e-01   3.58344e-04  -2.24755e-02
  2  sigma        3.20763e+00   2.39540e-01   2.78628e-04  -5.34724e-02
                               ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=0.5
 1.049e-01  3.338e-04
 3.338e-04  5.739e-02
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL      1      2
       1  0.00430   1.000  0.004
       2  0.00430   0.004  1.000
```

Minuit function HESSE

- Purpose: calculate error matrix from $\frac{d^2L}{dp^2}$

```
**********
**    18 **HESSE        1000
**********
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=257.304 FROM HESSE     STATUS=OK                          OTAL
                    EDM=2.36534e-06    STRATI                CURATE
 EXT PARAMETER                                   NTERNAL      INTERNAL
 NO.   NAME      VALUE            ERROR          STEP SIZE      VALUE
  1   mean         8.84225e-02   3.23861e-01   7.16689e-05   8.84237e-03
  2   sigma        3.20763e+00   2.39539e-01   5.57256e-05   3.26535e-01
                                ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=0.5
 1.049e-01  2.780e-04
 2.780e-04  5.739e-02
PARAMETER  CORRELATION COEFFICIENTS
      NO.   GLOBAL      1      2
       1  0.00358   1.000  0.004
       2  0.00358   0.004  1.000
```

Symmetric errors calculated from 2nd derivative of −ln(L) or χ^2

Minuit function HESSE

- Purpose: calculate error matrix from $\frac{d^2L}{dp^2}$

```
**********
**
***
COV                                  UCCESSFULLY
FCN                                  TUS=OK              10 CALLS          42 TOTAL
                                     4e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
 EX                                                  INTERNAL      INTERNAL
 NO                                    ERROR         STEP SIZE       VALUE
  1                                 3.23861e-01    7.16689e-05   8.84237e-03
  2   sig                           2.39539e-01    5.57256e-05   3.26535e-01
                                 ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=0.5
 1.049e-01  2.780e-04
 2.780e-04  5.739e-02
PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL      1      2
        1  0.00358   1.000  0.004
        2  0.00358   0.004  1.000
```

Error matrix (Covariance Matrix) calculated from

$$V_{ij} = \left(\frac{d^2(-\ln L)}{dp_i dp_j} \right)^{-1}$$

Minuit function HESSE

- Purpose: calculate error matrix from $\frac{d^2L}{dp^2}$

```
**********
**    18 **HESSE        1000
**********
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=257.304 FROM HESSE     STATUS=OK             10 CALLS          42 TOTAL
                    EDM=2.36534e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   INTERNAL      INTERNAL
 NO.   NAME      VALUE            ERROR          STEP SIZE       VALUE
  1   mean         8.84225e-02                                  8.84237e-03
  2   sigma        3.20763e+00                                  3.26535e-01

EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=0.5
 1.049e-01  2.780e-04
 2.780e-04  5.739e-02
PARAMETER  CORRELATION COEFFICIENT
       NO.   GLOBAL      1      2
        1  0.00358   1.000  0.004
        2  0.00358   0.004  1.000
```

Correlation matrix ρ_{ij} calculated from

$$V_{ij} = \sigma_i \sigma_j \rho_{ij}$$

Minuit function HESSE

- Purpose: calculate error matrix from $\frac{d^2L}{dp^2}$

```
**********
**    18 **HESSE        1000
**********
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=257.304 FROM HESSE     STATUS=OK              10 CALLS          42 TOTAL
                    EDM=2.36534e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   INTERNAL      INTERNAL
 NO.   NAME      VALUE            ERROR          STEP SIZE       VALUE
  1   mean                                      7.16689e-05   8.84237e-03
  2   sigma                                     5.57256e-05   3.26535e-01

EXTERNAL ERROR                                  2    ERR DEF=0.5
 1.049e-01  2.
 2.780e-04  5.739e-
PARAMETER  CORRELA   N COEFFICIENTS
      NO.   GLOBAL      1      2
        1  0.00358   1.000  0.004
        2  0.00358   0.004  1.000
```

Global correlation vector: correlation of each parameter with all other parameters

Minuit function MINOS

- Error analysis through Δnll contour finding

```
**********
**    23 **MINOS         1000
**********
FCN=257.304 FROM MINOS     STATUS=SUCCESSFUL     52 CALLS           94 TOTAL
                    EDM=2.36534e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                          PARABOLIC        MINOS ERRORS
 NO.   NAME       VALUE                  ERROR       NEGATIVE       POSITIVE
  1   mean          8.84225e-02   3.23861e-01  -3.24688e-01    3.25391e-01
  2   sigma         3.20763e+00   2.39539e-01  -2.23321e-01    2.58893e-01
                                 ERR DEF= 0.5
```

Symmetric error

(repeated result from HESSE)

MINOS error
Can be asymmetric

(in this example the 'sigma' error is slightly asymmetric)

What happens if there are problems in the NLL calculation

- Sometimes the likelihood cannot be evaluated do due an error condition.
  - PDF Probability is zero, or less than zero at coordinate where there is a data point 'infinitely improbable'
  - Normalization integral of PDF evaluates to zero
- Most problematic during MINUIT operations. How to handle error condition
  - All error conditions are gather and reported in consolidated way by RooMinuit
  - Since MINUIT has no interface deal with such situations, RooMinuit passes instead a large value to MINUIT to force it to retreat from the region of parameter space in which the problem occurred

```
[#0] WARNING:Minization -- RooFitGlue: Minimized function has error status.
Returning maximum FCN so far (99876) to force MIGRAD to back out of this region.
Error log follows. Parameter values: m=-7.397
RooGaussian::gx[ x=x mean=m sigma=sx ] has 3 errors
```

What happens if there are problems in the NLL calculation

- Can request more verbose error logging to debug problem
  - Add PrintEvalError(N) with N>1

```
[#0] WARNING:Minization -- RooFitGlue: Minimized function has error status.
Returning maximum FCN so far (-1e+30) to force MIGRAD to back out of this region.
Error log follows
Parameter values: m=-7.397
RooGaussian::gx[ x=x mean=m sigma=sx ]
     getLogVal() top-level p.d.f evaluates to zero or negative number
                 @ x=x=9.09989, mean=m=-7.39713, sigma=sx=0.1
     getLogVal() top-level p.d.f evaluates to zero or negative number
                 @ x=x=6.04652, mean=m=-7.39713, sigma=sx=0.1
     getLogVal() top-level p.d.f evaluates to zero or negative number
                 @ x=x=2.48563, mean=m=-7.39713, sigma=sx=0.1
```

Practical estimation – Fit converge problems

- Sometimes fits don't converge because, e.g.
  - MIGRAD unable to find minimum
  - HESSE finds negative second derivatives (which would imply negative errors)
- Reason is usually numerical precision and stability problems, but
  - The underlying cause of fit stability problems is usually by **highly correlated parameters** in fit
- HESSE correlation matrix in primary investigative tool

```
PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL      1       2
         1  0.99835   1.000   0.998
         2  0.99835   0.998   1.000
```

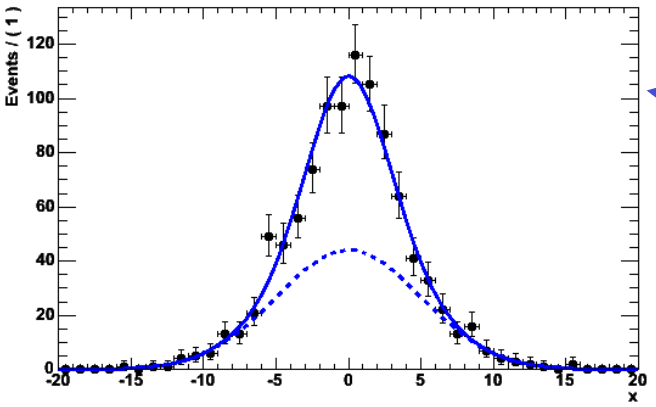
Signs of trouble…

  - In limit of 100% correlation, the usual point solution becomes a line solution (or surface solution) in parameter space. Minimization problem is no longer well defined

Mitigating fit stability problems

- Strategy I – More orthogonal choice of parameters
  - Example: fitting sum of 2 Gaussians of similar width

$$F(x; f, m, s_1, s_2) = fG_1(x; s_1, m) + (1-f)G_2(x; s_2, m)$$



HESSE correlation matrix

PARAMETER	CORRELATION COEFFICIENTS				
NO.	GLOBAL	[f]	[m]	[s1]	[s2]
[f]	0.96973	1.000	-0.135	0.918	0.915
[m]	0.14407	-0.135	1.000	-0.144	-0.114
[s1]	0.92762	0.918	-0.144	1.000	0.786
[s2]	0.92486	0.915	-0.114	0.786	1.000

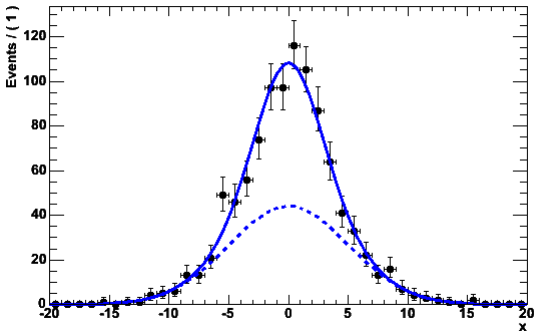
Widths s_1,s_2 strongly correlated fraction f

Mitigating fit stability problems

- Different parameterization:

$$fG_1(x;s_1,m_1)+(1-f)G_2(x;s_1\cdot s_2,m_2)$$

```
PARAMETER  CORRELATION COEFFICIENTS
      NO.   GLOBAL      [f]     [m]    [s1]    [s2]
 [ f]  0.96951    1.000 -0.134   0.917 -0.681
 [ m]  0.14312   -0.134  1.000 -0.143  0.127
 [s1]  0.98879    0.917 -0.143  1.000 -0.895
 [s2]  0.96156   -0.681  0.127 -0.895  1.000
```



- Correlation of width s2 and fraction f reduced from 0.92 to 0.68
- Choice of parameterization matters!

- Strategy II – Fix all but one of the correlated parameters
  - If floating parameters are highly correlated, some of them may be redundant and not contribute to additional degrees of freedom in your model

Browsing fit results with `RooFitResult`

- As fits grow in complexity (e.g. 45 floating parameters), number of output variables increases
  - Need better way to navigate output that MINUIT screen dump
- **`RooFitResult`** holds complete snapshot of fit results
  - Constant parameters
  - Initial and final values of floating parameters
  - Global correlations & full correlation matrix
  - Returned from `RooAbsPdf::fitTo()` when "`r`" option is supplied
- Compact & verbose printing mode

Compact Mode

Constant parameters omitted in compact mode

Alphabetical parameter listing

```
fitres->Print() ;

  RooFitResult: min. NLL value: 1.6e+04, est. distance to min: 1.2e-05

    Floating Parameter    FinalValue +/-  Error
  --------------------  --------------------------
                argpar   -4.6855e-01 +/-  7.11e-02
                g2frac    3.0652e-01 +/-  5.10e-03
                 mean1    7.0022e+00 +/-  7.11e-03
                 mean2    1.9971e+00 +/-  6.27e-03
                 sigma    2.9803e-01 +/-  4.00e-03
```

Browsing fit results with `RooFitResult`

Verbose printing mode

```
fitres->Print("v") ;

  RooFitResult: min. NLL value: 1.6e+04, est. distance to min: 1.2e-05

  Constant Parameter    Value
  --------------------  ------------
                cutoff    9.0000e+00
                g1frac    3.0000e-01

    Floating Parameter  InitialValue    FinalValue +/-  Error      GblCorr.
  --------------------  ------------  --------------------------  --------
                argpar   -5.0000e-01   -4.6855e-01 +/-  7.11e-02  0.191895
                g2frac    3.0000e-01    3.0652e-01 +/-  5.10e-03  0.293455
                 mean1    7.0000e+00    7.0022e+00 +/-  7.11e-03  0.113253
                 mean2    2.0000e+00    1.9971e+00 +/-  6.27e-03  0.100026
                 sigma    3.0000e-01    2.9803e-01 +/-  4.00e-03  0.276640
```

Constant parameters listed separately

Initial,final value and global corr. listed side-by-side

Correlation matrix accessed separately

Working with *profile* likelihoods

- Given a likelihood, the profile likelihood is defined as
  - Where the hatted quantities represent the value of that parameter at which the –log(L) is minimal

$$PL(p) = \frac{L(p,\hat{q})}{L(\hat{p},\hat{q})}$$

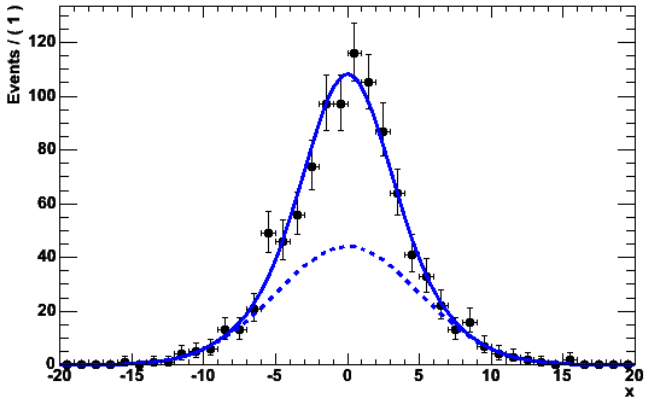
- Represented in RooFit with class **`RooProfileLL`**

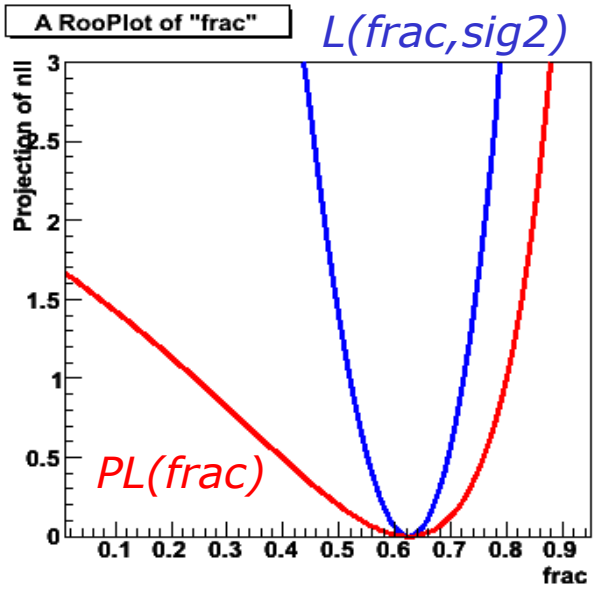
```
// First make regular likelihood object (with parameters frac and sg2)
RooNLLVar nll("nll","nll",model,*data) ;

// Now make profile likelihood in frac
RooProfileLL pll_frac("pll_frac","pll",nll,frac) ;
```

  - A profile likelihood is a regular function object in RooFit, you can plot it with plotOn() etc.
  - *However it is expensive*, as each function evaluation requires a MIGRAD minimization step!

Plotting a profile likelihood

- Example with strong correlations:
  - sum of two Gaussians with similar widths and floating fraction

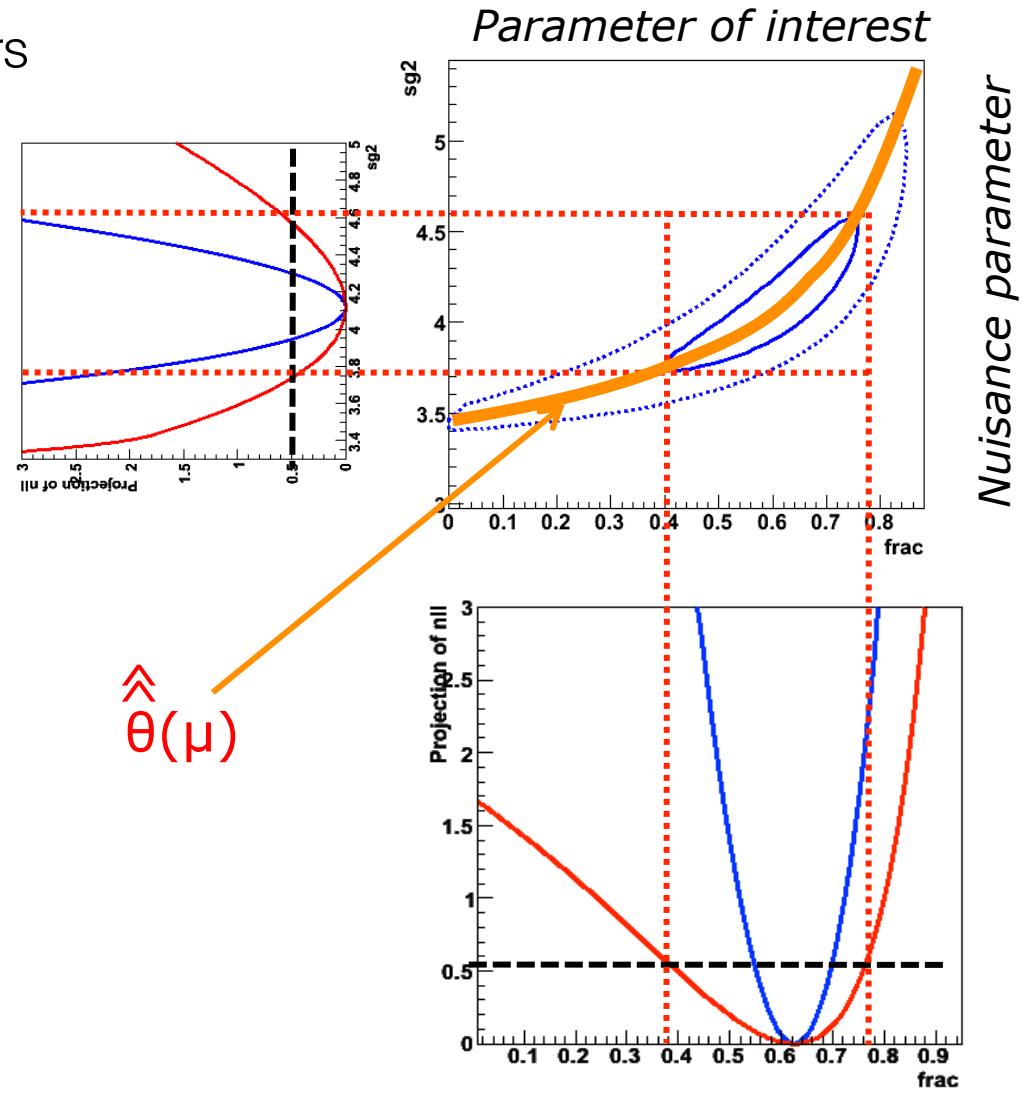




- Profile likelihood much broader than likelihood because changing the width the 2nd gaussian can largely compensate for off-value of fraction

Plotting a profile likelihood

- Picture with 2 parameters



Scaling up
‘Big modelling’

Collaborative modeling

- Tutorials so far described advanced techniques to build models inside a ROOT session that describe a single (multi-dimensional) dataset
  - With some exploration of simultaneous fits describing multiple datasets
- The 'build-model-on-the-fly' approach ultimately limits scope/ambition of the model
- To overcome this limitation RooFit offers 'the workspace' **– the ability to persist any RooFit model built in memory to a binary ROOT file**, and resurrect it with just 2 lines of code in another ROOT session
- The most powerful feature of RooFit
  - Persistence and resurrection works trivially for *any* model regardless of complexity
  - Simplifies the process of combining analysis channels*: combination can be done later a posteriori without any loss of model accuracy*
  - (Teams of) Physicists can collaborate and divide work efficiently
  - Allows to *completely* factorize statistical inference tools from model building tools (since inference is fully procedure once model and specs are fixed). RooStats provide truly universal tools for limit setting, confidence intervals etc.

The Workspace

How is Higgs discovery different from a simple fit?

Gaussian + polynomial *Higgs combination model*

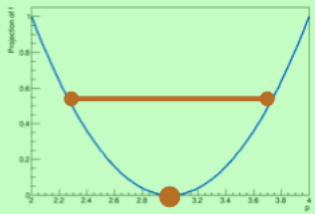
Design goal:

Separate **building of Likelihood model** as much as possible from statistical analysis **using the Likelihood model**

→ More modular software design

→ 'Plug-and-play with statistical techniques

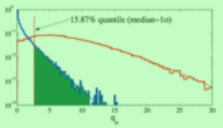
→ Factorizes work in collaborative effort



ML estimation of parameters μ,θ using MINUIT (MIGRAD, HESSE, MINOS)

$\mu = 5.3 \pm 1.7$

$$\lambda_\mu(\vec{N}_{ZZ},\vec{N}_{WW},\vec{N}_{\tau\tau}) = \frac{L(\vec{N}_{ZZ},\vec{N}_{WW},\vec{N}_{\tau\tau}|\mu,\hat{\hat{\theta}})}{L(\vec{N}_{ZZ},\vec{N}_{WW},\vec{N}_{\tau\tau}|\hat{\mu},\hat{\theta})}$$



$$p(H_\mu) = \int_{\lambda_{obs}}^{\infty} f(\lambda|H_\mu)\,d\lambda = \ldots$$

The idea behind the design of RooFit/RooStats/HistFactory

- Modularity, Generality and flexibility
- **Step 1 – Construct the likelihood function L(x|p)**

RooFit, or RooFit+HistFactory

- **Step 2 – Statistical tests on parameter of interest p**

  Procedure can be Bayesian, Frequentist, or Hybrid),
  but always based on L(x|p)

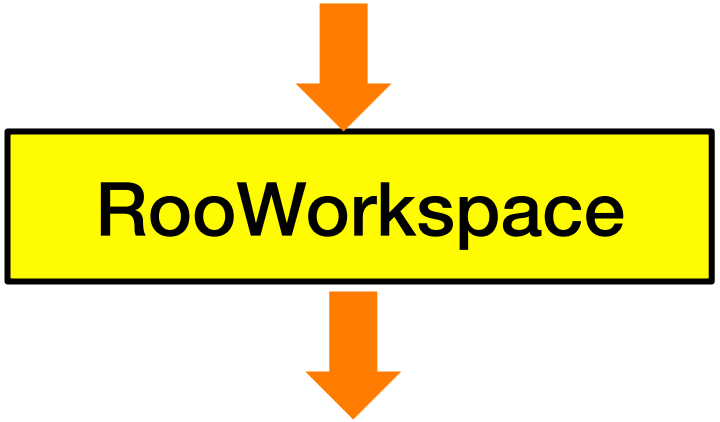
RooStats

- Steps 1 and 2 are conceptually separated,
  and in Roo* suit also implemented separately.

The idea behind the design of RooFit/RooStats/HistFactory

- Steps 1 and 2 can be 'physically' separated (in time, or user)
- **Step 1** – Construct the likelihood function $L(x|p)$

RooFit, or RooFit+HistFactory



Complete description of likelihood model, persistable in ROOT file

(RooFit pdf function)

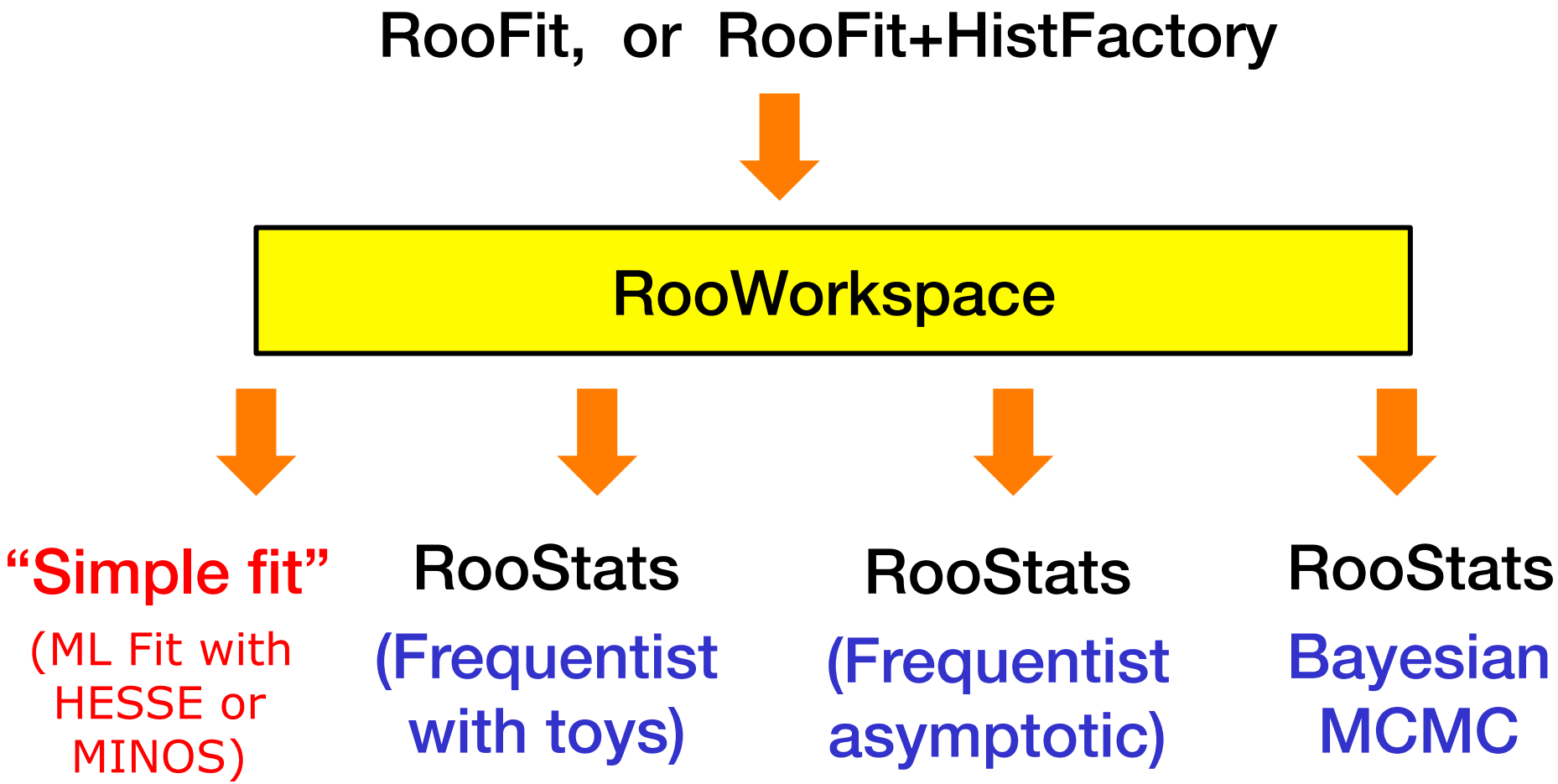
Allows full introspection and a-posteriori editing

- **Step 2** – Statistical tests on parameter of interest p

RooStats

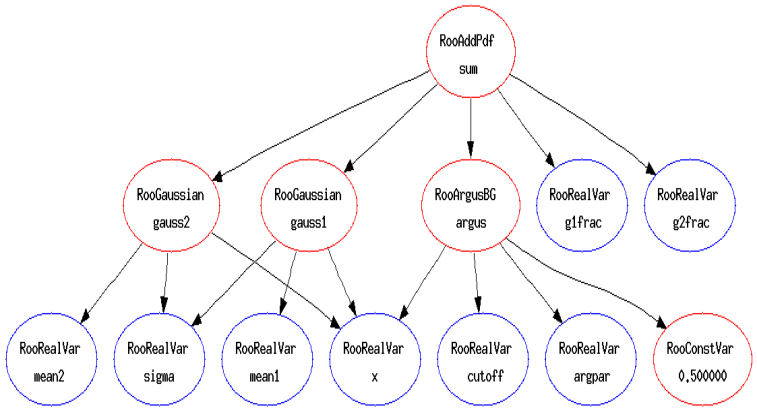
The benefits of modularity

- Perform different statistical test on exactly the same model

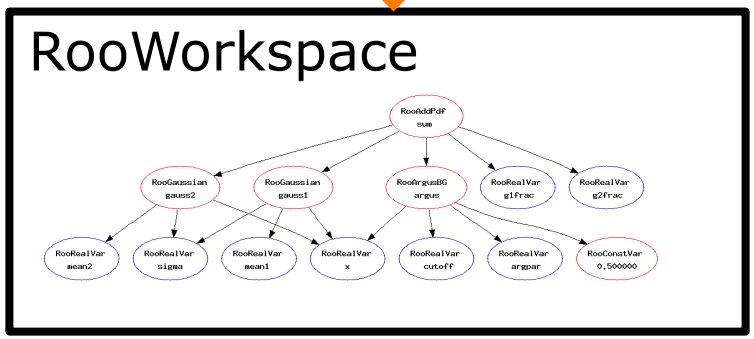


The workspace

- The workspace concept has revolutionized the way people share and combine analysis
  - Completely factorizes process of building and using likelihood functions
  - You can give somebody an analytical likelihood of a (potentially very complex) physics analysis in a way to the easy-to-use, provides introspection, and is easy to modify.

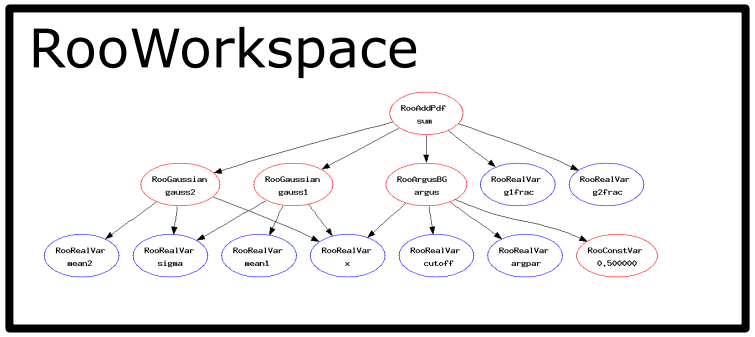


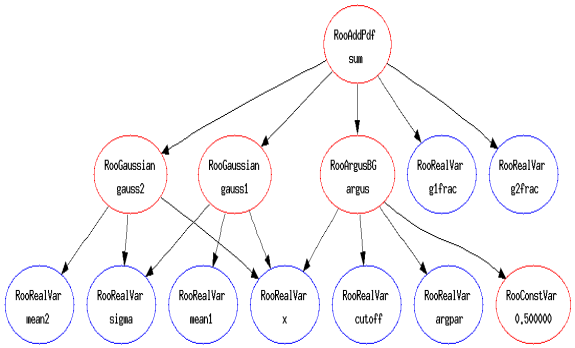

```
RooWorkspace w("w") ;
w.import(sum) ;
w.writeToFile("model.root") ;
```



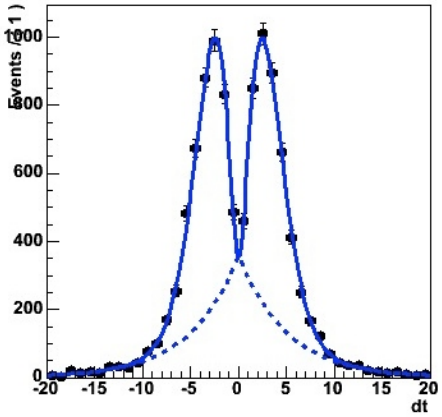
model.root

Using a workspace





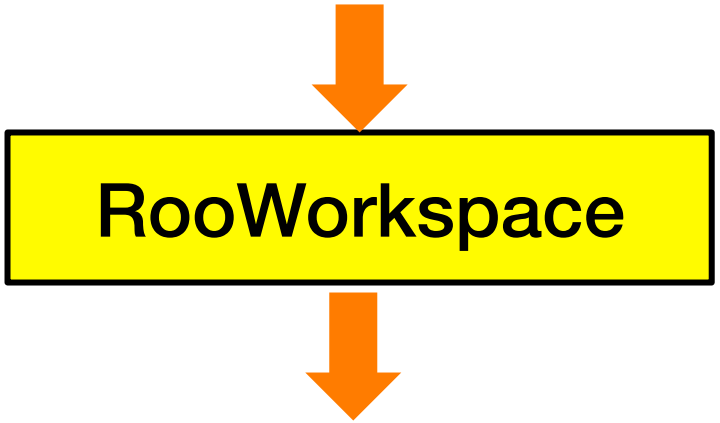

```
// Resurrect model and data
TFile f("model.root") ;
RooWorkspace* w = f.Get("w") ;
RooAbsPdf* model = w->pdf("sum") ;
RooAbsData* data = w->data("xxx") ;

// Use model and data
model->fitTo(*data) ;
RooPlot* frame =
          w->var("dt")->frame() ;
data->plotOn(frame) ;
model->plotOn(frame) ;
```



The idea behind the design of RooFit/RooStats/HistFactory

- **Step 1** – Construct the likelihood function *L(x|p)*

```
RooWorkspace w("w") ;
w.factory("Gaussian::sig(x[-10,10],m[0],s[1])") ;
w.factory("Chebychev::bkg(x,a1[-1,1])") ;
w.factory("SUM::model(fsig[0,1]*sig,bkg)") ;
w.writeToFile("L.root") ;
```



Complete description of likelihood model, persistable in ROOT file

(RooFit pdf function)

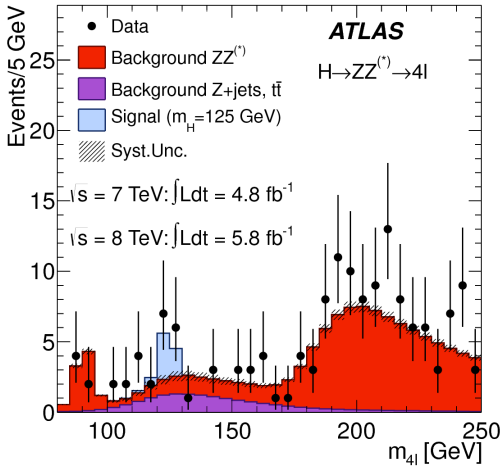
Allows full introspection and a-posteriori editing

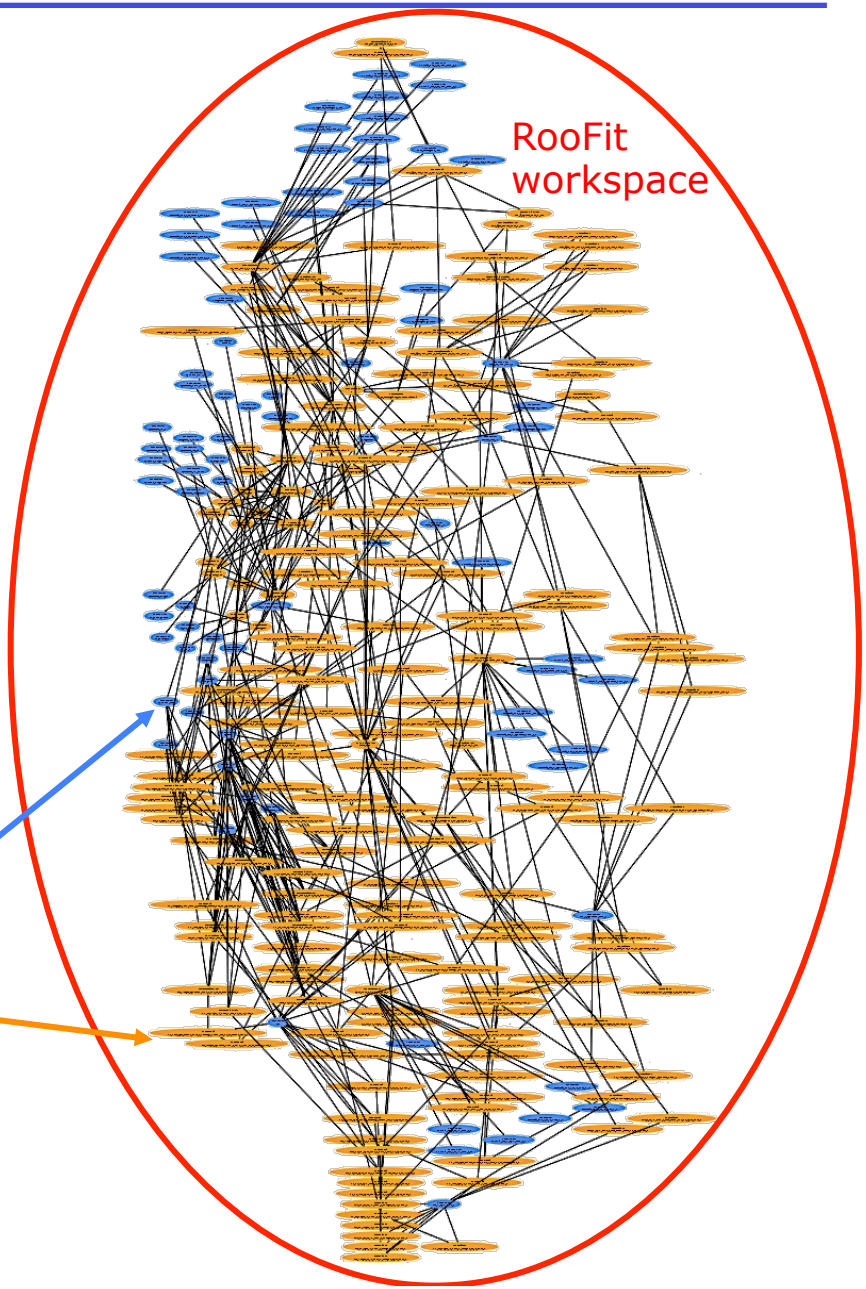
- **Step 2** – Statistical tests on parameter of interest *p*

```
RooWorkspace* w=TFile::Open("L.root")->Get("w") ;
RooAbsPdf* model = w->pdf("model") ;
pdf->fitTo(data) ;
```

Example RooFit component model for realistic Higgs analysis

Likelihood model describing the ZZ invariant mass distribution *including all possible systematic uncertainties*

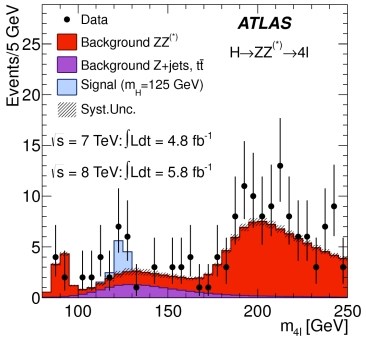


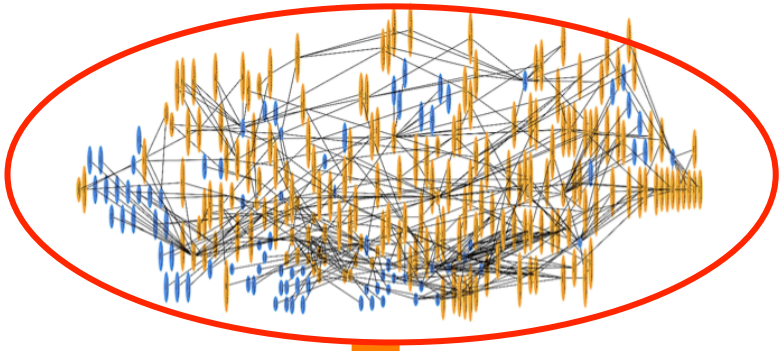


Graphical illustration of function components that call each other

Analysis chain identical for highly complex (Higgs) models

- **Step 1** – Construct the likelihood function *L(x|p)*





RooWorkspace

Complete description of likelihood model, persistable in ROOT file (RooFit pdf function)

Allows full introspection and a-posteriori editing

- **Step 2** – Statistical tests on parameter of interest *p*

```
RooWorkspace* w=TFile::Open("L.root")->Get("w") ;
RooAbsPdf* model = w->pdf("model") ;
pdf->fitTo(data,
           GlobalObservables(w->set("MC_GlObs"),
           Constrain(*w->st("MC_NuisParams") ;
```

Workspaces power collaborative statistical modelling

- Ability to persist complete(*) Likelihood models has profound implications for HEP analysis workflow
  - (*) Describing signal regions, control regions, and including nuisance parameters for all systematic uncertainties)
- **Anyone with ROOT (and one ROOT file with a workspace) can re-run any entire statistical analysis out-of-the-box**
  - About 5 lines of code are needed
  - Including estimate of systematic uncertainties
- Unprecedented new possibilities for cross-checking results, in-depth checks of structure of analysis
  - Trivial to run variants of analysis (what if 'Jet Energy Scale uncertainty' is 7% instead of 4%). Just change number and rerun.
  - But can also make structural changes a posteri. For example, rerun with assumption that JES uncertainty in forward and barrel region of detector are 100% correlated instead of being uncorrelated.

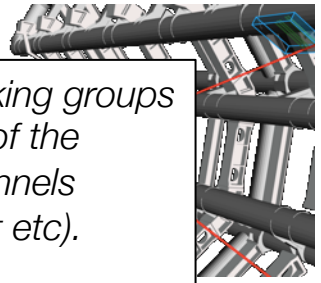
Collaborative statistical modelling

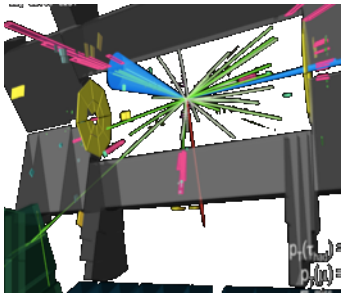
- As an experiment, you can effectively **build a library of measurements**, of which the full likelihood model is preserved for later use
  - Already done now, experiments have such libraries of workspace files,
  - Archived in AFS directories, or even in SVN….
  - Version control of SVN, or numbering scheme in directories allows for easy validation and debugging as new features are added

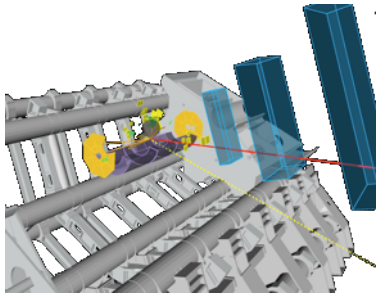
- Building of combined likelihood models greatly simplified.
  - Start from persisted components. No need to (re)build input components.
  - No need to know how individual components were built, or are internally structured. Just need to know meaning of parameters.
  - Combinations can be produced (much) later than original analyses.
  - Even analyses that were never originally intended to be combined with anything else can be included in joint likelihoods at a later time

Higgs **discovery** strategy – add everything together

H→ZZ→llll H→ττ H→WW→μνjj



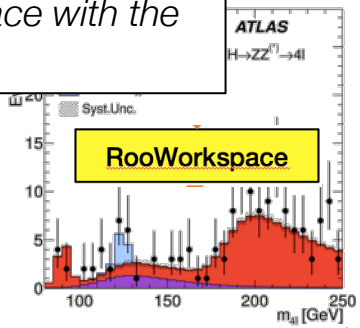


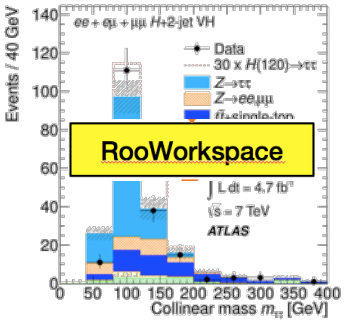


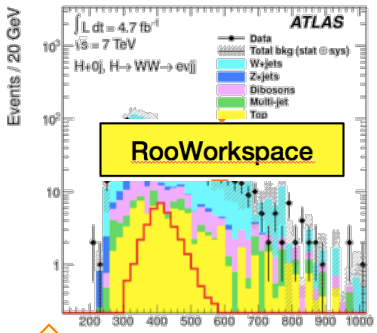
+ …

Dedicated physics working groups define search for each of the major Higgs decay channels (H→WW, H→ZZ, H→ττ etc).

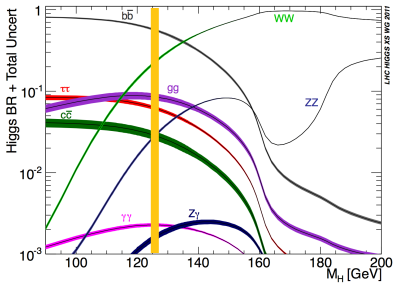
Output is physics paper or note, and a RooFit workspace with the full likelihood function







Assume SM rates



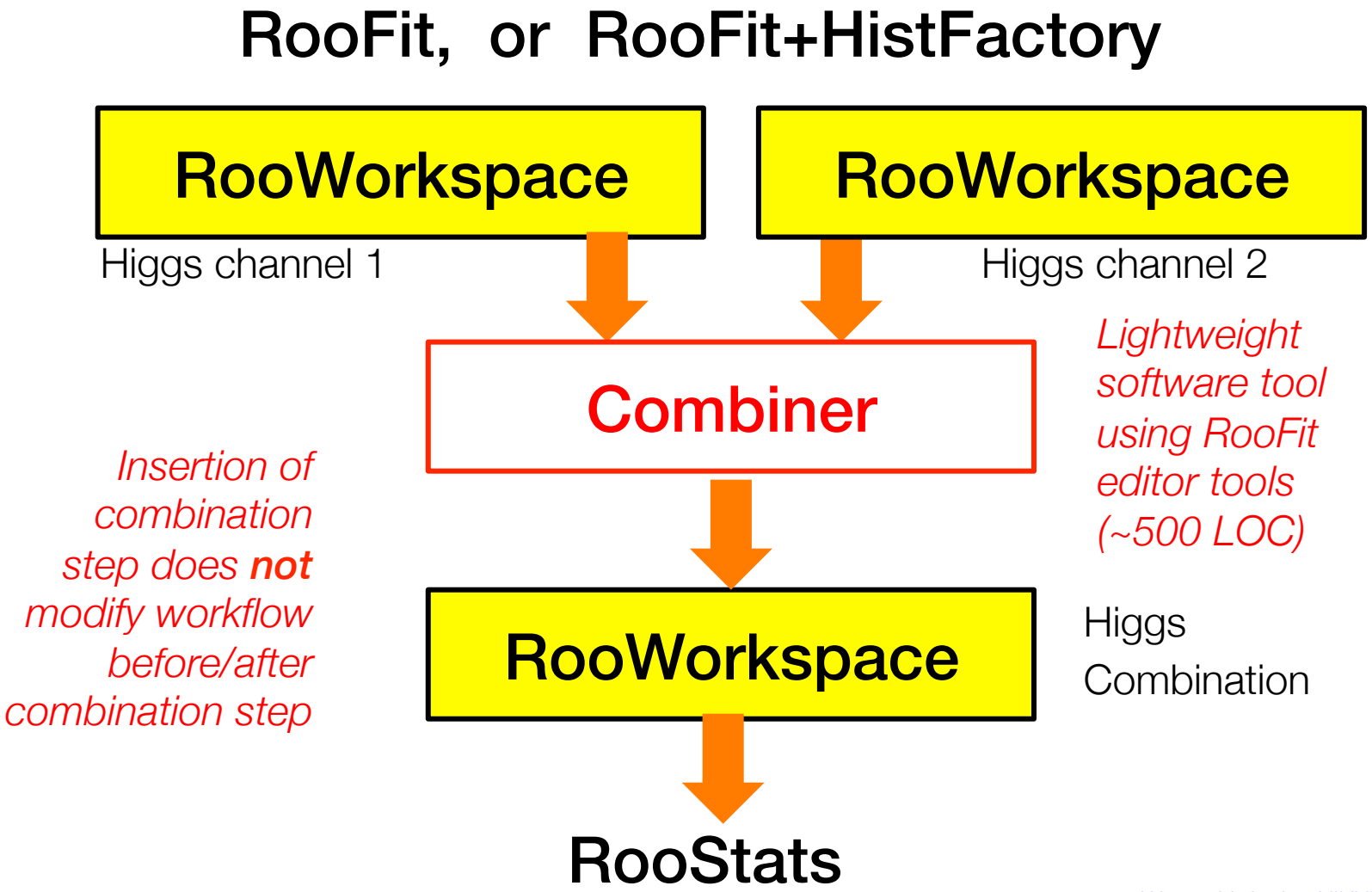
RooWorkspace

$$L(\mu,\vec{\theta}) = L_{H\to WW}(\mu_{WW},\vec{\theta})\cdot L_{H\to\gamma\gamma}(\mu_{\gamma\gamma},\vec{\theta})\cdot L_{H\to ZZ}(\mu_{ZZ},\vec{\theta})\cdot\ldots$$

A small dedicated team of specialists builds a combined likelihood from the inputs. Major discussion point: naming of parameters, choice of parameters for systematic uncertainties (a physics issue, largely)

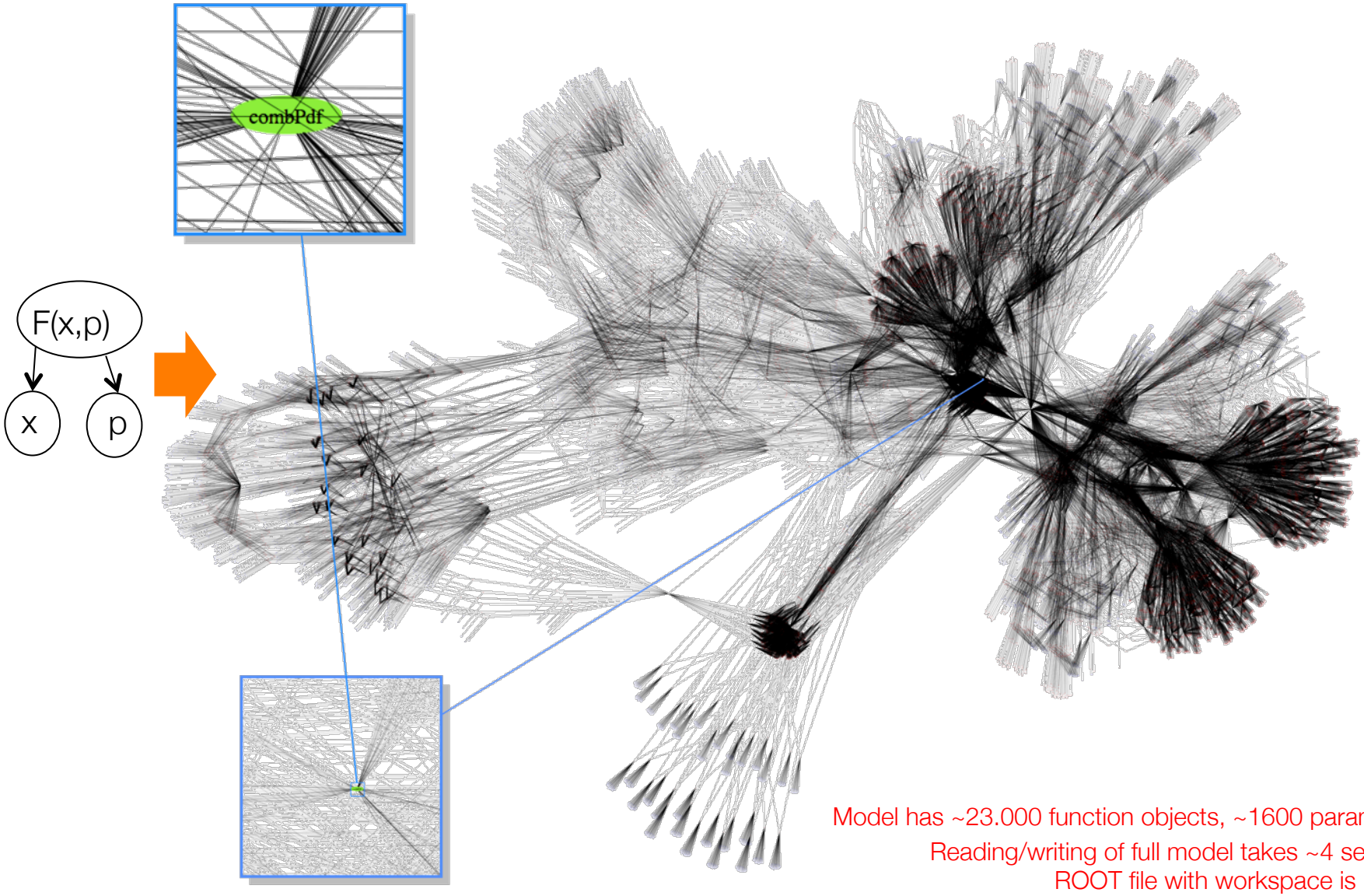
The benefits of modularity

- Technically very straightforward to combine measurements



Workspace persistence of *really* complex models works too!

Atlas Higgs combination model (23.000 functions, 1600 parameters)



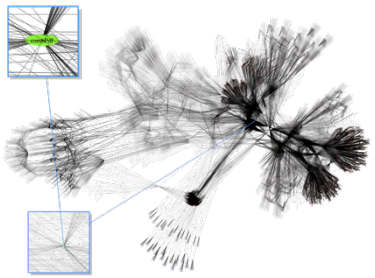
Model has ~23.000 function objects, ~1600 parameters

Reading/writing of full model takes ~4 seconds

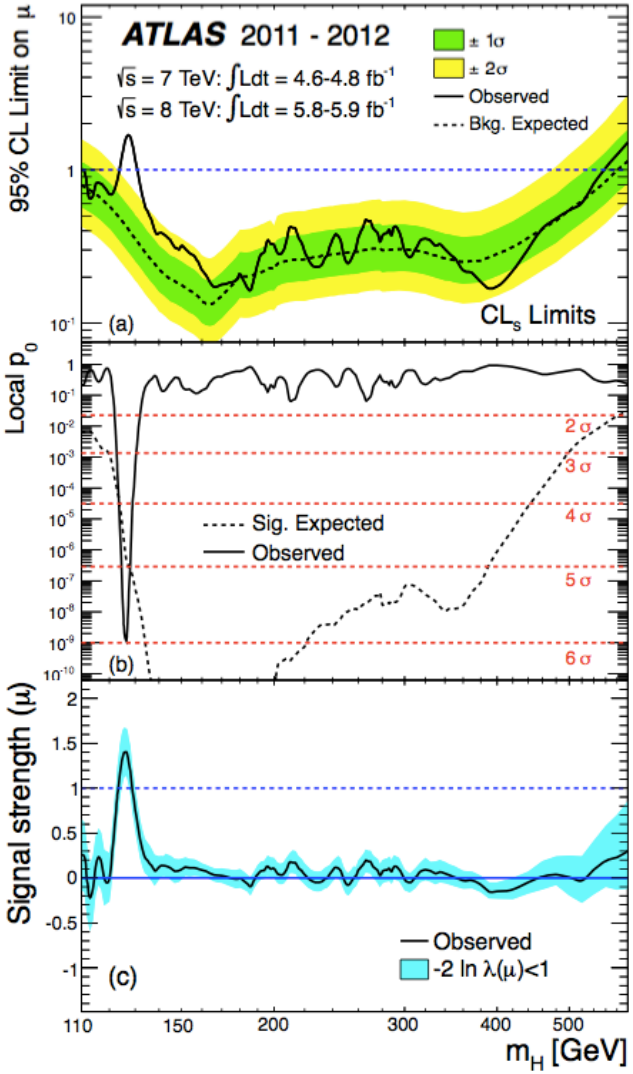
ROOT file with workspace is ~6 Mb

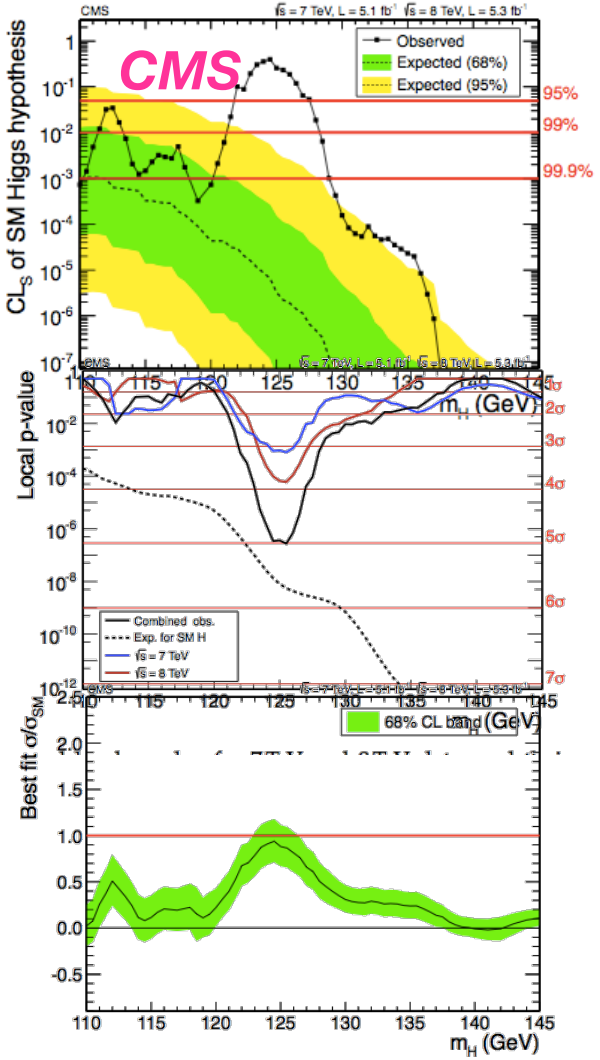
With these combined models the Higgs discovery plots were produced…

$L_{ATLAS}(\mu,\theta)$ =



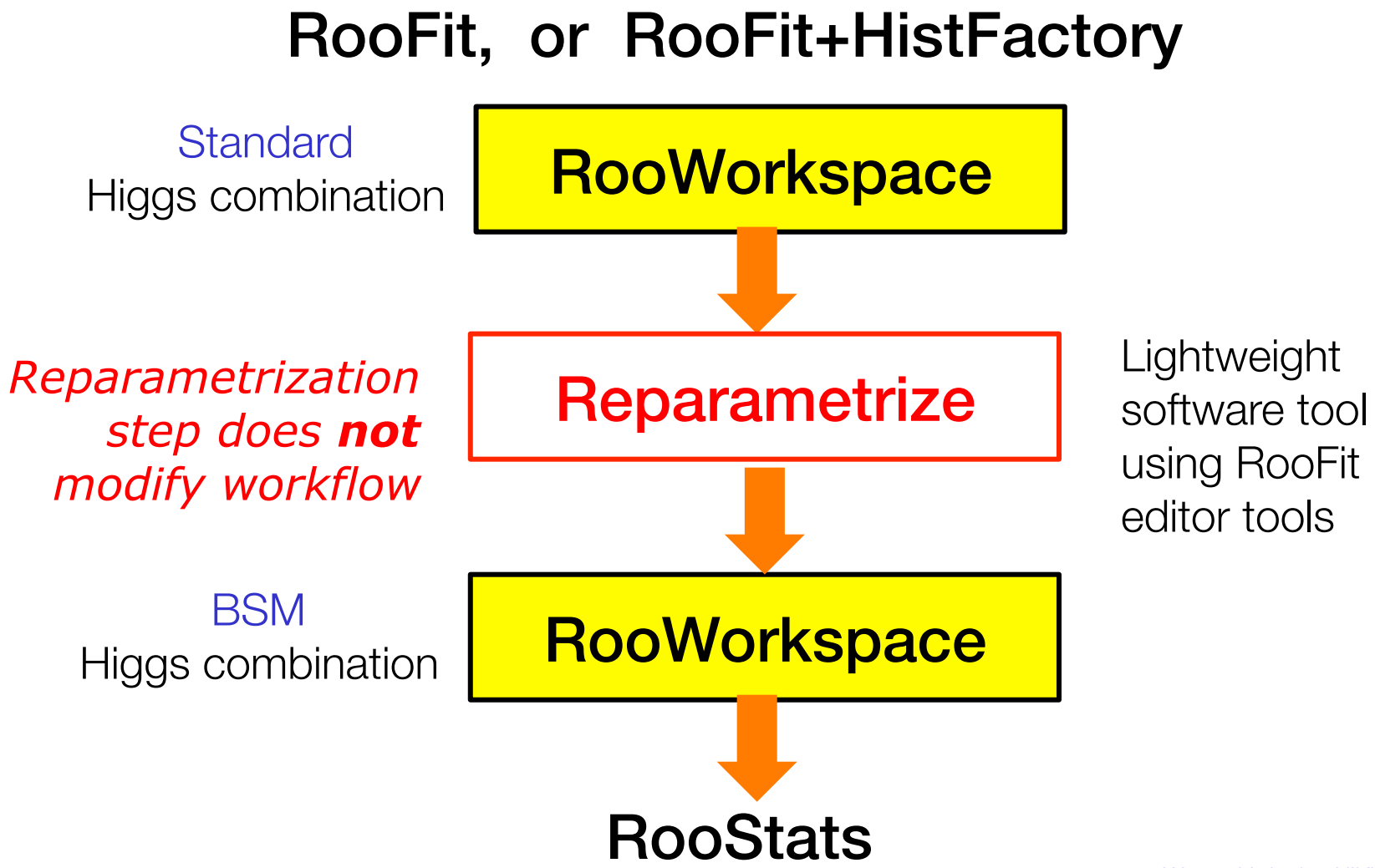
Neyman construction with profile likelihood ratio test





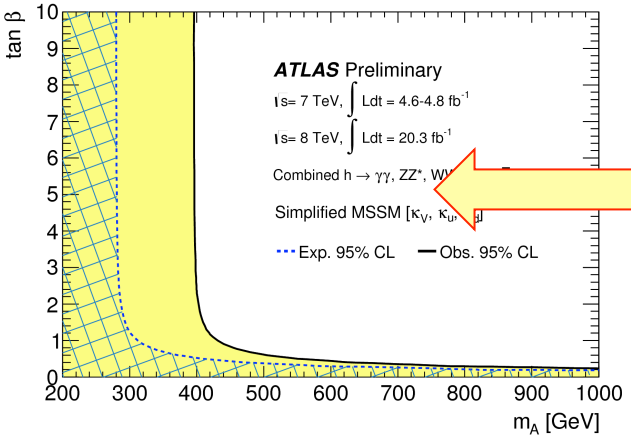
More benefits of modularity

- Technically very straightforward to reparametrize measurements

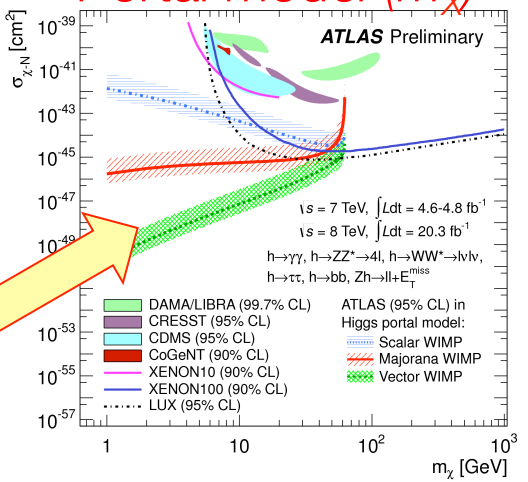


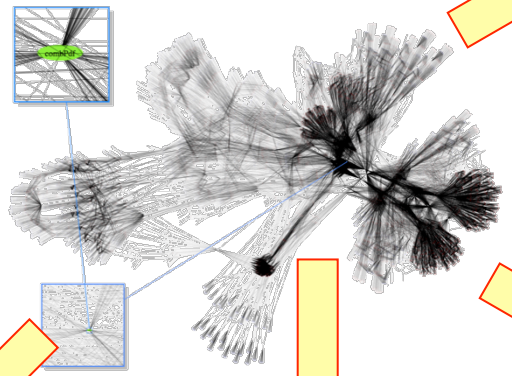
BSM Higgs constraints from reparametrization of SM Higgs Likelihood model

Simplified MSSM ($tan\beta, m_A$)

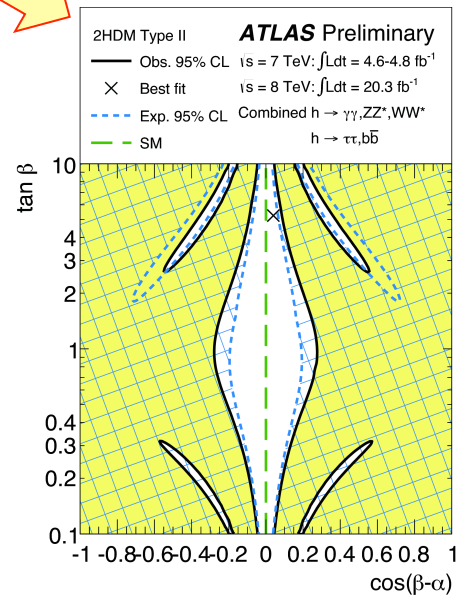


Portal model (m_χ)

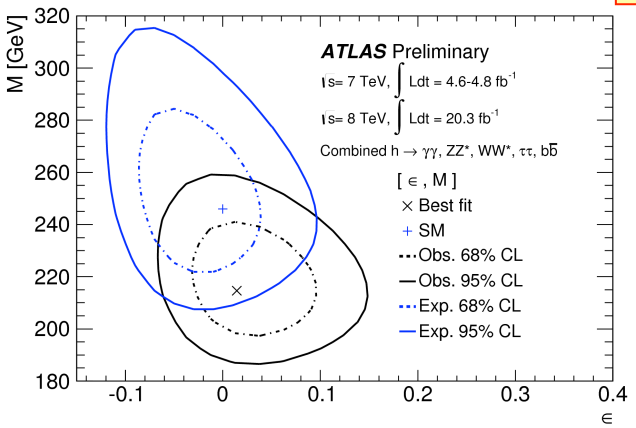




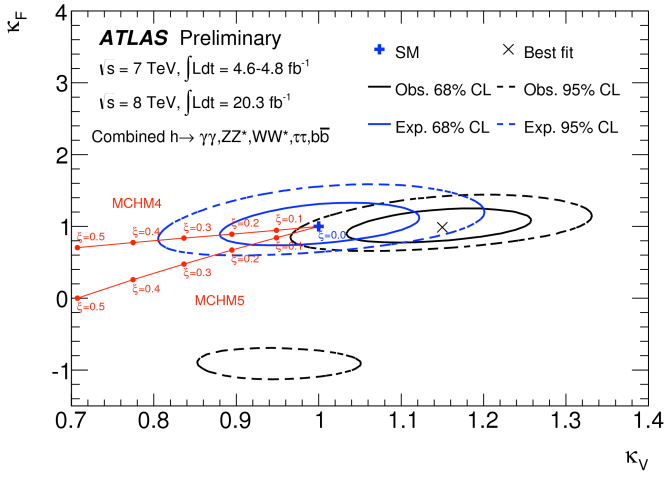
Two Higgs Double Model ($tan\beta, cos(\alpha-\beta)$)



Imposter model(M, ε)



Minimal composite Higgs(ξ)

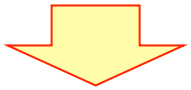


(ATLAS-CONF-2014-010)

An excursion – Collaborative analyses with workspaces

- *How can you reparametrize existing Higgs likelihoods in practice?*
- Write functions expressions corresponding to new parameterization

$$\sigma(gg \to H) * \mathrm{BR}(H \to \gamma\gamma) \sim \frac{\kappa_F^2 \cdot \kappa_\gamma^2(\kappa_F, \kappa_V)}{0.75 \cdot \kappa_F^2 + 0.25 \cdot \kappa_V^2}$$



```
w.factory("expr::mu_gg_func('(KF2*Kg2)/
                            (0.75*KF2+0.25*KV2)',
                            KF2,Kg2,KV2) ;
```

- Import transformation in workspace, edit *existing* model

```
w.import(mu_gg_func) ;
w.factory("EDIT::newmodel(model,mu_gg=mu_gg_gunc)") ;
```

RooWorkspace bonus – the model factory

- A nice bonus of the RooWorkspace as persistable model container is that one can write 'factory' tools that simplify model building

```
RooWorkspace w("w") ;

RooRealVar x("x","observable",-10,10) ;
RooRealVar mean("m","mean",-10,10) ;
RooRealVar sigma("s","sigma",-10,10) ;
RooGaussian g("g","gaus",x,m,s) ;
w.import(g) ;

RooDataSet* data = w.pdf("g")→
                   generate(*w.var("x"),1000) ;
RooFitResult* r = w.pdf("g").fitTo(*data) ;
```

RooWorkspace bonus – the model factory

- A nice bonus of the RooWorkspace as persistable model container is that one can write 'factory' tools that simplify model building

```
RooWorkspace w("w") ;

w.factory("Gaussian::g(x[-10,10],m[-10,10],s[3,0.1,10])") ;
```

```
RooDataSet* data = w.pdf("g")→
                   generate(*w.var("x"),1000) ;
RooFitResult* r = w.pdf("g").fitTo(*data) ;
```

Factory and Workspace

- This is *not* the same as reinventing Mathematica!
  String constructs an expression in terms of C++ objects, rather than being the expression
  - Objects can be tailored after construction through object pointers
  - For example: tune parameters and algorithms of numeric integration to be used with a given object
- Implementation in RooFit:
  Factory makes objects, Workspace owns them

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

w.Print("t") ;
variables
---------
(mean,sigma,x)

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

Accessing the workspace contents

- Contents can be accessing in two ways

- Through C++ namespace corresponding through w'space
  - Super easy (NB: does not always work on MS Windows)
  - But works in ROOT interpreted macros only

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

w::g.Print() ;
```

- Through accessor methods
  - A bit more clutter, but 100% ISO compliant C++ (and compilable)

```
RooAbsPdf* g = w.pdf("g") ;
RooRealVar* x = w.var("x") ;
```

Factory language

- The factory language has a 1-to-1 mapping to the constructor syntax of RooFit classes
  - With a few handy shortcuts for variables

- Creating variables

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

- Creating pdfs (and functions)

```
Gaussian::g(x,mean,sigma) → RooGaussian("g","g",x,mean,sigma)
Polynomial::p(x,{a0,a1}) → RooPolynomial("p","p",x",RooArgList(a0,a1));
```

  - Can always omit leading 'Roo'
  - Curly brackets translate to set or list argument (depending on context)

Factory language

- Composite expression are created by nesting statements
  - No limit to recursive nesting

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

- You can also use numeric constants whenever an unnamed constant is needed

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

- Names of nested function objects are optional
  - SUM syntax explained later

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

Factory language

- Interpreted function expressions allow to customize existing probability density functions

```
// construct Nexp=mu*S+B (a function)
expr::Nexp('mu*S+B',mu[0,5],S[50],B[50])

// construct a Poisson probability model describing
// the distribution of Nobs given Nexp
Poisson::p(Nobs[0,1000],Nexp) ;
```

- Generally: types starting with upper-case are Probability Density Functions, types starting with lower-case are simple functions
  - 'expr' is a special function type that implements an interpreted C++ function

Hands-on part

Goals of this hands-on session

1. Learn basics of RooFit model building
   - Learn to use to workspace factory to quickly specify models
   - Focus on building analytical models this afternoon
2. Short introduction to RooFit/RooStats interfacing
   - Demonstration of concept of separate model building from model analysis

- We only have ~2 hours – so tailored approach to that
  - First some introductory slides to familiarize you with the syntax of RooFit model building and RooFit model usage
  - Prepared macros that are fully functional and execute progressively complex tasks
  - You start from a functional working point – goal of your exercise time is to understand what they do and how they do it and work on some extensions and modifications of the macros

Exercises

- A series of 6 exercises/demonstrations is prepared for you
- To access these:
- Login to lxplus7.cern.ch (don't omit the 7!)
  - Create a directory insights/ in your home directory
  - Copy all files in ~verkerke/public/insights to your own insights directory
- To run them
  - Set up the correct ROOT version (6.14) by sourcing the setup script:
    unix> source setup_root_lxplus7.sh
  - Run them either from the unix command line, e.g
    unix> root –l ex01_Gaussian.C

    or from the root prompt
    root> .x ex01_Gaussian.C
  - The step-by-step instructions of each exercise are at the top of the file,
    the code in the file is generously sprinkled with helpful pedagogical comments
- Good luck and don't hesitate to ask for help, or ask questions in general!

Further practical information

- Most useful (in my experience) tutorial macros
  - In every ROOT installation, in directory $ROOTSYS/tutorials/roofit you will find 86 tutorial macros each demonstrating one important task or feature of RooFit

```
rf101_basics.C
rf102_dataimport.C
rf103_interprfuncs.C
rf104_classfactory.C
rf105_funcbinding.C
rf106_plotdecoration.C
rf107_plotstyles.C
rf108_plotbinning.C
rf109_chi2residpull.C
rf110_normintegration.C
rf111_derivatives.C
rf201_composite.C
rf202_extendedmlfit.C
rf203_ranges.C
rf204_extrangefit.C
rf205_compplot.C
rf206_treevistools.C
rf207_comptools.C
rf208_convolution.C
rf209_anaconv.C
rf210_angularconv.C
rf211_paramconv.C
rf301_composition.C
rf302_utilfuncs.C
rf303_conditional.C
rf304_uncorrprod.C
rf305_condcorrprod.C
rf306_condpereventerrors.C
rf307_fullpereventerrors.C
rf308_normintegration2d.C
rf309_ndimplot.C
rf310_sliceplot.C
rf311_rangeplot.C
rf312_multirangefit.C
rf313_paramranges.C
rf314_paramfitrange.C
rf315_projectpdf.C
rf316_llratioplot.C
rf401_importttreethx.C
rf402_datahandling.C
rf403_weightedevts.C
rf404_categories.C
rf405_realtocatfuncs.C
rf406_cattocatfuncs.C
rf407_latextables.C
rf501_simultaneouspdf.C
rf502_wspacewrite.C
rf503_wspaceread.C
rf504_simwstool.C
rf505_asciicfg.C
rf506_msgservice.C
rf507_debugtools.C
rf508_listsetmanip.C
rf509_wsinteractive.C
rf510_wsnamedsets.C
rf511_wsfactory_basic.C
rf512_wsfactory_oper.C
rf513_wsfactory_tools.C
rf601_intminuit.C
rf602_chi2fit.C
rf603_multicpu.C
rf604_constraints.C
rf605_profilell.C
rf606_nllerrorhandling.C
rf607_fitresult.C
rf608_fitresultaspdf.C
rf609_xychi2fit.C
rf610_visualerror.C
rf701_efficiencyfit.C
rf702_efficiencyfit_2D.C
rf703_effpdfprod.C
rf704_amplitudefit.C
rf705_linearmorph.C
rf706_histpdf.C
rf707_kernelestimation.C
rf708_bphysics.C
rf709_momentmorph.C
rf709_momentmorph1.C
rf710_momentmorph2.C
rf801_mcstudy.C
rf802_mcstudy_addons.C
rf803_mcstudy_addons2.C
rf804_mcstudy_constr.C
rf901_numintconfig.C
rf902_numgenconfig.C
rf903_numintcache.C
```

- Class reference on ROOT website
  - Most useful for syntax of key classes like RooAbsPdf, RooWorkspace etc…